INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA
E TECNOLOGIA DE SÃO PAULO

RODRIGO DOS SANTOS BARBOSA

SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

São Paulo
2017

RODRIGO DOS SANTOS BARBOSA

SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

Monografia de Conclusão de Curso apresentado ao Programa de Pós-graduação Lato Sensu do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, como requisito parcial para a obtenção do Título de Especialista em Gestão de TI.

Orientador: Prof. Me. André Evandro Lourenço

São Paulo

2017

Catalogação na fonte
Biblioteca Francisco Montojos - IFSP Campus São Paulo
Dados fornecidos pelo(a) autor(a)

B238s Barbosa, Rodrigo dos Santos
Seleção de métodos de desenvolvimento de software a partir das características do projeto / Rodrigo dos Santos Barbosa. São Paulo: [s.n.], 2017.
75 f. il.

Orientador: Prof. Me. André Evandro Lourenço

Monografia (Especialização em Gestão da Tecnologia da Informação) - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, IFSP, 2017.

1. Engenharia de Software. 2. Metodologia de Desenvolvimento de Software. 3. Desenvolvimento de Software. 4. Metodologia Ágil. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo II. Título.

CDD 658.404

Agradeço à DEUS, por me favorecer com a enorme benção de ingressar no curso e por me capacitar com resiliência e longanimidade ao longo de toda jornada!

AGRADECIMENTOS

Ao Prof. Me. André Evandro Lourenço pelo direcionamento e suporte;

À minha esposa Cristiane Rios e aos meus filhos Gabriel e Giovanna por me apoiarem o tempo todo;

À minha irmã Cristiane Barbosa que me indicou o curso;

Aos meus pais pelas bases de educação, valores e princípios morais que me deram;

Ao meu gestor profissional Eliseu Pereira de Lima que me ajudou na escolha do tema para o trabalho;

A todos os meus colegas de classe pelo companheirismo e amizade durante toda a jornada do curso;

A todos que, direta ou indiretamente, colaboraram na execução deste trabalho.

Apega-te à instrução, não a largues;
guarda-a, porque ela é a tua vida.

Provérbios 4:13

RESUMO

Os métodos de desenvolvimento de *software* foram criados como ferramenta de apoio à engenheiros e desenvolvedores. Diversos métodos foram criados, no entanto a falta de critérios que apoiem os profissionais na seleção daquele que se encaixe nas características do projeto a ser desenvolvido faz com que as escolhas sejam feitas a partir de crenças ou preferências pessoais. O objetivo deste trabalho é identificar critérios direcionadores na seleção de métodos de desenvolvimento de *software* a partir das características de um projeto e desenvolver uma aplicação parametrizável que indique os métodos a partir do preenchimento de um questionário. O trabalho foi desenvolvido como pesquisa exploratória e qualitativa com o levantamento bibliográfico do conhecimento disponível a partir das teorias publicadas em livros ou obras congêneres. Resultados do trabalho mostram que um método pode ser selecionado considerando o cruzamento de suas características com as características específicas do projeto a ser desenvolvido através de uma aplicação parametrizável.

Palavras-chave**:** Engenharia de *Software*. Metodologia de desenvolvimento de *software*. Desenvolvimento de *software.* Metodologia Ágil.

ABSTRACT

Software development methods were created as a tool to support engineers and developers. Several methods have been created, however, the lack of criteria that support the professionals in selecting the one that fits the characteristics of the project to be developed causes how choices are made from beliefs or preferences. The objective of this work is to identify the guiding criteria in the selection of methods of software development from the characteristics of a project and to develop a parameterizable application that indicates methods from the completion of a questionnaire. The work was developed as an exploratory and qualitative research with the bibliographical survey of the knowledge available from theories published in books or similar works. Results of the work show that a method can be selected considering the crossing of its characteristics with the specific characteristics of the project to be developed through a parameterizable application.

Keywords: Software Engineering. Methodology of software development. Software development. Agile Methodology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

ASD	*Adaptive software development*
BOK	*Book of Knowledge*
IBM	*International Business Machines*
IRUP	IBM Rational *Unified Process*
RUP	*Rational unified process*
UML	*Unified Modeling Language*
XP	*Extreme programming*

SUMÁRIO

1. INTRODUÇÃO

Quando um cliente solicita o desenvolvimento de um *software* espera receber um produto que atende às suas expectativas, que não exceda o orçamento previsto e que seja entregue dentro da data prevista, e essas expectativas tocam em pontos críticos do projeto de *software*: escopo, custo e prazo. Até o início dos anos 70 os desenvolvedores não tinham um método definido para a produção de *software*, então Winston W. Royce propôs em 1970 através de seu artigo “Gerenciando o desenvolvimento de grandes sistemas de software” o método que ficou conhecido como “modelo em cascata”.

Na proposta de Royce o escopo é totalmente definido antes do início do desenvolvimento (ROYCE, 1970), o que o caracteriza como um ciclo de vida predeterminado com escopo, prazo e custos definidos o mais cedo possível para atingir a expectativa do cliente (PMI, 2013). O grande problema é que há situações em que não é possível prever todo o escopo do *software* desde o início e o custo de alterações fica maior a cada avanço do processo, e pensando nisso outros engenheiros criaram métodos com ciclo de vida iterativos e incrementais em que as fases repetem atividades conforme a compreensão do produto aumenta (PMI, 2013).

Na década de 90 surgiram os chamados métodos ágeis com a proposta de atender cenários de mudança constante utilizando ciclos de vida adaptativos caracterizados por envolvimento contínuo dos interessados e iterações curtas (PMI, 2013). No contexto da engenharia de *software*, agilidade é a capacidade de reagir rapidamente a mudanças (PRESSMAN, 2011). O manifesto ágil prega uma comunicação mais fácil entre a equipe de projeto e o cliente, a diminuição da importância dos artefatos (exceto os essenciais) e a rápida disponibilização de um *software* operacional para o cliente (MANIFESTO, 2001).

Os métodos vieram para organizar o desenvolvimento de *software* e oferecer suporte aos desenvolvedores na solução de problemas de produção, mas com a grande variedade de opções e propostas a escolha do método a ser utilizado tornou-se algo pessoal, onde as crenças tomam o lugar dos fatos e passam a orientar as tomadas de decisão (PRESSMAN, 2011).

1.1. PROBLEMA DE PESQUISA

A cada projeto o desenvolvedor terá situações distintas e é possível que o método utilizado não ofereça suporte para a solução dos problemas que surgirão. Nesse momento voltamos no tempo e a insatisfação do cliente com o escopo, prazo e custo do projeto acontecem como antes da criação dos métodos. O gráfico 1, “Pesquisa sobre sucesso em projetos” mostra os resultados de uma pesquisa realizada com 365 gestores de TI de diversas empresas em setores variados da economia sobre 8.380 projetos:

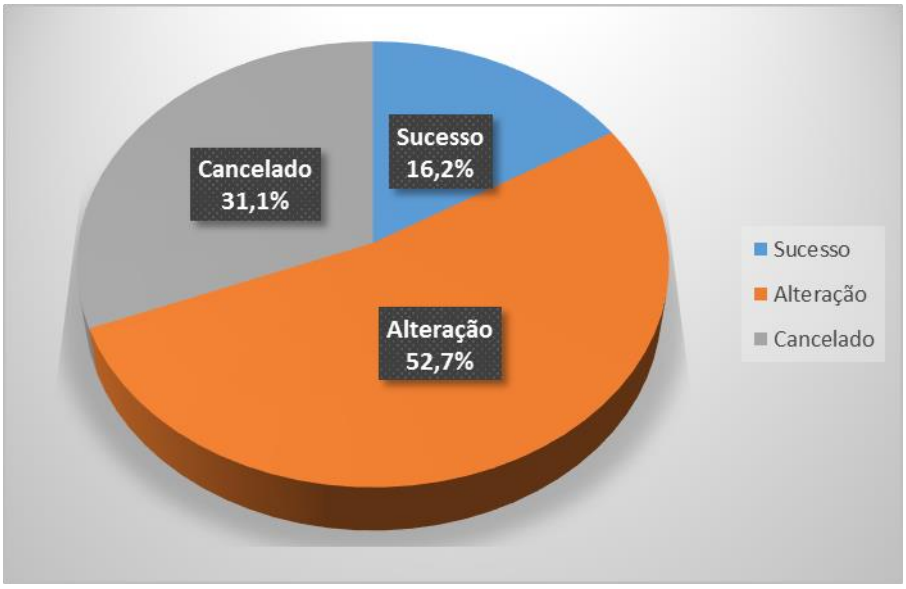

Gráfico 1 - Pesquisa sobre sucesso em projetos

Fonte: (The Standish Group Internacional, 2014)

A pesquisa mostra que apenas 16,2% dos projetos foram concluídos com sucesso, enquanto 52,7% foram concluídos com alterações e 31,1% foram cancelados (The Standish Group Internacional, 2014).

Se existem vários métodos com propostas de ciclo de vida distintas torna-se também importante considerar as características do projeto a ser desenvolvido antes de realizar uma escolha. Verifica-se então, a importância da existência de critérios racionais para a seleção de métodos de desenvolvimento de *software*. Deseja-se responder à pergunta: “Quais os critérios devem ser considerados na seleção de métodos de desenvolvimento de *software* considerando-se as características do projeto a ser realizado? ”

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Este trabalho tem por objetivo realizar um levantamento bibliográfico de alguns dos métodos de desenvolvimento de *software* existentes, analisar suas características e identificar critérios que sirvam como direcionadores na seleção de métodos baseado em informações prévias de um projeto a ser realizado.

1.2.2. Objetivos Específicos

Os objetivos específicos são: associar aos critérios direcionadores situações pré-definidas com base no levantamento das características dos métodos de desenvolvimento de *software* e ciclos de vida em projetos analisados; associar os métodos de desenvolvimento de *software* as situações pré-definidas de modo que as alternativas selecionadas levem a identificação de um ou mais métodos adequados para as condições do projeto proposto; desenvolver uma aplicação que faça a indicação de métodos de desenvolvimento de *software* a partir do preenchimento de um formulário com os critérios direcionadores.

1.3. JUSTIFICATIVA

Com o crescimento do número de métodos de desenvolvimento de *software*, engenheiros e desenvolvedores de *software* de todo o mundo tem opções de escolha na hora de decidir por um método de desenvolvimento de *software*. Os métodos possuem atividades, processos e princípios diferentes uns dos outros, o que torna a escolha uma questão racional para se identificar qual é o mais adequado para atender as necessidades do seu projeto e não uma atividade de simples escolha. Em vez disso a seleção poderá considerar fatores contextuais como as características do projeto (VIJAYASARATHY, et al., 2016).

Cada projeto será diferente do outro e apresentará necessidades distintas. A utilização de um método para desenvolvimento de *software* independente das características do projeto limitará a equipe de desenvolvimento as situações previstas por tal método, expondo a equipe

e o cliente a problemas que poderiam ter sido previstos caso houvesse uma análise de aderência do método diante de tais possibilidades.

Para que seja possível tornar a escolha do método de desenvolvimento de *software* numa análise racional é necessário que existam critérios que direcionem o desenvolvedor na previsão de situações de seu projeto e com isso identificar aqueles que possuam processos e atividades que suportem tais necessidades, evitando assim problemas previsíveis e aumentando as chances de atender as necessidades de seu cliente.

Dessa forma, convém realizar um estudo sobre métodos de desenvolvimento de *software* existentes explorando suas características de processo, aplicações, vantagens e desvantagens para que se consiga obter critérios que levem a um método recomendado para uma necessidade específica.

1.4. ESTRUTURA DO TRABALHO

O capítulo 1, introdução, apresenta o problema, os objetivos e a justificativa. O capítulo 2, metodologia, apresenta o tipo de pesquisa realizada e o método aplicado no levantamento bibliográfico. O capítulo 3, revisão da literatura, apresenta o conceito de modelo de processo genérico da engenharia de *software*, cinco métodos de desenvolvimento de *software* existentes mostrando suas características, princípios e atividades e os ciclos de vida em projeto associados aos métodos analisados. O capítulo 4, seleção de métodos de desenvolvimento de *software* a partir das características do projeto, analisa cada um dos métodos de desenvolvimento de *software* apresentados no capítulo 3, revisão da literatura, explorando suas peculiaridades e identificando uma lista de critérios direcionadores bem como uma associação destes aos métodos de desenvolvimento de *software*. Apresenta também uma aplicação que indica métodos de desenvolvimento de *software* a partir do preenchimento de um formulário com os critérios direcionadores. O capítulo 5, considerações finais, apresenta uma revisão da pesquisa realizada, além de mostrar suas contribuições e propor novos trabalhos a respeito do assunto.

2. METODOLOGIA

O tipo de pesquisa empregada foi a exploratória e qualitativa com levantamento bibliográfico, utilizando o conhecimento disponível em livros e obras congêneres com o objetivo de conhecer e analisar as contribuições teóricas sobre o tema utilizando-as como base para a construção de um modelo teórico explicativo. (KÖCHE, 2011).

A busca pelas referências foi totalmente realizada pela *internet* utilizando os seguintes repositórios:

- Periódicos CAPES (http://www.periodicos.capes.gov.br);
- ACM *Digital Library* (http://dl.acm.org);
- IEEE Xplore *Digital Library* (http://ieeexplore.ieee.org);

Adicionalmente, o mecanismo de busca Google (http://www.google.com.br) foi utilizado para localizar materiais específicos. O levantamento foi realizado em cinco etapas:

2.1.1. Etapa 1 – Busca por palavras-chave e seleção por título

Na primeira etapa foi feita uma consulta por palavras-chave em cada repositório e o critério de seleção dos resultados foi a análise do título. Foram selecionadas as obras cuja o título tivesse relação com a pesquisa. A tabela 1 mostra os resultados obtidos por repositório e palavra-chave:

REPOSITÓRIO	BUSCA		RESULTADOS		
	PALAVRA-CHAVE	DATA	OBTIDOS	SELECIONADOS	%
PERIÓDICOS CAPES	No título é exato a "desenvolvimento de *software*"	18/08/2016	382	13	3,40
PERIÓDICOS CAPES	No título é exato a "engenharia de *software*"	18/08/2016	277	3	1,08
PERIÓDICOS CAPES	No título é exato a "Metodologia Ágil"	18/08/2016	28	6	21,43
PERIÓDICOS CAPES	No título é exato a "Metodologia de desenvolvimento de *software*"	18/08/2016	5	1	20,00
ACM DL	No título é exato a "*agile methodology*" a partir de 2014	25/08/2016	88	8	9,09
ACM DL	No título é exato a "desenvolvimento de software"	25/08/2016	143	1	0,70
ACM DL	No título é exato a "*software development*" a partir de 2010	25/08/2016	131	23	17,56
ACM DL	No título é exato a "*software engineering*" a partir de 2016	25/08/2016	79	1	1,27
IEEE XPLORE DL	No título é exato a "*agile methodology*" a partir de 2010	08/09/2016	58	10	17,24
IEEE XPLORE DL	No título é exato a "*software development methodology*" a partir de 2010	08/09/2016	52	4	7,69
IEEE XPLORE DL	No título é exato a "*software development*" a partir de 2016	08/09/2016	79	4	5,06
IEEE XPLORE DL	No título é exato a "*software engineering*" a partir de 2016	08/09/2016	90	2	2,22
TOTALIZAÇÃO			**1.412**	**76**	**5,38**

Tabela 1 - Resultados obtidos por repositório e palavra-chave

No nível de detalhe total, a busca com o maior número de resultados foi no repositório "Periódicos CAPES" filtrando por "No título é exato a 'desenvolvimento de *software*'" com 382 resultados. A busca mais eficiente foi no mesmo repositório filtrando por "No título é exato a 'Metodologia Ágil'" com 21,43% de resultados selecionados.

Na tabela 2 temos a contabilização por repositório e por palavra-chave:

REPOSITÓRIO	RESULTADOS		
	TOTAL	SELECIONADOS	%
PERIÓDICOS CAPES	692	23	3,32
ACM DL	441	33	7,48
IEEE XPLORE DL	279	20	7,17
TOTALIZAÇÃO	**1.412**	**76**	**5,38**

Tabela 2 - Totalização por repositório

Verificando a totalização por repositório, a maior quantidade de resultados foi obtida no "Periódicos CAPES" com 692 resultados na soma de todas as palavras-chave. Já o mais eficiente foi o "ACM DL" com 7,48% de resultados selecionados.

PALAVRA-CHAVE	RESULTADOS		
	TOTAL	SELECIONADOS	%
METODOLOGIA ÁGIL	174	24	13,79
DESENVOLVIMENTO DE *SOFTWARE*	735	41	5,58
ENGENHARIA DE *SOFTWARE*	446	6	1,35
METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE*	57	5	8,77
TOTALIZAÇÃO	**1.412**	**76**	**5,38**

Tabela 3 - Totalização por palavra-chave

Na tabela 3 temos a totalização por palavra-chave, a maior quantidade de resultados foi obtida utilizando “desenvolvimento de *software*” com 735 resultados na soma de todos os repositórios. A mais eficiente foi “metodologia ágil” com 13,79% de resultados selecionados.

2.1.2. Etapa 2 – Análise do resumo

Na segunda etapa foi realizada a leitura do resumo de cada um dos itens selecionados na etapa anterior, verificando se o conteúdo poderia contribuir com o desenvolvimento da pesquisa. Os resultados são demonstrados na tabela 4:

REPOSITÓRIO	RESULTADOS			
	ANALISADOS	EXCLUÍDOS	SELECIONADOS	%
PERIÓDICOS CAPES	23	8	15	65,22
ACM DL	33	14	19	57,58
IEEE XPLORE DL	20	4	16	80,00
TOTALIZAÇÃO	**76**	**26**	**50**	**65,79**

Tabela 4 - Resultados obtidos por análise do resumo

Na análise do resumo a maior quantidade selecionada foi do repositório “ACM DL” com 19 itens. O mais eficiente foi do “IEEE XPLORE DL” com 80,00% de itens selecionados.

2.1.3. Etapa 3 – Análise completa das obras selecionadas

Na terceira etapa foi realizada uma análise completa das obras selecionadas, com o objetivo de verificar se o material tinha conteúdos que contribuiriam com a pesquisa. Os resultados são demonstrados na tabela 5:

REPOSITÓRIO	RESULTADOS			
	ANALISADOS	EXCLUÍDOS	SELECIONADOS	%
PERIÓDICOS CAPES	15	7	8	53,33
ACM DL	19	14	5	26,32
IEEE XPLORE DL	16	10	6	37,50
TOTALIZAÇÃO	**50**	**31**	**19**	**38,00**

Tabela 5 - Resultados obtidos por análise completa das obras selecionadas

Na análise completa a maior quantidade selecionada foi do repositório “PERIÓDICOS CAPES” com 8 itens. Este foi também o mais eficiente com 53,33% de itens selecionados.

2.1.4. Etapa 4 – Tabulação e identificação das principais referências bibliográficas

Na quarta etapa foi realizada uma tabulação das referências bibliográficas utilizadas a partir dos 23 itens selecionados, totalizando 978 referências. A obra com o maior número de citações foi “*Software Engineering 9th ed*” do autor Ian Somerville. Por autor verificou-se Kent Beck com 18 ocorrências distribuídas em 4 obras.

Após contabilização das obras referenciadas, realizou-se uma apuração para identificação das principais referências no assunto. No total, 15 referências foram identificadas. Após esse levantamento foi verificado em cada uma das 23 obras selecionadas se ao menos uma das 15 principais referências estava presente. Os resultados são demonstrados na tabela 6:

REPOSITÓRIO	RESULTADOS			
	ANALISADOS	EXCLUÍDOS	SELECIONADOS	%
PERIÓDICOS CAPES	8	0	8	100,00
ACM DL	5	0	5	100,00
IEEE XPLORE DL	6	2	4	66,67
TOTALIZAÇÃO	**19**	**2**	**17**	**89,47**

Tabela 6 - Resultados obtidos por análise das referências bibliográficas

2.1.5. Etapa 5 – Leitura completa das obras selecionadas

Na quinta etapa foi realizada uma leitura completa de todos os itens. Na análise foram consideradas a relevância da pesquisa, a abordagem do autor e relação do assunto com a pesquisa a ser realizada. Os resultados são demonstrados na tabela 7:

REPOSITÓRIO	RESULTADOS			
	ANALISADOS	EXCLUÍDOS	SELECIONADOS	%
PERIÓDICOS CAPES	8	4	4	50,00
ACM DL	5	1	4	80,00
IEEE XPLORE DL	4	0	4	100,00
TOTALIZAÇÃO	**17**	**5**	**12**	**70,59**

Tabela 7 - Resultados obtidos após leitura completa das obras

2.1.6. Resultado final

Após as cinco etapas, os resultados são os demonstrados na tabela 8:

REPOSITÓRIO	BUSCA		RESULTADOS		
	PALAVRA-CHAVE	DATA	ETAPA 1	ETAPA 5	%
PERIÓDICOS CAPES	No título é exato a "desenvolvimento de *software*"	18/08/2016	13	3	23,08
PERIÓDICOS CAPES	No título é exato a "engenharia de *software*"	18/08/2016	3	0	0,00
PERIÓDICOS CAPES	No título é exato a "Metodologia Ágil"	18/08/2016	6	1	16,67
PERIÓDICOS CAPES	No título é exato a "Metodologia de desenvolvimento de *software*"	18/08/2016	1	0	0,00
ACM DL	No título é exato a "*agile methodology*" a partir de 2014	25/08/2016	8	0	0,00
ACM DL	No título é exato a "desenvolvimento de software"	25/08/2016	1	0	0,00
ACM DL	No título é exato a "*software development*" a partir de 2010	25/08/2016	23	4	17,39
ACM DL	No título é exato a "*software engineering*" a partir de 2016	25/08/2016	1	0	0,00
IEEE XPLORE DL	No título é exato a "*agile methodology*" a partir de 2010	08/09/2016	10	2	20,00
IEEE XPLORE DL	No título é exato a "*software development methodology*" a partir de 2010	08/09/2016	4	2	50,00
IEEE XPLORE DL	No título é exato a "*software development*" a partir de 2016	08/09/2016	4	0	0,00
IEEE XPLORE DL	No título é exato a "*software engineering*" a partir de 2016	08/09/2016	2	0	0,00
TOTALIZAÇÃO			**76**	**12**	**15,79**

Tabela 8 – Manutenção de resultados por repositório e palavra-chave

No nível de detalhe total, o maior número de resultados mantidos foi no repositório "ACM DL" filtrando por "No título é exato a "'*software development*' a partir de 2010" com

4 obras mantidas. A manutenção mais eficiente foi no repositório “IEEE XPLORE DL” filtrando “No título é exato a "’*software development methodology*’ a partir de 2010” com 50,0% de resultados mantidos.

Na tabela 9 temos a manutenção de resultados por repositório após as cinco etapas de análise:

REPOSITÓRIO	RESULTADOS		
	ETAPA 1	ETAPA 5	%
PERIÓDICOS CAPES	23	4	17,39
ACM DL	33	4	12,12
IEEE XPLORE DL	20	4	20,00
TOTALIZAÇÃO	**76**	**12**	**15,79**

Tabela 9 – Manutenção de resultados por repositório

Verificando a manutenção de resultados por repositório, todos apresentaram a mesma quantidade de obras. O mais eficiente foi o “IEEE XPLORE DL” com 20,00% de resultados mantidos. Na tabela 10 temos a manutenção de resultados por palavra-chave após as cinco etapas de análise:

PALAVRA-CHAVE	RESULTADOS		
	ETAPA 1	ETAPA 5	%
METODOLOGIA ÁGIL	24	3	12,50
DESENVOLVIMENTO DE *SOFTWARE*	41	7	17,07
ENGENHARIA DE *SOFTWARE*	6	0	0,00
METODOLOGIA DE DESENVOLVIMENTO DE *SOFTWARE*	5	2	40,00
TOTALIZAÇÃO	**76**	**12**	**15,79**

Tabela 10 - Manutenção de resultados por palavra-chave

Verificando a manutenção de resultados por palavra-chave, a maior quantidade mantida foi com a palavra-chave “desenvolvimento de *software*” com 7 resultados na soma de todos os repositórios. A mais eficiente foi “metodologia de desenvolvimento de *software*” com 40,00% de resultados mantidos.

Essas obras foram a base para compor o referencial bibliográfico da pesquisa. A elas foram associadas as obras identificadas na etapa quatro com grande número de citações, obras sugeridas durante as leituras e que também contribuem com a pesquisa, artigos originais e *Books of Knowledge* (BOK).

3. REVISÃO DA LITERATURA

3.1. MODELO DE PROCESSOS GENÉRICO

Processo é um conjunto de atividades, ações e tarefas realizadas na criação de algum produto, porém no contexto da engenharia de software um processo não é um roteiro exato para desenvolver um software e sim uma abordagem flexível que permite a equipe de desenvolvimento escolher o conjunto ideal de atividades, ações e tarefas para a sua necessidade (PRESSMAN, 2011).

Considerando a possibilidade de adequação das atividades é possível uma equipe de desenvolvimento de *software* adaptar um método para necessidades específicas como, por exemplo, o cumprimento de normas internas de segurança da informação, a criação de artefatos específicos, etapas de validação intermediárias.

Todos os processos de *software* devem compreender quatro atividades que são fundamentais para a engenharia de software: Especificação das funcionalidades e restrições do *software*, *design* de *software*, implementação, validação para garantir que o *software* atinge o objetivo esperado pelo cliente e evolução do *software* para atender às necessidades dos clientes (SOMERVILLE, 2011).

Independente da proposta de um método ou de seus princípios, essas quatro atividades básicas sempre estarão presentes pois formam a base para um método de desenvolvimento de *software*.

3.2. MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

Os métodos de desenvolvimento de *software* surgiram no início dos anos 70 quando o Winston W. Royce propôs o método em cascata (RUPARELIA, 2010). Os métodos foram criados para apoiar as equipes de desenvolvimento e oferecer ferramentas que auxiliem na organização da atividade de desenvolvimento de *software* com o objetivo de entregar um produto adequado as expectativas do cliente respeitando escopo, prazo e custo que foram fatores indicados em pesquisa realizada com gestores de TI (PAIVA, et al., 2011) como sendo os principais fatores de sucesso em projetos de desenvolvimento de *software*.

3.2.1. Cascata

O método em cascata é uma proposta sequencial no qual todas as tarefas devem ser planejadas e agendadas antes do início dos trabalhos (PRESSMAN, 2011). O resultado de cada fase contempla documentos aprovados e a fase seguinte não deve ser iniciada até que a fase anterior esteja concluída. As etapas se sobrepõem e se retroalimentam. (SOMERVILLE, 2011).

Um dos princípios do método é a riqueza em documentações (ROYCE, 1970). Para entregar artefatos a cada nova iteração haverá a necessidade de empenhar maior esforço da equipe de desenvolvimento para produzir os documentos, o que pode aumentar os custos e o prazo do projeto. As etapas do modelo em cascata estão diretamente ligadas as atividades fundamentais do processo de *software* citado na seção 3.1, modelo de processos genérico: Requisitos de sistema, requisitos de *software*, análise, *design* do programa, codificação, testes e operação (ROYCE, 1970). A figura 1 mostra as iterações entre as etapas do processo de desenvolvimento:

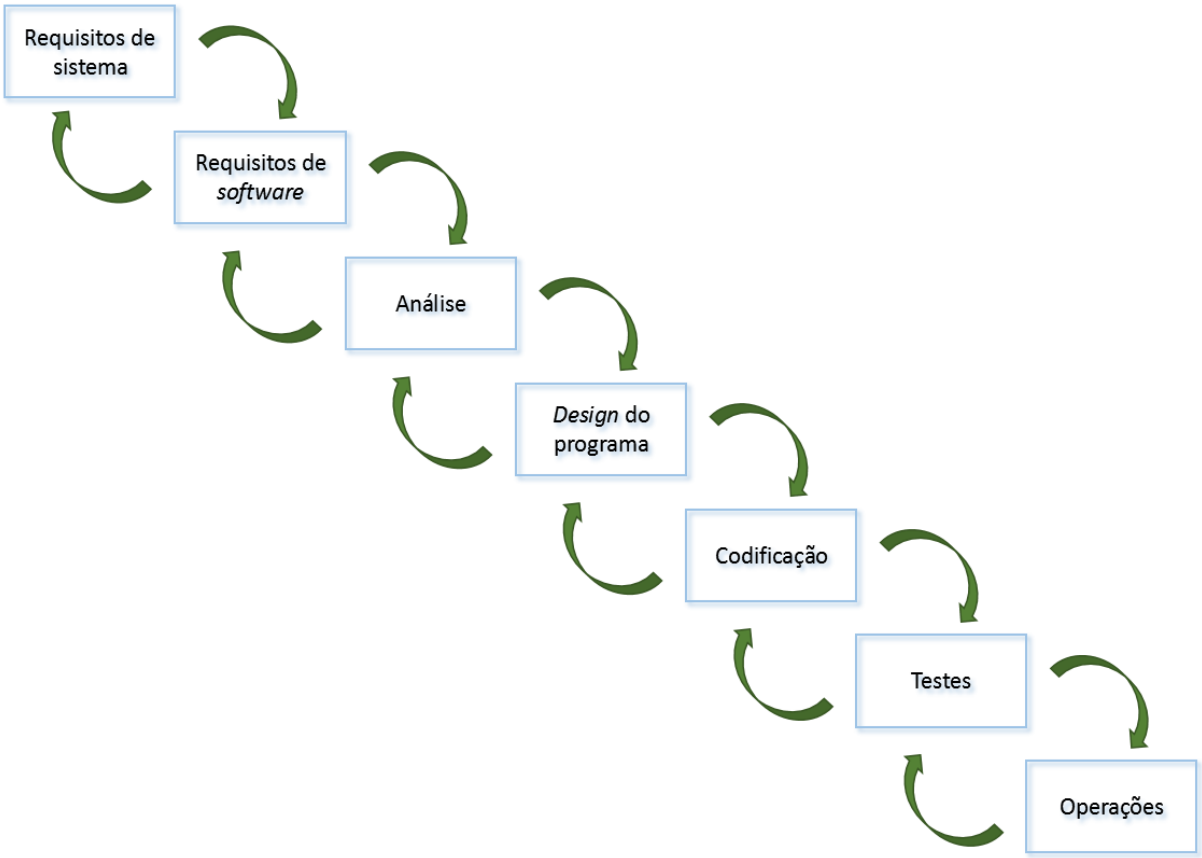

Figura 1 – Iterações entre as etapas do método de desenvolvimento de *software* em cascata

Fonte: (ROYCE, 1970)

Um projeto somente terá total êxito em relação a custo e prazo se a execução das etapas acontecer como é mostrado na figura 1, “Iterações entre as etapas do método de desenvolvimento de *software* em cascata”, onde cada etapa retroalimenta sua sucessora, o que viabilizaria também um planejamento para que a equipe pudesse desenvolver vários projetos em paralelo como em um processo fabril, em que as atividades vêm por uma esteira são realizadas pelos membros da equipe e devolvidas à esteira para que o próximo membro dê sequência ao trabalho.

A figura 2, “Retrocesso nas iterações entre as etapas do método de desenvolvimento de *software* em cascata”, mostra um retrocesso na sequência das etapas, o que representa grande impacto ao projeto no método em cascata. Uma mudança de escopo descoberta na etapa de testes, por exemplo, pode comprometer toda a estrutura do *software* e gerar enormes retrabalhos. O retrocesso de etapas pode gerar a paralisação de outras etapas do processo caso

seja necessário aguardar a atualização da documentação de etapas predecessoras para serem retomadas.

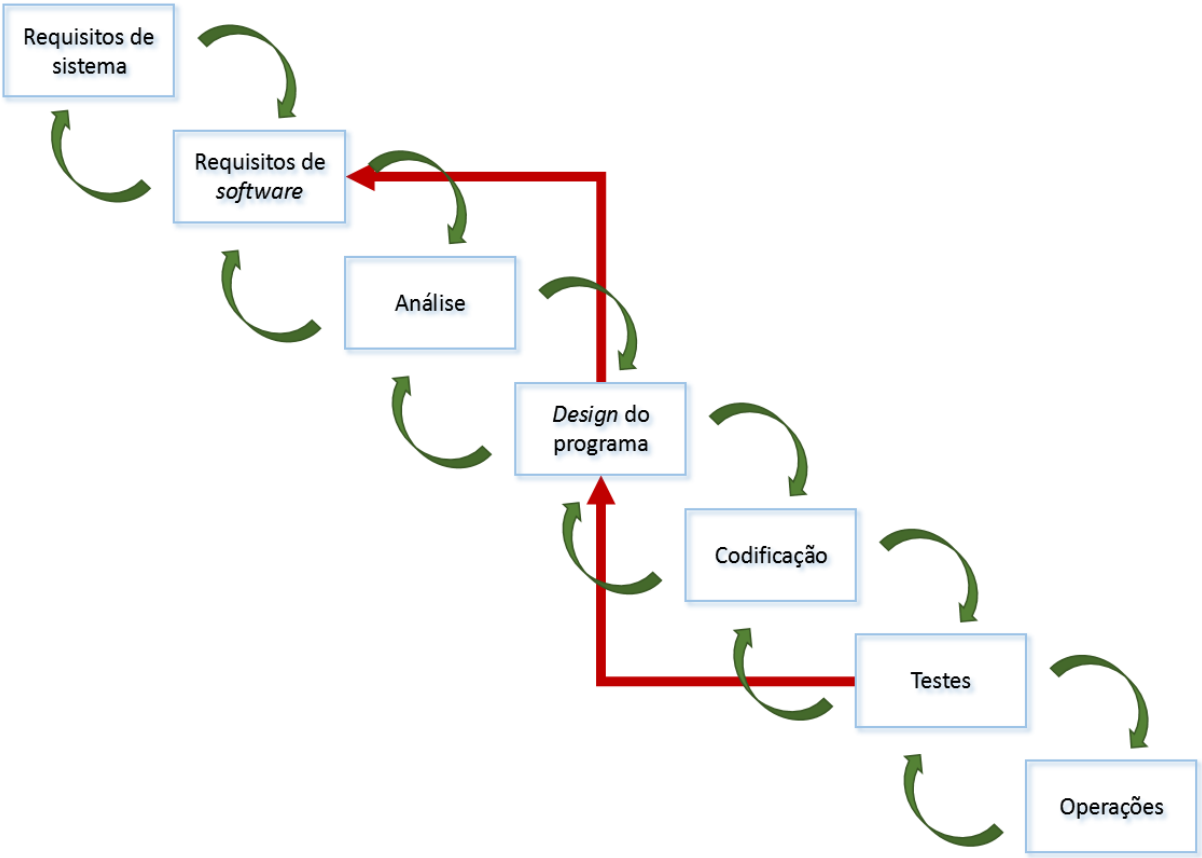

Figura 2 – Retrocesso nas iterações entre as etapas do método de desenvolvimento de *software* em cascata

Fonte: (ROYCE, 1970)

Por ser sequencial e exigir a conclusão de uma etapa antes do início de sua sucessora as mudanças de escopo sempre vão gerar impactos em custo e prazo já que dessa forma será necessário retroceder fases para atualização dos artefatos. O cliente verá a primeira versão do *software* apenas na fase de testes, e caso o levantamento de requisitos não tenha sido feito de forma abrangente o suficiente, as falhas funcionais serão percebidas neste momento.

3.2.2. Incremental

O método incremental propõe uma repetição de um processo de desenvolvimento gerando uma entrega operacional a cada execução denominada “incremento”. A figura 3, “Sequências lineares do método incremental”, mostra a repetição de um processo. A primeira execução é denominada *“Incremento n° 1”* onde normalmente é disponibilizado um *“produto essencial”* (PRESSMAN, 2011) que será utilizado e avaliado pelo cliente. A execução seguinte denominada *“Incremento n° 2”* será planejada com base na avaliação do cliente, considerando novas funcionalidades e possíveis adequações identificadas para o incremento anterior. Esse processo é repetido a cada incremento (OLIVEIRA, et al., 2015) e até que o *software* atenda a necessidade do cliente.

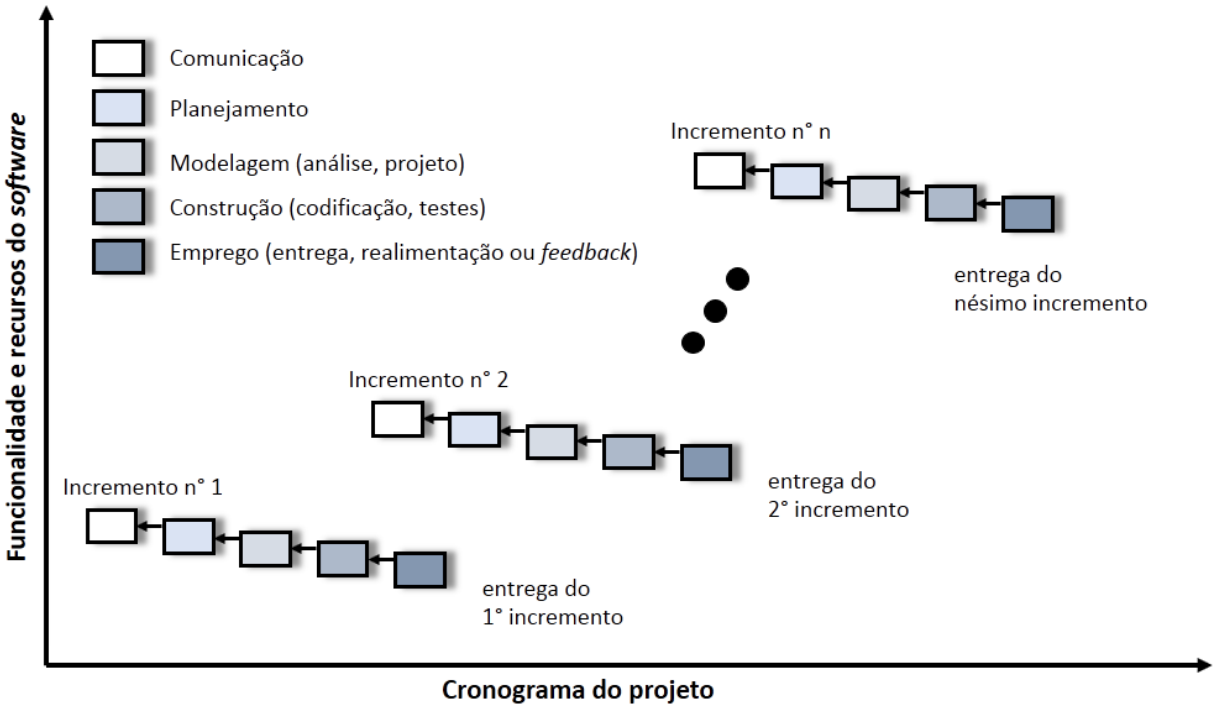

Figura 3 – Sequências lineares do método incremental

Fonte: (PRESSMAN, 2011)

O método incremental é adequado para modelos de negócio em que o escopo normalmente não é conhecido desde o início do projeto como *e-commerce* e sistemas pessoais

(SOMERVILLE, 2011) já que permite aprimorar o *software* a partir de *feedbacks* a cada incremento de *software* disponibilizado.

O desenvolvimento fracionado do *software* viabiliza alterações sem grandes impactos de custo e prazo por retrabalho, apresenta rapidamente uma versão do *software* e dá ao cliente a flexibilidade para optar por funcionalidades essenciais nos primeiros incrementos e por funcionalidades acessórias em incrementos posteriores além de permitir iniciar projetos mesmo sem que haja toda a disponibilidade de recursos tecnológicos (PRESSMAN, 2011) *(hardware e software)* para a implementação completa do *software*.

3.2.3. Rational Unified Process (RUP)

Criado pela Rational Software Corporation, o RUP disponibiliza à equipe de *software* uma base de conhecimento com diretrizes, modelos e mentores de ferramentas, viabilizando linguagem, processo e visão comuns sobre como desenvolver *softwares* a todos (RATIONAL, 1988).

É um processo derivado do trabalho sobre UML e do método *Unified Process (UP)* de propriedade da IBM disponível como um produto para venda (OLIVEIRA, et al., 2015). Enfatiza o desenvolvimento e a manutenção de modelos (representações semanticamente ricas do *software*) ao invés de um grande volume de documentos (RATIONAL, 1988) e suporta entrega incremental (SOMERVILLE, 2011). Por ser configurável, fornece suporte para vários tipos e tamanhos de projetos em uma organização (SEMEDO, 2012) e pode ser utilizado por equipes pequenas ou por grandes organizações de desenvolvimento de *software* (RATIONAL, 1988).

Assim como o método incremental, o RUP apresenta-se como uma alternativa para o desenvolvimento de *software* para clientes que não tem uma noção completa do escopo antes do início do desenvolvimento, porém apresenta a desvantagem de ser um *framework* proprietário aumentando os custos do projeto. A utilização da linguagem UML torna-se um pré-requisito já que a documentação será baseada em diagramas.

A figura 4, “Fases do método RUP”, ilustra a divisão em quatro fases distintas do método RUP estreitamente relacionadas ao negócio: iniciação, onde é estabelecido um caso

de negócios para o sistema; elaboração, onde são feitos o entendimento do domínio do problema, estabelecida uma arquitetura, realizado o planejamento e identificados os riscos; construção do projeto de sistemas, programação e testes; transição, onde é feita a implementação do *software* e sua respectiva documentação para o cliente (SOMERVILLE, 2011).

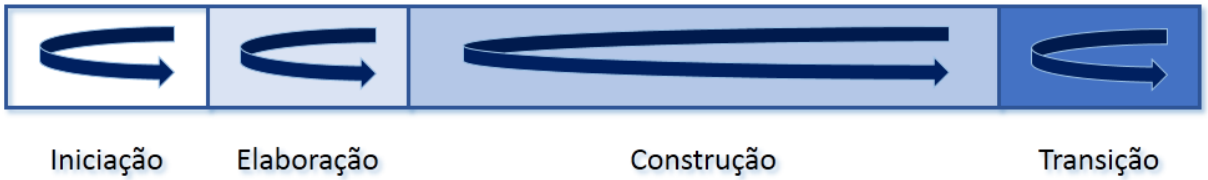

Figura 4 – Fases do método RUP

Fonte: (SOMERVILLE, 2011)

3.2.4. Adaptative Software Development (ASD)

Proposto por Jim Highsmith em 1992, o ASD é um método adaptativo para a construção de sistemas complexos com a colaboração humana e auto-organização das equipes como princípios básicos (PRESSMAN, 2011), com foco em pessoas, resultados aos métodos mínimos e a colaboração máxima (SEMEDO, 2012).

O princípio de colaboração em métodos ágeis tem o objetivo de reduzir a burocracia e aproximar as pessoas. A documentação foi um recurso criado com o objetivo de organizar a atividade de desenvolvimento de *software*, mas que em escopos complexos para a equipe e para o cliente acaba se tornando uma dificuldade adicional. Nesse sentido, um dos princípios do manifesto ágil é que o método mais eficiente e eficaz de transmitir informações para uma equipe de desenvolvimento e fazer fluir essa informação dentro da equipe é através da conversa face a face (MANIFESTO, 2001).

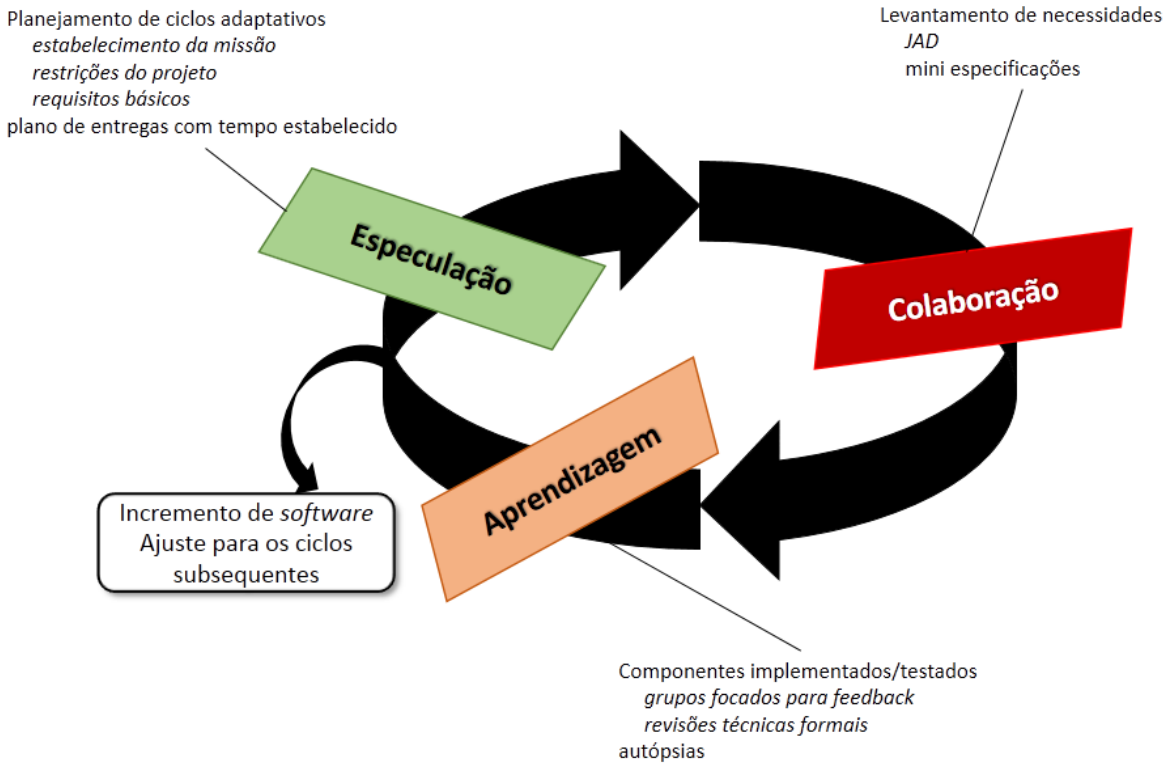

Figura 5 - O processo do método ASD

Fonte: (PRESSMAN, 2011)

A figura 5, “O processo do método ASD”, ilustra o processo do método ASD que possui três fases: especulação, onde é feito o “planejamento de ciclos adaptáveis” com informações sobre requisitos básicos; colaboração, onde é feito o levantamento de requisitos de negócios e especificações; aprendizagem, onde é feito o desenvolvimento, testes e implementação (SEMEDO, 2012).

Outra característica importante do método ASD é a auto-organização de equipes, princípio ágil de que as melhores arquiteturas, requisitos e *designs* emergem de equipes auto organizáveis (MANIFESTO, 2001).

3.2.5. Extreme Programming (XP)

Cinco valores estabelecem a base para o trabalho realizado na metodologia XP: comunicação, através da colaboração entre clientes e desenvolvedores sem utilizar documentação como meio de comunicação; simplicidade para projetar necessidades imediatas em um código de fácil implementação, deixando necessidades futuras para projetos futuros; *feedback* do cliente para e equipe em testes unitários ou a partir da implementação de um incremento de *software* ou da equipe para o cliente a partir de novas necessidades apresentadas. Reuniões diárias e rápidas *(stand up meeting)* são realizadas para discutir problemas e soluções além de reposicionar o foco da equipe (GOMES, 2009); disciplina para manter as práticas da XP sem ceder a pressões externas; respeito entre os membros, entre clientes e membros da equipe e indiretamente para o *software* (PRESSMAN, 2011; SILVA, 2013). Outros princípios podem ser adicionados se o contexto do projeto necessitar (WINGO, et al., 2015).

O método XP utiliza princípios do manifesto ágil que enfatizam a relação entre os interessados (clientes e usuários) e a equipe do projeto como a comunicação, o *feedback* constante e mútuo e o respeito. O manifesto ágil prega que interessados e equipe devem trabalhar diariamente em conjunto durante o projeto e que o fluxo de informações via conversa face a face é a forma mais eficiente de comunicação (MANIFESTO, 2001).

Conforme demonstrado na figura 6, “As quatro atividades metodológicas do método XP”, o método XP envolve um conjunto de regras e práticas de quatro atividades metodológicas: planejamento, que envolve o levantamento de requisitos para que os membros da equipe tenham percepção sobre os resultados solicitados, os fatores principais e funcionalidades através da coleta das “histórias de usuários”; projeto com ênfase para a implementação simples, desencorajando funcionalidades supostamente úteis para o futuro; codificação realizada em duplas, que é um conceito-chave da XP, fornece mecanismos para resolver problemas e garantir qualidade em tempo real e manter os desenvolvedores focados numa questão; testes de aceitação especificados pelo cliente com foco nas características e funcionalidade do sistema como um todo são obtidos das histórias de usuários implementadas (PRESSMAN, 2011; SOMERVILLE, 2011).

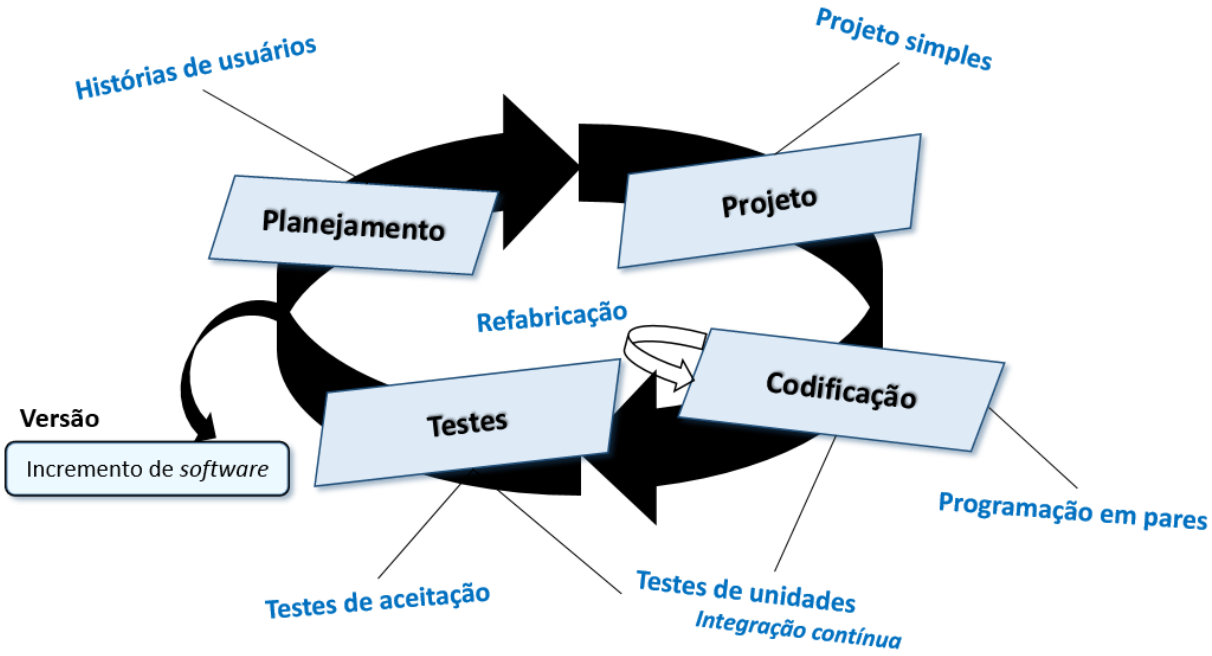

Figura 6 – As quatro atividades metodológicas do método XP

Fonte: (SOMERVILLE, 2011)

Outro princípio abordado indispensável para um método ágil é a simplicidade nos requisitos e soluções. O manifesto ágil prega que a simplicidade é essencial e maximiza a quantidade de trabalho não realizado (MANIFESTO, 2001). Este princípio pode ser observado em duas atividades: no planejamento, com o levantamento de requisitos de forma objetiva e restringindo o tempo de desenvolvimento e com soluções; no projeto, com ênfase em soluções simples e com as funcionalidades essenciais, deixando necessidades futuras para projetos futuros mesmo que isso represente novos custos (SOARES, 2004). Nos dois casos os princípios ágeis de simplicidade, entrega de *software* em pouco tempo e contínua (MANIFESTO, 2001) podem ser observados.

Pesquisas realizadas na Europa em diversos setores da economia revelaram 52% de empresas utilizando o método XP, sendo este então um dos métodos mais utilizados (TULI, et al., 2014).

3.3. CICLO DE VIDA EM PROJETOS

Os ciclos de vida em projetos são as fases do projeto, do início ao término, e as entregas e atividades nesse período de tempo podem variar muito de acordo com o tamanho e a complexidade do projeto. Podem ser abordagens previsíveis ao longo de uma sequência contínua ou adaptativas acionadas por mudanças (PMI, 2013).

Assim como foi dito na seção 3.1, modelo de processos genérico, existe a possibilidade de adaptar um método para situações específicas já que o desenvolvimento de um projeto não é como uma fabricação em linha que repetirá sempre a mesma sequência sob as mesmas condições. Apesar da possibilidade de adequação, três tipos de ciclo de vida em projetos são observados:

3.3.1. Predeterminados

Tem o escopo, prazo e custos para atingir o objetivo determinado o mais cedo possível. No início do projeto a equipe define o escopo geral e o plano de entrega, e dá sequência às fases de execução progredindo por fases sequenciais ou sobrepostas com atividades distintas. Mudanças são controladas e exigem replanejamento e aceite formal de novo escopo. São recomendados quando o produto é bem definido, quando há domínio da prática na indústria ou quando é desejável receber o produto por inteiro (PMI, 2013).

Por ser predeterminado este ciclo de vida exige uma noção sólida do escopo desde o início para que se possa planejar e orçar o projeto de forma segura e definitiva e a segurança de que não haverá mudanças durante o projeto que gere retrocesso de atividades e consequentemente atraso em cronograma e novos custos por retrabalho. A progressão por fases sequenciais demonstra a necessidade de não iniciar uma atividade antes de concluir sua predecessora imediata e a cada etapa concluída o custo de alteração aumentará já que o volume de retrabalho será cada vez maior. A rigidez em relação a mudanças torna a aplicação deste ciclo de vida eficaz para projetos com requisitos estáveis e com restrição para alterações como, por exemplo, o desenvolvimento de adequações legais e obrigatórias.

3.3.2. Iterativos e incrementais

São aqueles em que as fases repetem atividades conforme a compreensão do produto pela equipe aumenta. As iterações desenvolvem o produto por repetição de ciclos à medida que os incrementos agregam funcionalidades ao produto, sendo os incrementos futuros aprimoramentos dos antecessores ou novas. Normalmente é definida uma visão de alto nível, e o escopo detalhado é elaborado a cada iteração. O planejamento também é feito à medida que o projeto avança. Em alguns momentos a equipe de desenvolvimento pode mudar entre uma iteração e outra, dependendo do esforço exigido. Mudanças são controladas (PMI, 2013).

Neste caso, ao invés de se executar todo o projeto de uma única, executa-se uma sequência repetitiva com partes do escopo. Essa prática dá ao cliente e à equipe de projetos a oportunidade de aumentar o entendimento do produto de *software* a ser desenvolvido a cada novo incremento, porém os prazos e custos definitivos não serão conhecidos no início do projeto e sim durante sua execução.

São recomendados quando é necessário administrar mudanças de objetivo e escopo, diminuir a complexidade do projeto ou quando uma entrega parcial já é capaz de gerar valor ao cliente e não tem impacto na entrega final. Muitos projetos grandes e complexos são realizados de maneira iterativa, reduzindo o risco com a incorporação do *feedback* e lições aprendidas entre as iterações (PMI, 2013).

Apesar da possibilidade de realizar entregas incrementais é importante que cada incremento de fato gere valor ao cliente e não seja apenas o acréscimo de componentes que não representem novas funcionalidades para o cliente pois o *feedback* será construído a partir de novas experiências.

3.3.3. Adaptativos

Projetados para cenários de grande mutação e envolvimento contínuo dos interessados. Também são iterativos e incrementais, porém com iterações com duração de 2 a 4 semanas, tempo e recursos fixos. O escopo pode ser desmembrado em um conjunto de trabalhos. Ao final de cada iteração o produto será submetido para análise do cliente, que deve estar continuamente envolvido no projeto para fornecer *feedback*. Recomendado quando há rápida

mutação, os requisitos são difíceis de prever e há possibilidade de entregas incrementais que gerem valor aos interessados (PMI, 2013).

As principais diferenças em relação ao ciclo de vida iterativo e incremental são as iterações curtas e o trabalho a quatro mãos, ou seja, o envolvimento do cliente como integrante da equipe de desenvolvimento de software. É importante observar a disponibilidade integral do cliente como pré-requisito para projetos desse tipo.

3.4. RELAÇÃO ENTRE OS MÉTODOS E OS TIPOS DE CICLO DE VIDA EM PROJETOS

Comparando as características dos métodos de desenvolvimento de *software* com as características obtidas no levantamento bibliográfico e detalhadas na seção 3.2, métodos de desenvolvimento de *software*, é possível posicionar os métodos de desenvolvimento de *software* nos três ciclos de vida de projetos apresentados na seção 3.3, ciclo de vida em projetos.

O método em cascata tem proposta sequencial, necessidade de escopo, custo e prazos definidos no início do projeto e atividades retroalimentadas, caracteriza-se como um método com ciclo de vida predeterminado.

No método RUP, o *software* pode ser desenvolvido em uma única iteração caso exista maturidade do escopo, mas como existe a possibilidade de entregas incrementais, compreensão evolucionária do escopo e facilidade para absorver alterações o RUP caracteriza-se como um método com ciclo de vida iterativo e incremental apesar da possibilidade de ser utilizado como predeterminado. Com proposta de repetição de sequência de atividades, compreensão evolucionária do escopo e facilidade para absorver alterações o método incremental caracteriza-se como um método com ciclo de vida iterativo e incremental.

A proposta de repetição de sequência de atividades, compreensão evolucionária do escopo, facilidade para absorver alterações e colaboração mútua entre equipe de desenvolvimento e cliente faz do método ASD um método com ciclo de vida adaptativo. Já o método XP possui todas as características do método ASD além de iterações curtas, porém também se caracteriza como um método com ciclo de vida adaptativo.

Figura 7 – Métodos de desenvolvimento de *software* versus ciclo de vida em projetos

A figura 7, “Métodos de desenvolvimento de *software* versus ciclo de vida em projetos”, mostra a distribuição dos métodos de desenvolvimento de *software* nos ciclos de vida em projetos a partir de uma análise comparativa de suas respectivas características.

4. SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

Os dados apresentados no capítulo 3, revisão da literatura, mostram que os métodos de desenvolvimento de *software* são elaborados a partir de um modelo de processo genérico da engenharia baseado em quatro atividades fundamentais para a engenharia de *software* que são especificação, *design*, implementação e validação (SOMERVILLE, 2011), possuem princípios, atividades específicas e uma proposta de ciclo de vida que pode ser predeterminado, iterativos e incrementais ou adaptativos (PMI, 2013).

Na seção 3.4, relação entre os métodos e os tipos de ciclo de vida em projeto, o enquadramento dos cinco métodos analisados mostra que, apesar de existirem diferenças de atividades e princípios entre os métodos, suas propostas de ciclo de vida os torna similares do ponto de vista de projetos. Sendo assim, a análise para identificação dos critérios direcionadores pode ser feita considerando o ciclo de vida em que o método está enquadrado, os seus princípios e as peculiaridades de suas atividades. Nas próximas seções esses pontos serão explorados de modo que se identifique critérios direcionadores na seleção de um método de desenvolvimento de *software* a partir das características do projeto.

4.1. IDENTIFICAÇÃO DOS CRITÉRIOS DIRECIONADORES PARA SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

4.1.1. Análise do ciclo de vida predeterminado e do método em cascata

Uma característica importante do ciclo de vida predeterminado é a necessidade de escopo, prazo e custos definidos no início do projeto, o que vai exigir **requisitos claros e bem definidos**. Para isso será necessário que o cliente tenha de forma antecipada uma ideia pronta e madura do produto de *software* que deseja receber. Com a definição antecipada de um escopo maduro o ciclo de vida predeterminado possibilita a **definição segura de planejamento e custos de forma antecipada**, já que se sabe o que será feito do início ao fim do projeto. Essa característica justifica uma das recomendações de aplicação do ciclo de vida predeterminado que é para situações em que o produto está bem definido como, por exemplo,

adequações legais e obrigatórias que envolvem regras predefinidas e que não podem ser alteradas.

A progressão por fases sequenciais expõe o risco de retrabalho em caso de mudança de escopo e justifica a **restrição de aceitação a mudanças** do ciclo de vida. Se o cliente opta por uma mudança em um requisito, as fases anteriores a fase em que o projeto sem encontra precisam ser revisadas como demonstra a figura 2, “Retrocesso nas iterações entre as etapas do método de desenvolvimento de *software* em cascata”, na seção 3.2.1, Cascata. O impacto em custos e em prazo pode variar de acordo com a fase em que o projeto está, independentemente do tamanho da alteração solicitada, o que é indesejado caso o cliente tenha **restrições de prazo e custo no projeto**.

Como o ciclo não prevê a disponibilização de incrementos do *software* durante a execução, a **entrega do produto será realizada em uma única versão ao término do projeto**, justificando a recomendação de aplicação quando é desejável receber o produto por inteiro. Será necessário também **dispor de forma antecipada de arquitetura de *software* e *hardware*** para uma implementação completa dos componentes como, por exemplo, em um projeto em que será necessário obter algum recurso específico para a equipe de desenvolvimento, um acréscimo de espaço em disco rígido no servidor ou uma atualização de versão de um banco de dados ou sistema operacional, esses recursos precisarão ser disponibilizados desde o início do projeto.

Por não existir um processo evolucionário durante o projeto a equipe não terá a oportunidade de desenvolver aprendizado sobre o produto de *software* e sobre as tecnologias empregadas, justificando a recomendação de aplicação quando há domínio da prática na indústria e exigindo **experiência da equipe de desenvolvimento para trabalhar com as tecnologias que serão utilizadas no projeto**. Pelo mesmo motivo o **conhecimento prévio do domínio funcional do *software* pela equipe** pode ter influência também. Se o domínio funcional for algo totalmente novo para a equipe, ela pode ter mais dificuldade para assimilar os requisitos do que se for em um tema de censo comum ou de conhecimento prévio. Por exemplo, um *software* contábil pode ser de difícil compreensão para uma equipe que nunca desenvolveu algo parecido, mas não para uma equipe que já tenha realizado um projeto similar anteriormente ou para uma equipe especializada em desenvolvimento de *software* contábil. Já um *software* para um comércio eletrônico pode ser mais fácil de ser assimilado por ser um domínio funcional popularmente conhecido.

O método em cascata contempla a **entrega de um documento e sua respectiva aprovação** ao final de cada uma das etapas. Essa atividade exigirá **maior esforço da equipe** e por consequência aumentará o **custo do projeto**. Como as fases são iniciadas somente após a conclusão e aprovação de uma fase predecessora, uma mudança de escopo representa a paralisação da etapa atual do projeto, o retrocesso até a etapa que precisa ser revisada e a revisão de todas as etapas sucessoras impactadas. Essa é mais uma característica que justifica a restrição a mudanças e a recomendação de uso apenas quando o produto está bem definido.

Será necessário o envolvimento do cliente no levantamento de requisitos, na aprovação de documentos e testes. O cliente não participa da execução do projeto junto com a equipe de desenvolvimento. Dessa forma, é possível realizar o projeto mesmo quando a **disponibilidade do cliente é baixa**.

4.1.2. Análise do ciclo de vida iterativo e incremental e dos métodos incremental e RUP

No ciclo de vida iterativo e incremental é possível trabalhar com **requisitos vagos e indefinidos** pois o *software* é desenvolvido em partes, o que viabiliza ao cliente e a equipe de desenvolvimento aumentar a compreensão sobre o produto a partir do *feedback* após cada entrega como demonstra a figura 3, “Sequências lineares do método incremental”, na seção 3.2.2, Incremental. O cliente pode apresentar uma ideia em alto nível do *software* que necessita e a equipe vai planejar o projeto, porém a cada *feedback* os requisitos e o planejamento são revistos, o que **dificulta ter um cronograma e orçamento precisos de forma antecipada** já que é esperado que haja mudanças durante a execução do projeto. Para tanto, é necessário que o cliente esteja disposto a **aceitar mudanças de prazo e custo**. Essa característica justifica a recomendação de aplicação quando há **necessidade de administrar mudanças** e de **diminuir a complexidade do projeto**.

Como o *software* é disponibilizado ao longo da execução do projeto **o cliente já recebe uma primeira versão antes da conclusão do projeto**, e por isso cada incremento deve agregar valor ao *software*. A equipe de projetos terá a oportunidade de **trabalhar com tecnologias que ainda não tem total domínio** e com **temas dos quais ainda não possui especialização do conhecimento funcional,** pois poderá adquirir experiência a cada iteração. Pelo mesmo motivo é **possível lidar com possíveis ausências de arquitetura de *software* e**

hardware para a implementação completa, pois os componentes que dependem desses recursos poderão ser planejados para iterações futuras. Essas possibilidades justificam a recomendação de aplicação quando há necessidade de diminuir a complexidade do projeto.

O método incremental não impõe a produção de uma documentação volumosa, mas não apresenta oposição a documentação. Sendo assim é **possível definir a quantidade de documentos que serão produzidos**. Já o método RUP tem como princípio a utilização da linguagem UML como forma de documentação e **desmotiva a produção volumosa de documentos**. Uma desvantagem na utilização do método RUP em relação ao incremental é o fato de ser um *framework* proprietário, gerando custos adicionais ao projeto.

O cliente tem uma participação grande em projetos de ciclo de vida iterativo e incremental. Participam da fase de levantamento de requisitos, aprovação dos testes e *feedback* após a entrega de incrementos. Como se trata de um processo cíclico, o cliente é envolvido várias vezes até que a versão final do *software* seja alcançada. Sendo assim, **é necessário que o cliente tenha alta disponibilidade** para atender a equipe de desenvolvimento.

4.1.3. Análise do ciclo de vida adaptativo e dos métodos ASD e XP

O ciclo de vida adaptativo também é iterativo e incremental, herdando suas características. As principais diferenças estão nos princípios de **iterações de curto prazo** que vão entregar *software* em uma dinâmica mais rápida, **pré-disposição para aceitar mudanças** e reagir a elas de forma rápida e no **envolvimento do cliente como integrante da equipe** que exige disponibilidade total do mesmo.

Nos métodos ASD e XP um princípio importante do manifesto ágil a **redução da produção de documentos e o incentivo as relações humanas**. As informações devem fluir preferencialmente através da conversa face a face. No método XP existe o conceito-chave da codificação em duplas que pode contribuir para um **custo mais alto do projeto**.

4.1.4. Relação de critérios direcionadores para a seleção de métodos de desenvolvimento de software

Os dados analisados nas seções 4.1.1, “Análise do ciclo de vida predeterminado e do método em cascata”, 4.1.2, “Análise do ciclo de vida iterativo e incremental e dos métodos incremental e RUP” e 4.1.3, “Análise do ciclo de vida adaptativo e dos métodos ASD e XP” mostram pontos em comum entre os ciclos de vida em projetos e os métodos de desenvolvimento de *software* analisados. O quadro 1, “pontos em comum entre os ciclos de vida em projetos e os métodos de desenvolvimento de *software*”, expõe uma compilação dos dados analisados com nove critérios para a seleção de métodos de desenvolvimento de *software*:

	Ciclos de vida em projetos e os métodos enquadrados		
	Predeterminado *Método em cascata*	**Iterativo e incremental** *Métodos incremental e RUP*	**Adaptativo** *Métodos ASD e XP*
1 - Requisitos	Claros e bem definidos	Vagos e indefinidos	Mudança frequente
2 - Planejamento e orçamento	Definidos de forma antecipada	Atualizados a cada *feedback*	Atualizados a cada *feedback*
3 - Mudanças	Restrição	Controladas	Aceita
4 - Entregas	No final do projeto	A cada iteração planejada	A cada iteração *(de 2 a 4 semanas)*
5 - Arquitetura de *software* e *hardware*	Antecipada	Não antecipada	Não antecipada
6 - Experiência da equipe	Recomendado dominar as tecnologias utilizadas	Possibilidade de adaptação durante o projeto	Possibilidade de adaptação durante o projeto
7 - Conhecimento funcional da equipe	Recomendado ter conhecimento prévio	Possibilidade de adaptação durante o projeto	Possibilidade de adaptação durante o projeto
8 - Documentação	Volumosa	Reduzida	Pouca ou informal
9 - Disponibilidade do cliente	Pouca	Alta	Integral

Quadro 1 - Pontos em comum entre os ciclos de vida em projetos e os métodos de desenvolvimento de *software*

4.2. ATRIBUIÇÃO DE CARACTERÍSTICAS POR CRITÉRIOS PARA SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

Os métodos de desenvolvimento de *software* serão recomendados a partir da indicação das características do projeto informadas. Nesse caso, é necessário que exista um formulário que contenha as características identificadas na análise dos dados e compiladas no quadro 1, "Pontos em comum entre os ciclos de vida em projetos e os métodos de desenvolvimento de *software*" da seção 4.1.4, "Relação de critérios direcionadores para a seleção de métodos de desenvolvimento de *software"*.

O quadro 2, "Formulário para indicação das características do projeto", apresenta um formulário contendo os critérios identificados e as opções de resposta detalhadas:

Critérios	Opções de resposta
Requisitos do usuário	Claros e bem definidos *(adequações legais com restrição a mudanças)*
	Vagos e indefinidos
	Previsão de mudança constante
Planejamento e orçamento	Definidos no início do projeto
	Definidos em alto nível no início do projeto e atualizados a cada *feedback*
Mudanças de requisitos e de escopo	Restrição *(não aceita ou controla sob aprovação do cliente)*
	Controlada *(aceita mudanças a partir de feedback após iteração)*
	Aceita *(reação rápida a mudanças)*
Entrega do *software*	Ao final do projeto
	Um incremento que agrega valor a cada iteração planejada
	Um incremento que agrega valor a cada iteração *(de 2 a 4 semanas)*
Arquitetura de *software* e *hardware*	Disponível no início do projeto
	Parcial no início e disponibilizada durante o projeto
Experiência da equipe com as tecnologias utilizadas	Domínio no início do projeto
	Sem domínio ou em adaptação no início do projeto
Conhecimento funcional da equipe com o tema envolvido	Conhecimento no início do projeto
	Sem conhecimento ou em adaptação no início do projeto
Documentação	Volumosa *(documentação a cada fase concluída)*
	Reduzida *(possibilidade de definição dos documentos produzidos)*
	Pouca ou informal *(preferência por relações humanas e fluidez de informação por conversas)*
Disponibilidade do cliente	Pouca *(envolvimento no levantamento de requisitos, aprovação de documentos e de testes)*
	Alta *(envolvimento no levantamento de requisitos, aprovação de testes e feedback a cada iteração planejada)*
	Integral *(o cliente faz parte da equipe de desenvolvimento)*

Quadro 2 - Formulário para indicação das características do projeto

Outros autores também destacam a importância de critérios direcionadores na seleção de métodos de desenvolvimento de *software* como a familiaridade da equipe com a tecnologia utilizada (Executive Brief, 2008), a clareza e a estabilidade dos requisitos do usuário

(AHMAR, 2005; Executive Brief, 2008), a disponibilidade do cliente (SHARMA, 2011) e a infraestrutura disponível para suportar o *software* (OZTURK, 2013).

4.3. ATRIBUIÇÃO DOS MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE POR CRITÉRIOS PARA SELEÇÃO DE MODELOS E CARACTERÍSTICAS

Os métodos de desenvolvimento de *software* herdam as características dos ciclos de vida em projeto, mas como existem diferenças entre os ciclos para alguns dos critérios é necessário atribuir os métodos às diferentes variações de características. Nesta seção cada um dos critérios será demonstrado com suas respectivas características e métodos atribuídos considerando os dados analisados na seção 4.1, "Identificação dos critérios direcionadores para seleção de métodos de desenvolvimento de *software*".

No critério 1, "Requisitos do usuário", existem três possibilidades. Na primeira delas o método em cascata é recomendado para requisitos claros e definidos, porém essa condição não é restritiva para os demais métodos, então todos os métodos são aplicáveis. Na segunda os métodos incremental, RUP são recomendados para requisitos vagos e indefinidos, porém essa condição não é restritiva para os métodos ASD e XP, então eles também são recomendados. Na terceira os métodos ASD e XP são recomendados para requisitos com previsão de mudança constante. A distribuição é demonstrada no quadro 3, "Atribuição de métodos às características do critério 1, 'Requisitos do usuário'":

	Requisitos do usuário		
	Claros e bem definidos *(Adequações legais com restrição a mudanças)*	Vagos e indefinidos	Previsão de mudança constante
Predeterminados	Cascata	-	-
Iterativos e Incrementais	Incremental RUP	Incremental RUP	-
Adaptativos	ASD XP	ASD XP	ASD XP

Quadro 3 – Atribuição de métodos às características do critério 1, "Requisitos do usuário"

No critério 2, “Planejamento e orçamento”, existem duas possibilidades. Na primeira delas o método em cascata é recomendado para projetos com planejamento e orçamento definidos no início. Na segunda os métodos incremental, RUP, ASD e XP são recomendados para projetos com planejamento e orçamento definidos em alto nível no início e atualizados a cada *feedback*. A distribuição é demonstrada no quadro 4, “Atribuição de métodos às características do critério 2, ‘Planejamento e orçamento’”:

	Planejamento e orçamento	
	Definidos no início do projeto	Definidos em alto nível no início do projeto e atualizados a cada *feedback*
Predeterminados	Cascata	-
Iterativos e Incrementais	-	Incremental RUP
Adaptativos	-	ASD XP

Quadro 4 - Atribuição de métodos às características do critério 2, “Planejamento e orçamento”

No critério 3, “Mudanças de requisitos e de escopo”, existem três possibilidades. Na primeira delas o método em cascata é recomendado para projetos com restrição a mudanças. Na segunda os métodos incremental e RUP são recomendados para projetos que aceitam mudanças de forma controlada. Na terceira os métodos ASD e XP são recomendados para projetos com mudança frequente. A distribuição é demonstrada no quadro 5, “Atribuição de métodos às características do critério 3, ‘Mudanças de requisitos e de escopo’”:

	Mudanças de requisitos e de escopo		
	Restrição *(Não aceita ou controla sob aprovação do cliente)*	Controlada *(Aceita mudanças a partir de feedback após iteração)*	Aceita *(Reação rápida a mudanças)*
Predeterminados	Cascata	-	-
Iterativos e Incrementais	-	Incremental RUP	-
Adaptativos	-	-	ASD XP

Quadro 5 - Atribuição de métodos às características do critério 3, “Mudanças de requisitos e de escopo”

No critério 4, “Entrega do software”, existem três possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que a expectativa do cliente é receber o *software* ao final do projeto. Na segunda os métodos incremental e RUP são recomendados para projetos em que a expectativa é receber um incremento que agrega valor ao produto a cada iteração planejada. Na terceira os métodos ASD e XP são recomendados para projetos em que a expectativa é receber um incremento que agrega valor ao produto a cada iteração mas em períodos curtos que variam de duas a quatro semanas. A distribuição é demonstrada no quadro 6, “Atribuição de métodos às características do critério 4, ‘Entrega do software’”:

	Entrega do *software*		
	Ao final do projeto	Um incremento que agrega valor a cada iteração planejada	Um incremento que agrega valor a cada iteração *(De 2 a 4 semanas)*
Predeterminados	Cascata	-	-
Iterativos e Incrementais	-	Incremental RUP	-
Adaptativos	-	-	ASD XP

Quadro 6 - Atribuição de métodos às características do critério 4, “Entrega do *software*”

No critério 5, “Arquitetura de software e hardware”, existem duas possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que há disponibilidade da arquitetura no início do projeto, porém essa condição não é restritiva para os demais métodos, então todos os métodos são aplicáveis. Na segunda os métodos incremental, RUP, ASD e XP são recomendados para projetos em que a arquitetura está disponível parcialmente no início do projeto e será disponibilizada durante a execução. A distribuição é demonstrada no quadro 7, “Atribuição de métodos às características do critério 5, ‘Arquitetura de software e hardware’”:

	Arquitetura de *software* e *hardware*	
	Disponível no início do projeto	Parcial no início e disponibilizada durante o projeto
Predeterminados	Cascata	-
Iterativos e Incrementais	Incremental RUP	Incremental RUP
Adaptativos	ASD XP	ASD XP

Quadro 7 - Atribuição de métodos às características do critério 5, "Arquitetura de *software* e *hardware*"

No critério 6, "Experiência da equipe com as tecnologias utilizadas", existem duas possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que a equipe de desenvolvimento domina as tecnologias utilizadas desde o início do projeto, porém essa condição não é restritiva para os demais métodos, então todos os métodos são aplicáveis. Na segunda os métodos incremental, RUP, ASD e XP são recomendados para projetos em que a equipe de desenvolvimento não domina ou está em adaptação no emprego das tecnologias utilizadas no início do projeto. A distribuição é demonstrada no quadro 8, "Atribuição de métodos às características do critério 6, 'Experiência da equipe com as tecnologias utilizadas'":

	Experiência da equipe com as tecnologias utilizadas	
	Domínio no início do projeto	Sem domínio ou em adaptação no início do projeto
Predeterminados	Cascata	-
Iterativos e Incrementais	Incremental RUP	Incremental RUP
Adaptativos	ASD XP	ASD XP

Quadro 8 - Atribuição de métodos às características do critério 6, "Experiência da equipe com as tecnologias utilizadas"

No critério 7, “Conhecimento funcional da equipe com o tema envolvido”, existem duas possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que a equipe de desenvolvimento tem conhecimento funcional desde o início do projeto, porém essa condição não é restritiva para os demais métodos, então todos os métodos são aplicáveis. Na segunda os métodos incremental, RUP, ASD e XP são recomendados para projetos em que a equipe de desenvolvimento não tem conhecimento ou está em adaptação no início do projeto. A distribuição é demonstrada no quadro 9, “Atribuição de métodos às características do critério 6, ‘Conhecimento funcional da equipe com o tema envolvido’”:

	Conhecimento funcional da equipe com o tema envolvido	
	Conhecimento no início do projeto	Sem conhecimento ou em adaptação no início do projeto
Predeterminados	Cascata	-
Iterativos e Incrementais	Incremental RUP	Incremental RUP
Adaptativos	ASD XP	ASD XP

Quadro 9 - Atribuição de métodos às características do critério 7, “Conhecimento funcional da equipe com o tema envolvido”

No critério 8, “Documentação”, existem três possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que será necessário produzir documentação volumosa com artefatos a cada fase concluída. Na segunda os métodos incremental e RUP são recomendados para projetos em que a documentação pode ser reduzida com possibilidade de definir os documentos necessários. Na terceira os métodos ASD e XP são recomendados para projetos em que a produção de documentação é desmotivada e as relações humanas são incentivadas com a fluidez de informações por conversas. A distribuição é demonstrada no quadro 10, “Atribuição de métodos às características do critério 8, ‘Documentação’”:

	Documentação		
	Volumosa *(Artefatos a cada fase concluída)*	Reduzida *(Possibilidade de definição dos documentos produzidos)*	Pouca *(Preferência por relações humanas e fluidez de informação por conversas)*
Predeterminados	Cascata	-	-
Iterativos e Incrementais	-	Incremental RUP	-
Adaptativos	-	-	ASD XP

Quadro 10 - Atribuição de métodos às características do critério 8, “Documentação”

No critério 9, “Disponibilidade do cliente”, existem três possibilidades. Na primeira delas o método em cascata é recomendado para projetos em que o cliente não tem muita disponibilidade e será envolvido apenas no levantamento de requisitos e na aprovação de documentos e testes. Na segunda os métodos incremental e RUP são recomendados para projetos em que o cliente tem maior disponibilidade e será envolvido no levantamento de requisitos, na aprovação de testes e nos *feedbacks* a cada iteração, porém essa condição não é restritiva para o método em cascata, então ele também é aplicável. Na terceira os métodos ASD e XP são recomendados para projetos em que o cliente tem disponibilidade integral para fazer parte da equipe de desenvolvimento, porém essa condição não é restritiva para os método em cascata, incremental e RUP, então eles também são aplicáveis, mas sem que o cliente faça parte da equipe. A distribuição é demonstrada no quadro 11, “Atribuição de métodos às características do critério 9, ‘Disponibilidade do cliente’”:

	Disponibilidade do cliente		
	Pouca *(Envolvimento no levantamento de requisitos, aprovação de documentos e de testes)*	Alta *(Envolvimento no levantamento de requisitos, aprovação de testes e feedback a cada iteração planejada)*	Integral *(O cliente pode fazer parte da equipe de desenvolvimento)*
Predeterminados	Cascata	Cascata	Cascata
Iterativos e Incrementais	-	Incremental RUP	Incremental RUP
Adaptativos	-	-	ASD XP

Quadro 11 - Atribuição de métodos às características do critério 9, “Disponibilidade do cliente”

Os métodos de desenvolvimento incremental e RUP estão relacionados ao ciclo de vida iterativo e incremental e podem ser indicados para aplicação sob as mesmas condições de projeto, no entanto será atribuído um peso maior para o método incremental porque não envolve custos adicionais. Já o método RUP envolve custos por ser um *framework* proprietário. O mesmo acontece entre os métodos ASD e XP que estão relacionados ao ciclo de vida adaptativo, mas o método XP pode ter um custo maior por ter como conceito-chave a codificação em duplas que pode gerar custos adicionais de alocação de profissionais.

4.4. APLICAÇÃO PARA A INDICAÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

Esta seção apresenta uma proposta para uma aplicação que faça a recomendação dos métodos de desenvolvimento de *software* a partir da indicação das características para cada um dos critérios apresentados na seção 4.2, “Atribuição de características por critérios para seleção de métodos de desenvolvimento de *software*”. A proposta é de uma aplicação na plataforma *web* utilizando banco de dados Microsoft Access e as linguagens ASP *(Active Server Pages)* e JavaScript.

O banco de dados armazena os dados de ciclos de vida em projetos, métodos de desenvolvimento de *software*, critérios, características e a relação entre todas as informações. A figura 8, “Modelo de entidade e relacionamento do banco de dados da aplicação”, mostra as tabelas e suas respectivas relações:

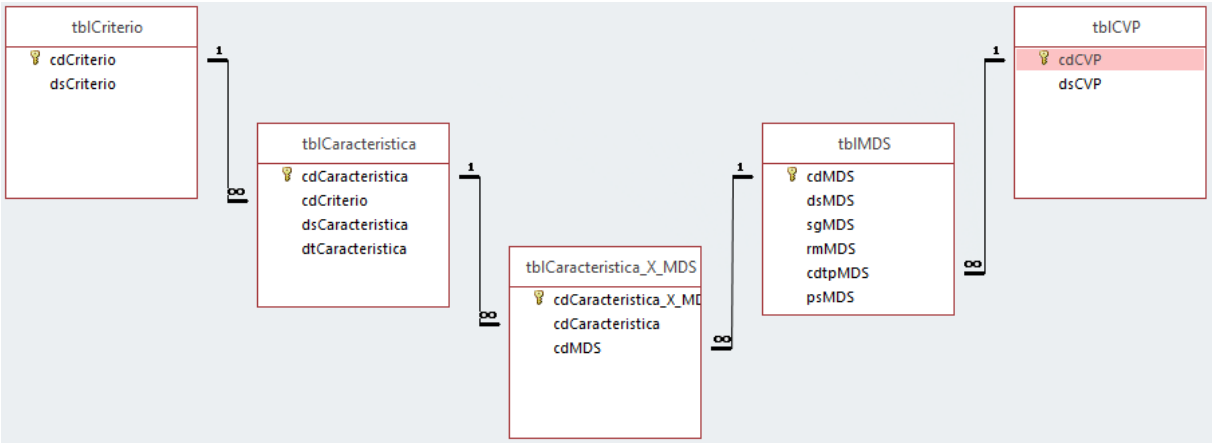

Figura 8 - Modelo de entidade e relacionamento do banco de dados da aplicação

A tabela “tblCVP” armazena os dados dos ciclos de vida em projetos que estão detalhados na seção 3.3, “Ciclo de vida em projetos”. São armazenados um código numérico e os nomes dos ciclos de vida em projetos são armazenados. A tabela “tblMDS” armazena os dados métodos de desenvolvimento de *software* que estão detalhados na seção 3.2, “Métodos de desenvolvimento de *software*”. São armazenados um código numérico, o nome, sigla, descrição resumida, ciclo de vida em projeto relacionado conforme detalhado na seção 3.3, “Ciclo de vida em projetos” e um peso conforme detalhado na seção 4.3, “Atribuição dos

métodos de desenvolvimento de *software* por critérios para seleção de modelos e características".

A tabela "tblCriterio" armazena os critérios direcionadores para seleção de métodos de desenvolvimento de *software* que estão detalhados na seção 4.1.4, "Relação de critérios direcionadores para a seleção de métodos de desenvolvimento de *software*. São armazenados um código numérico e as descrições dos critérios são armazenadas. A tabela "tblCaracteristica" armazena as características dos critérios direcionadores que estão detalhadas na seção 4.2, "Atribuição de características por critérios de seleção de métodos de desenvolvimento de *software*". São armazenados um código numérico, as descrições, o detalhe (caso exista) e o critério ao qual está relacionada a característica.

A tabela "tblCaracteristica_X_MDS" armazena a relação entre as características dos critérios direcionadores e os métodos de desenvolvimento de *software* que está detalhada na seção 4.3, "Atribuição dos métodos de desenvolvimento de *software* por critérios para seleção de modelos e características". São armazenados um código numérico, o código da característica e o código do método atribuído.

Ao acessar a aplicação o usuário visualiza um formulário com uma relação dos critérios e as características de cada critério para que sejam selecionadas. Pelo menos uma característica de cada critério deve ser selecionada para que seja possível analisar as combinações e recomendar um método.

Depois de preencher todo o formulário, o usuário clica no botão "Enviar" para que um método seja recomendado. A aplicação faz uma pesquisa no banco de dados verificando quais métodos estão associados ao conjunto de características indicadas. A aplicação retorna para o formulário, exibe na parte de cima da tela o resultado da pesquisa que pode ser a relação dos métodos ordenados por peso ou uma mensagem de que nenhum método está associado as características de projeto indicadas e recarrega as opções marcadas pelo usuário na última consulta. O usuário pode então alterar a pesquisa e executar a busca novamente.

A figura 9, "Formulário de critérios e características de projeto da aplicação", mostra o formulário que o usuário preenche para realizar a busca de métodos de desenvolvimento de *software* por características de projeto:

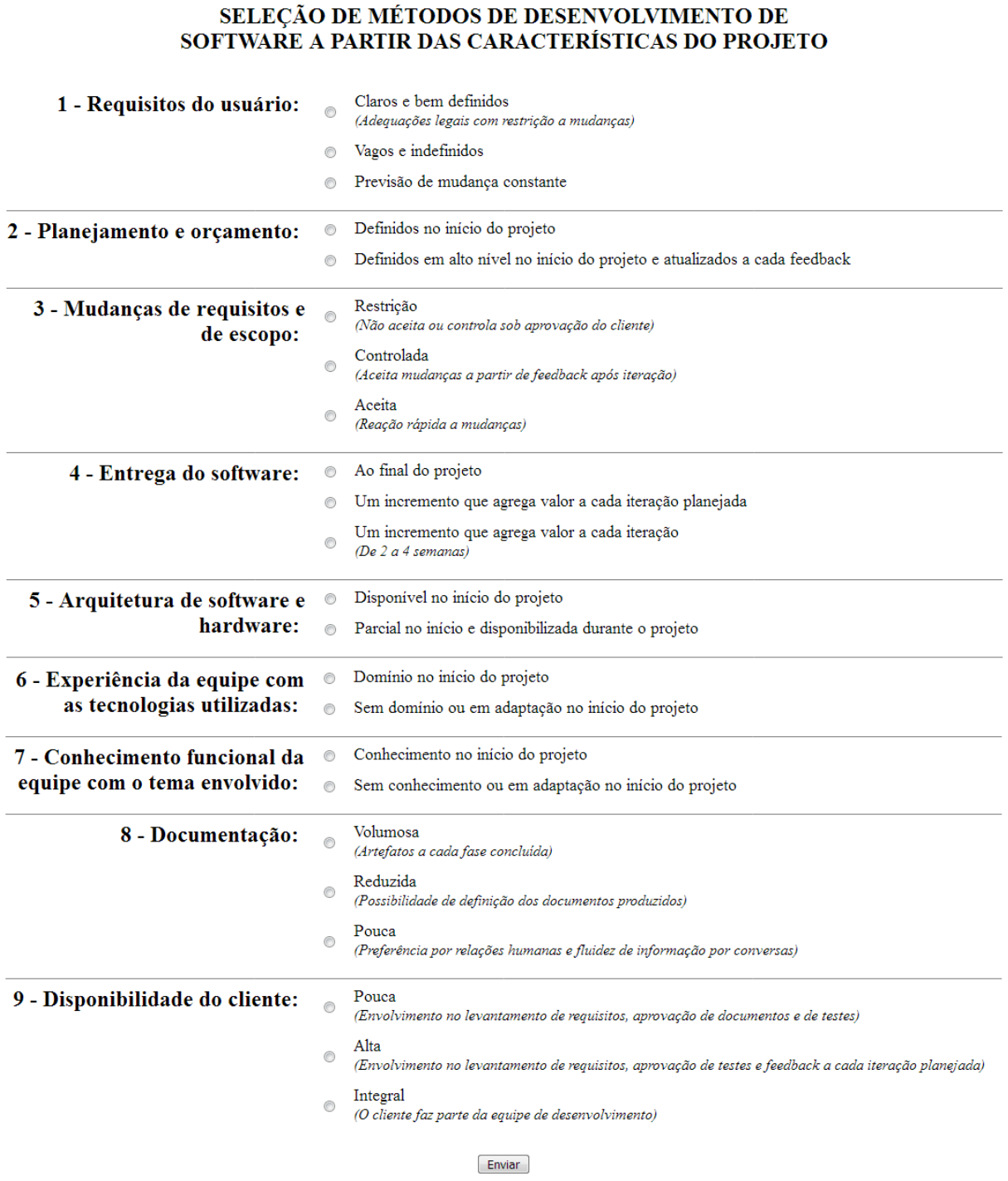

Figura 9 - Formulário de critérios e características de projeto da aplicação

Na figura 10, “Resultado da pesquisa com recomendação de métodos”, as características informadas pelo usuário resultaram na recomendação de métodos do ciclo de vida em projeto iterativo e incremental:

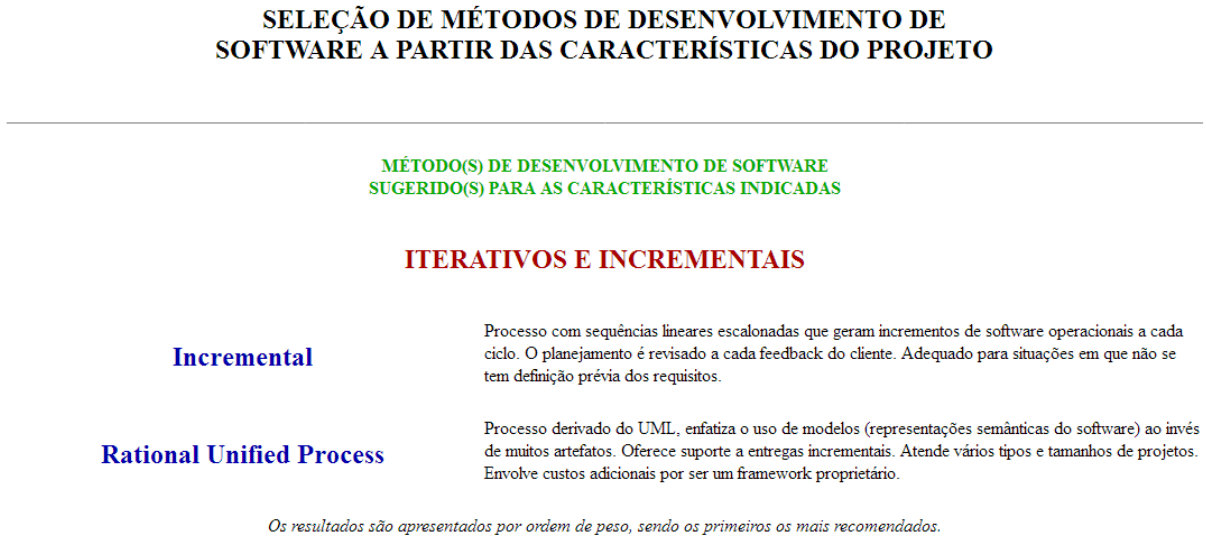

Figura 10 - Resultado da pesquisa com recomendação de métodos

Na figura 11, “Resultado da pesquisa sem recomendação de métodos”, as características informadas pelo usuário não estão associadas a nenhum método. Nesse caso sugere-se que seja feita uma revisão das condições do projeto:

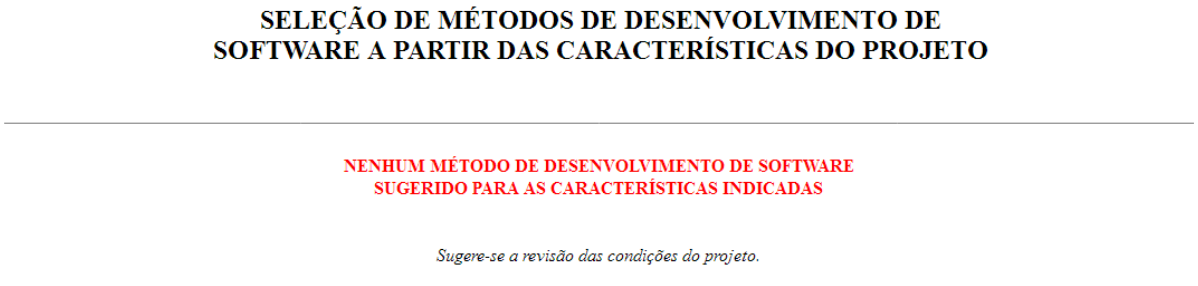

Figura 11 - Resultado da pesquisa sem recomendação de métodos

4.5. DEMONSTRAÇÃO DE APLICAÇÃO DAS REGRAS PARA SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE

Esta seção apresenta demonstrações de indicação de métodos de desenvolvimento de *software* feitos pela aplicação detalhada na seção 4.4, “Aplicação para a indicação de métodos de desenvolvimento de *software* a partir das características do projeto” e também de uma pesquisa sem resultados.

A figura 12, “Indicação de método de desenvolvimento de *software* com ciclo de vida predeterminado”, mostra a aplicação recomendando o método em cascata a partir do preenchimento do formulário com seguintes as opções: requisitos do usuário – claros e bem definidos; planejamento e orçamento – definidos no início do projeto; mudanças de requisitos e de escopo – restrição, entrega do *software* – ao final do projeto; arquitetura de *software* e *hardware* – disponível no início do projeto; experiência da equipe com as tecnologias utilizadas – domínio no início do projeto; conhecimento funcional da equipe com o tema envolvido – conhecimento no início do projeto; documentação – volumosa; disponibilidade do cliente – pouca.

A recomendação se encaixa em métodos do ciclo de vida em projetos predeterminado pois é um cenário em que o cliente não tem muita disponibilidade para participar do projeto, mas já tem o escopo bem definido, restrito a mudanças e aceita receber o *software* somente após toda a sua construção. Com isso, a necessidade por documentação volumosa e a antecipação de planejamento e orçamento não serão um problema, já que a situação provavelmente permanecerá estável até o final do projeto. Contribuem também para um desenvolvimento sequencial a disponibilidade imediata da arquitetura de *software* e *hardware* e os prévios domínios técnico e funcional da equipe de desenvolvimento.

SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

MÉTODO(S) DE DESENVOLVIMENTO DE SOFTWARE SUGERIDO(S) PARA AS CARACTERÍSTICAS INDICADAS

PREDETERMINADOS

Cascata

Proposta sequencial e sistemática com retroalimentação entre as etapas. Um artefato seguido de uma aprovação acontece ao fim de cada etapa. Exige planejamento antecipado de todas as tarefas e definição de todos os requisitos antes do início da fase de execução. Indicado para situações onde os requisitos estão bem definidos e há pouca ou nenhuma possibilidade de alteração de escopo. Uma versão do software será apresentada apenas na fase de testes, que é quando serão conhecidos os possíveis problemas no desenvolvimento.

Os resultados são apresentados por ordem de peso, sendo os primeiros os mais recomendados.

---

1 - Requisitos do usuário:
- (●) Claros e bem definidos *(Adequações legais com restrição a mudanças)*
- ( ) Vagos e indefinidos
- ( ) Previsão de mudança constante

2 - Planejamento e orçamento:
- (●) Definidos no início do projeto
- ( ) Definidos em alto nível no início do projeto e atualizados a cada feedback

3 - Mudanças de requisitos e de escopo:
- (●) Restrição *(Não aceita ou controla sob aprovação do cliente)*
- ( ) Controlada *(Aceita mudanças a partir de feedback após iteração)*
- ( ) Aceita *(Reação rápida a mudanças)*

4 - Entrega do software:
- (●) Ao final do projeto
- ( ) Um incremento que agrega valor a cada iteração planejada
- ( ) Um incremento que agrega valor a cada iteração *(De 2 a 4 semanas)*

5 - Arquitetura de software e hardware:
- (●) Disponível no início do projeto
- ( ) Parcial no início e disponibilizada durante o projeto

6 - Experiência da equipe com as tecnologias utilizadas:
- (●) Domínio no início do projeto
- ( ) Sem domínio ou em adaptação no início do projeto

7 - Conhecimento funcional da equipe com o tema envolvido:
- (●) Conhecimento no início do projeto
- ( ) Sem conhecimento ou em adaptação no início do projeto

8 - Documentação:
- (●) Volumosa *(Artefatos a cada fase concluída)*
- ( ) Reduzida *(Possibilidade de definição dos documentos produzidos)*
- ( ) Pouca *(Preferência por relações humanas e fluidez de informação por conversas)*

9 - Disponibilidade do cliente:
- (●) Pouca *(Envolvimento no levantamento de requisitos, aprovação de documentos e de testes)*
- ( ) Alta *(Envolvimento no levantamento de requisitos, aprovação de testes e feedback a cada iteração planejada)*
- ( ) Integral *(O cliente faz parte da equipe de desenvolvimento)*

[Enviar]

Figura 12 - Indicação de método de desenvolvimento de software com ciclo de vida predeterminado

A figura 13, “Indicação de método de desenvolvimento de software com ciclo de vida iterativo e incremental”, mostra a aplicação recomendando os métodos incremental e RUP a partir do preenchimento do formulário com seguintes as opções: requisitos do usuário – vagos e indefinidos; planejamento e orçamento – definidos em alto nível no início do projeto e atualizados a cada *feedback*; mudanças de requisitos e de escopo – controlada, entrega do *software* – um incremento que agrega valor a cada iteração planejada; arquitetura de *software* e *hardware* – parcial no início e disponibilizada durante o projeto; experiência da equipe com as tecnologias utilizadas – sem domínio ou em adaptação no início do projeto; conhecimento funcional da equipe com o tema envolvido – sem conhecimento ou em adaptação no início do projeto; documentação – reduzida; disponibilidade do cliente – alta.

A recomendação se encaixa em métodos do ciclo de vida em projetos iterativos e incrementais pois é um cenário em que o cliente tem disponibilidade para apoiar a equipe do projeto, mas ainda não tem uma ideia muito clara do escopo, está aberto a mudanças mesmo que controladas e aceita receber o *software* em várias entregas partindo de uma primeira versão básica até uma versão final completa. A possibilidade de documentação reduzida e de atualização de planejamento e orçamento a cada *feedback* viabiliza o trabalho, já que os requisitos provavelmente mudarão durante o projeto. Outras questões que exigem um desenvolvimento incremental são a falta da arquitetura de *software* e *hardware* no início do projeto e a necessidade de adequação técnica e funcional da equipe de desenvolvimento.

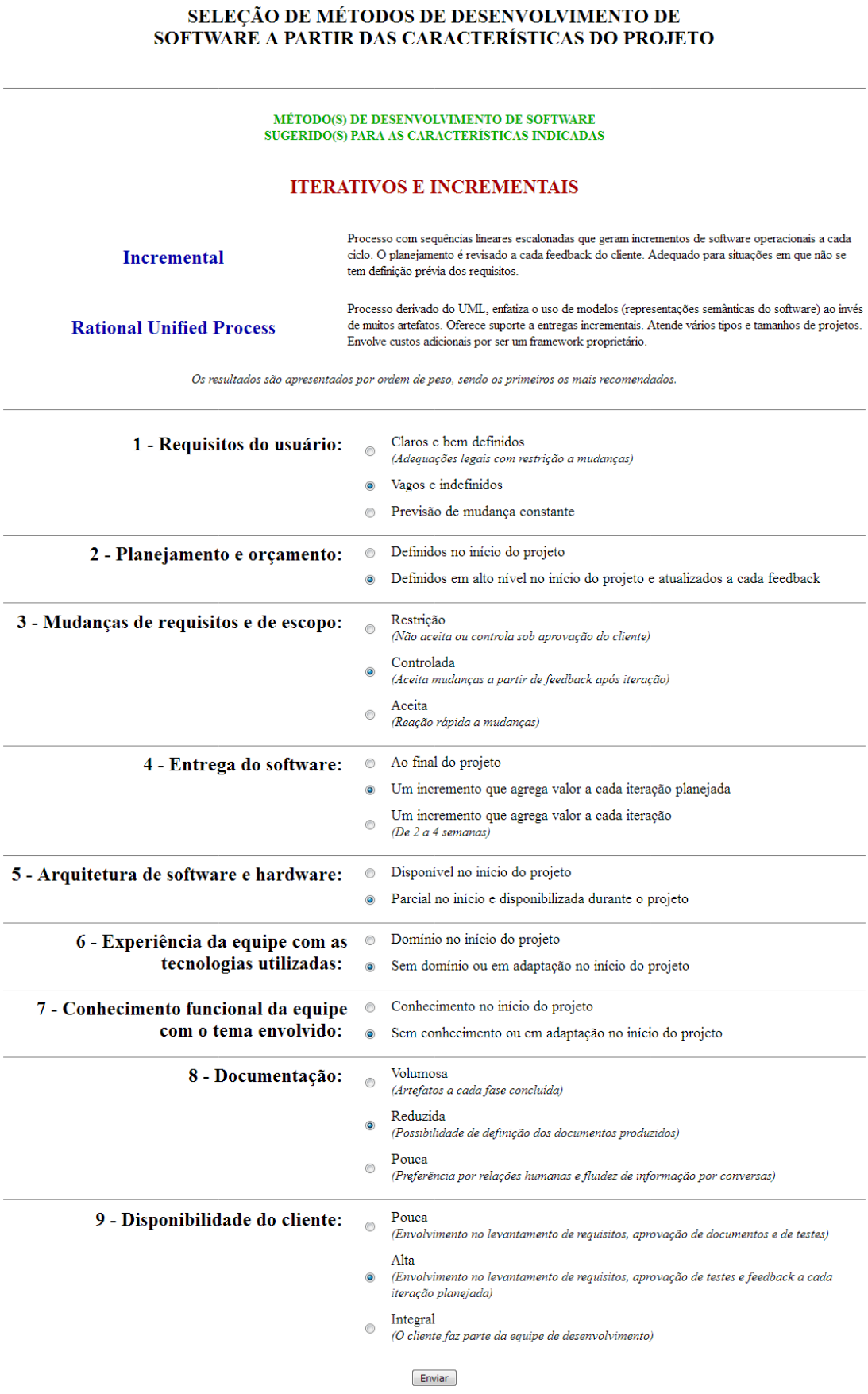

Figura 13 - Indicação de método de desenvolvimento de software com ciclo de vida iterativo e incremental

A figura 14, “Indicação de método de desenvolvimento de software com ciclo de vida adaptativo”, mostra a aplicação recomendando os métodos incremental e RUP a partir do preenchimento do formulário com seguintes as opções: requisitos do usuário – previsão de mudança constante; planejamento e orçamento – definidos em alto nível no início do projeto e atualizados a cada *feedback*; mudanças de requisitos e de escopo – aceita, entrega do *software* – um incremento que agrega valor a cada iteração; arquitetura de *software* e *hardware* – parcial no início e disponibilizada durante o projeto; experiência da equipe com as tecnologias utilizadas – sem domínio ou em adaptação no início do projeto; conhecimento funcional da equipe com o tema envolvido – sem conhecimento ou em adaptação no início do projeto; documentação – pouca; disponibilidade do cliente – integral.

A recomendação se encaixa em métodos do ciclo de vida em projetos adaptativo pois é um cenário em que o cliente tem disponibilidade integral para fazer parte da equipe do projeto e seus requisitos sofrerão mudança constante, aceita mudanças e receber o *software* em várias entregas construídas em fases curtas. A possibilidade de pouca documentação e de atualização de planejamento e orçamento a cada *feedback* viabiliza o trabalho, já que os requisitos provavelmente mudarão durante o projeto. Outras questões que exigem um desenvolvimento adaptativo são a falta da arquitetura de *software* e *hardware* no início do projeto e a necessidade de adequação técnica e funcional da equipe de desenvolvimento.

SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

MÉTODO(S) DE DESENVOLVIMENTO DE SOFTWARE SUGERIDO(S) PARA AS CARACTERÍSTICAS INDICADAS

ADAPTATIVOS

Adaptive Software Development	Proposta para a construção de sistemas complexos com colaboração humana e auto-organização de equipes. Dá ênfase a comunicação pessoal evitando documentações
Extreme Programming	Adequado para equipes pequenas e médias que desenvolvem com requisitos vagos e sujeitos a mudança constante. Os clientes fazem parte da equipe e participam das definições. Possibilidade de custos adicionais para atender ao princípio da codificação em pares.

Os resultados são apresentados por ordem de peso, sendo os primeiros os mais recomendados.

1 - Requisitos do usuário:	○ Claros e bem definidos *(Adequações legais com restrição a mudanças)* ○ Vagos e indefinidos ◉ Previsão de mudança constante
2 - Planejamento e orçamento:	○ Definidos no início do projeto ◉ Definidos em alto nível no início do projeto e atualizados a cada feedback
3 - Mudanças de requisitos e de escopo:	○ Restrição *(Não aceita ou controla sob aprovação do cliente)* ○ Controlada *(Aceita mudanças a partir de feedback após iteração)* ◉ Aceita *(Reação rápida a mudanças)*
4 - Entrega do software:	○ Ao final do projeto ○ Um incremento que agrega valor a cada iteração planejada ◉ Um incremento que agrega valor a cada iteração *(De 2 a 4 semanas)*
5 - Arquitetura de software e hardware:	○ Disponível no início do projeto ◉ Parcial no início e disponibilizada durante o projeto
6 - Experiência da equipe com as tecnologias utilizadas:	○ Domínio no início do projeto ◉ Sem domínio ou em adaptação no início do projeto
7 - Conhecimento funcional da equipe com o tema envolvido:	○ Conhecimento no início do projeto ◉ Sem conhecimento ou em adaptação no início do projeto
8 - Documentação:	○ Volumosa *(Artefatos a cada fase concluída)* ○ Reduzida *(Possibilidade de definição dos documentos produzidos)* ◉ Pouca *(Preferência por relações humanas e fluidez de informação por conversas)*
9 - Disponibilidade do cliente:	○ Pouca *(Envolvimento no levantamento de requisitos, aprovação de documentos e de testes)* ○ Alta *(Envolvimento no levantamento de requisitos, aprovação de testes e feedback a cada iteração planejada)* ◉ Integral *(O cliente faz parte da equipe de desenvolvimento)*

Enviar

Figura 14 - Indicação de método de desenvolvimento de software com ciclo de vida adaptativo

A figura 15, “Pesquisa sem resultado”, mostra a aplicação sugerindo uma revisão das condições e sinalizando que não há métodos compatíveis com as seguintes características do projeto: requisitos do usuário – vagos e indefinidos; planejamento e orçamento – definidos no início do projeto; mudanças de requisitos e de escopo – controlada, entrega do *software* – ao final do projeto; arquitetura de *software* e *hardware* – parcial no início e disponibilizada durante o projeto; experiência da equipe com as tecnologias utilizadas – domínio no início do projeto; conhecimento funcional da equipe com o tema envolvido – sem conhecimento ou em adaptação no início do projeto; documentação – pouca; disponibilidade do cliente – pouca.

Nesse caso não há recomendação de método pois é um cenário em que o cliente não tem muita disponibilidade para participar do projeto, não tem o escopo bem definido, está aberto a mudanças mesmo que controladas e aceita receber o *software* somente após toda a sua construção. Apesar de aceitar pouca documentação, pede a antecipação de planejamento e orçamento, o que seria um problema já que acontecerão mudanças durante o projeto que vão exigir retrabalho na revisão de documentos e descontentamento do cliente já que o planejamento e o orçamento serão impactados frequentemente por mudanças. A equipe de desenvolvimento possui domínio técnico desde o início, mas ainda precisa de adaptação no conhecimento funcional, o que causaria impactos no andamento do projeto e na compreensão de requisitos complexos. A falta da arquitetura de *software* e *hardware* no início do projeto também contribui para riscos de alteração de planejamento e orçamento.

SELEÇÃO DE MÉTODOS DE DESENVOLVIMENTO DE SOFTWARE A PARTIR DAS CARACTERÍSTICAS DO PROJETO

NENHUM MÉTODO DE DESENVOLVIMENTO DE SOFTWARE SUGERIDO PARA AS CARACTERÍSTICAS INDICADAS

Sugere-se a revisão das condições do projeto.

1 - Requisitos do usuário:	○ Claros e bem definidos *(Adequações legais com restrição a mudanças)* ◉ Vagos e indefinidos ○ Previsão de mudança constante
2 - Planejamento e orçamento:	◉ Definidos no início do projeto ○ Definidos em alto nível no início do projeto e atualizados a cada feedback
3 - Mudanças de requisitos e de escopo:	○ Restrição *(Não aceita ou controla sob aprovação do cliente)* ◉ Controlada *(Aceita mudanças a partir de feedback após iteração)* ○ Aceita *(Reação rápida a mudanças)*
4 - Entrega do software:	◉ Ao final do projeto ○ Um incremento que agrega valor a cada iteração planejada ○ Um incremento que agrega valor a cada iteração *(De 2 a 4 semanas)*
5 - Arquitetura de software e hardware:	○ Disponível no início do projeto ◉ Parcial no início e disponibilizada durante o projeto
6 - Experiência da equipe com as tecnologias utilizadas:	◉ Domínio no início do projeto ○ Sem domínio ou em adaptação no início do projeto
7 - Conhecimento funcional da equipe com o tema envolvido:	○ Conhecimento no início do projeto ◉ Sem conhecimento ou em adaptação no início do projeto
8 - Documentação:	○ Volumosa *(Artefatos a cada fase concluída)* ○ Reduzida *(Possibilidade de definição dos documentos produzidos)* ◉ Pouca *(Preferência por relações humanas e fluidez de informação por conversas)*
9 - Disponibilidade do cliente:	◉ Pouca *(Envolvimento no levantamento de requisitos, aprovação de documentos e de testes)* ○ Alta *(Envolvimento no levantamento de requisitos, aprovação de testes e feedback a cada iteração planejada)* ○ Integral *(O cliente faz parte da equipe de desenvolvimento)*

Enviar

Figura 15 - Pesquisa sem resultado

5. CONSIDERAÇÕES FINAIS

O levantamento bibliográfico mostra que os métodos de desenvolvimento de *software* são propostos para organizar a atividade de desenvolver *softwares* associando as atividades do modelo de processos genérico a um determinado ciclo de vida em projetos considerando princípios específicos. Na verdade, o que os diferencia de fato são os princípios. Por exemplo, os métodos RUP e incremental são baseados em um ciclo de vida iterativo e incremental, porém o princípio de utilização da UML para especificar requisitos através de representações semânticas ao invés de produzir documentações volumosas os diferencia.

Ao analisar os métodos é possível também perceber a possibilidade de adapta-los à uma necessidade específica, mas é importante manter seus princípios para que não se perca o foco. Por exemplo, se uma equipe resolve fazer uma adaptação ao implantar o método XP e desconsidera a inclusão do cliente como parte da equipe pode comprometer um dos princípios do manifesto ágil que é o de que pessoas de negócio e desenvolvedores devem trabalhar em conjunto diariamente por todo o projeto.

Se uma equipe utilizar um método predeterminado para atender a um cliente que não tem requisitos bem definidos poderá ter problemas para cumprir atividades. O mesmo se aplica a uma equipe que tentar aplicar um método ágil para atender a um cliente indisponível e que exige diversos documentos ao longo do projeto.

A decisão pela utilização de um método inadequado para as características do projeto pode comprometer a satisfação do cliente. Solicitar ao cliente todos os requisitos de um *software* antes do início do desenvolvimento e direcionar a solução funcional em um projeto no qual o cliente não tenha clareza dos requisitos ao invés de aumentar a compreensão dos requisitos em um processo incremental pode fazer com que o *software* entregue não atenda totalmente as necessidades do cliente.

A utilização do método inadequado pode comprometer também os prazos de um projeto. Apresentar uma data de entrega à um cliente para um projeto que tem prazo estipulado quando a equipe não tem domínio funcional do tema associado torna-se um risco pois haverá uma necessidade de adaptação que se não for respeitada pode gerar falhas e retrabalho que serão refletidos em ajustes de cronograma.

Outra questão importante a ser considerada pela equipe de desenvolvimento é se o projeto tem características suficientes para ser encaixado em algum método. Um projeto em que o cliente não tenha clareza dos requisitos e não tenha disponibilidade para apoiar a equipe pode ter problemas para alcançar uma maturidade da solução.

A aplicação para seleção dos métodos de desenvolvimento de *software* mostrou ser uma ferramenta importante no momento de se verificar a aderência do projeto a um método devido à complexidade da análise. Da forma como foi apresentada, a aplicação permite a parametrização de novos métodos, critérios e características.

A pesquisa foi limitada a cinco métodos de desenvolvimento de *software* distribuídos nos três ciclos de vida em projetos, de modo que foi possível verificar nove critérios direcionadores e vinte e três características distribuídas entre eles. Outros métodos analisados podem apresentar novos e novas características.

REFERÊNCIAS BIBLIOGRÁFICAS

AHMAR, M. Ayman Al. 2005. RULE BASED EXPERT SYSTEM FOR SELECTING SOFTWARE DEVELOPMENT METHODOLOGY. *Journal of Theoretical and Applied Information Technology.* 2, 2005, Vol. 19.

Executive Brief. 2008. Project Smart. *Which life cycle is best for your project?* [Online] 2008. [Citado em: 02 de 05 de 2017.] https://www.projectsmart.co.uk/which-life-cycle-is-best-for-your-project.php.

GOMES, Pedro Miguel Ribeiro Veloso. 2009. *Integração de modelos de desenvolvimento de software mais e menos ágeis.* Porto : Faculdade de Engenharia da Universidade do Porto, 2009.

KÖCHE, José Carlos. 2011. *Fundamentos de Metodologia Científica: Teoria da ciência e iniciação à pesquisa.* Petrópolis : Vozes, 2011.

MANIFESTO, Agile. 2001. *Manifesto for Agile Software Development.* [Online] 2001. [Citado em: 24 de 04 de 2017.] http://agilemanifesto.org/.

OLIVEIRA, Fernando Gonçalves de e SEABRA, João Manuel Pimentel. 2015. METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE: UMA ANÁLISE NO DESENVOLVIMENTO DE SISTEMAS NA WEB. *Tecnologias em Projeção.* 1, 2015, Vol. 6.

OZTURK, Veysi. 2013. Selection of appropriate software development life cycle using fuzzy logic. *Journal of Intelligent & Fuzzy Systems.* 3, 2013, Vol. 25.

PAIVA, Anabela, et al. 2011. Principais aspectos na avaliação do sucesso de projectos de desenvolvimento de software: Há alguma relação com o que é considerado noutras indústrias? *Interciência.* 3, 2011, Vol. 36.

PMI, Project Management Institute. 2013. *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®). — Quinta edição.* Newtown Square : Project Management Institute, Inc., 2013.

PRESSMAN, Roger S. 2011. *Engenharia de Software: Uma abordagem profissional - 7ª Edição.* Porto Alegre : AMGH, 2011.

RATIONAL. 1988. *Rational Unified Process: Best Practices for Software Development Teams.* Lexington : Rational Software, 1988.

ROYCE, Winston W. 1970. *Managing the development of large software systems.* s.l. : IEEE WESCON, 1970.

RUPARELIA, Nayan B. 2010. Software Development Lifecycle. *ACM SIGSOFT Software Engineering Notes.* 3, 2010, Vol. 35.

SEMEDO, Maria João Moreno. 2012. *Ganhos de produtividade e de sucesso de Metodologias Ágeis VS Metodologias em Cascata no desenvolvimento de projectos de software.* Lisboa : Universidade Lusófona de Humanidades e Tecnologias, 2012.

SHARMA, Manish. 2011. A Survey of project scenario impact in SDLC models selection process. *International Journal of Scientific & Engineering Research.* 7, 2011, Vol. 2.

SILVA, Jaime Humberto Samamé . 2013. *APLICACIÓN DE UNA METODOLOGÍA ÁGIL EN EL DESARROLLO DE UN SISTEMA DE INFORMACIÓN.* Pando : PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ ESCUELA DE POSGRADO, 2013.

SOARES, Michel dos Santos. 2004. *Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software.* Conselheiro Lafaiete : Revista Eletrônica de Sistemas de Informação, 2004.

SOMERVILLE, Ian. 2011. *Software Engeneering - 9th edition.* Massachusetts : Addison-Wesley, 2011.

The Standish Group Internacional, Inc. 2014. *Chaos Report.* s.l. : The Standish Group Report, 2014.

TULI, Anupriya, et al. 2014. Empirical Investigation of Agile Software Development: A Cloud Perspective. *ACM SIGSOFT Software Engineering Notes.* 4, 2014, Vol. 39.

VIJAYASARATHY, Leo R. e BUTLER, Charles W. 2016. Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter? *FOCUS: THE SOFTWARE ENGINEERING PROCESS.* 740, 2016, Vol. 7459/16.

WINGO, R. Steven e TANIK, Murat M. 2015. *Using an Agile Software Development Methodology for a Complex Problem Domain.* Fort Lauderdale : IEEE, 2015.