UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM LICENCIATURA EM COMPUTAÇÃO

RODRIGO BRAZÃO TEIXEIRA

Análise de processos de desenvolvimento de software, tradicional e ágil, com foco em sistemas médicos.

CAMPINA GRANDE – PB
2014

RODRIGO BRAZÃO TEIXEIRA

Análise de processos de desenvolvimento de software, tradicional e ágil, com foco em sistemas médicos.

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação.

Orientador: Frederico Moreira Bublitz

CAMPINA GRANDE – PB
2014

T266a Teixeira, Rodrigo Brazão.
Análise de processos de desenvolvimento de software, tradicional e ágil, com foco em sistemas médicos [manuscrito] / Rodrigo Brazão Teixeira. - 2014.
19 p. : il. color.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2014.
"Orientação: Prof. Dr. Frederico Moreira Bublitz, Departamento de Computação".

1. Desenvolvimento de software. 2. Metodologia tradicional. 3. Metodologia ágil. 4. Sistemas críticos. I. Título.

21. ed. CDD 005.2

RODRIGO BRAZÃO TEIXEIRA

Análise de processos de desenvolvimento de software, tradicional e ágil, com foco em sistemas médicos.

Trabalho de Conclusão de Curso apresentado ao Curso de Licenciatura em Computação da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação

Aprovada em 25/03/2014.

Prof. Dr. Frederico Moreira Bublitz / UEPB
Orientador

Profª Msc. Ana Isabella Muniz Leite/ UEPB
Examinadora

Profª Drª Luciana de Queiroz Leal Gomes / UEPB
Examinadora

Análise de processos de desenvolvimento de software, tradicional e ágil, com foco em sistemas críticos.

BRAZÃO, Rodrigo Teixeira[1], BUBLITZ, Frederico[2]

RESUMO

O uso de sistemas inseridos na área da saúde se torna cada vez mais comum com a finalidade de melhorar a prestação de serviços a pacientes. Sistemas como estes são considerados como críticos, pelo fato de que suas consequências de fracasso podem gerar prejuízos elevados, desde danos ao meio ambiente até a morte em humanos. Portanto, desenvolver um software com estas particularidades não é tarefa fácil, faz-se necessário o uso de um processo estruturado que se enquadre a todas as características inerentes a esta finalidade. Nesse sentido, este trabalho identifica as principais metodologias existentes, elenca as prioridades que devem ser observadas em sistemas críticos e, por fim, realiza um comparativo entre metodologias com foco em sistemas críticos, identificando como cada metodologia se aplica diante de projetos como este.

PALAVRAS-CHAVE: Processos de desenvolvimento de software. Metodologia Tradicional. Metodologia Ágil. Sistemas Críticos.

1 INTRODUÇÃO

Com o avanço tecnológico, o número de produtos de software desenvolvidos para automatizar atividades do nosso cotidiano vem crescendo. Esses sistemas são aplicados nas mais diversas áreas sempre visando otimizar os processos e/ou reduzir os custos (SOMMERVILLE, 2007). Uma tendência é o desenvolvimento de sistemas para assistência à saúde, que é motivada, principalmente, pela necessidade de redução de custos mediante a conjuntura atual em que a população global está envelhecendo e necessita de mais cuidados com a saúde. Assim, como afirmado por COOPER (2008), esses sistemas atuam principalmente na automatização de processos para melhorar o atendimento de pacientes, viabilizando uma melhoria na qualidade de vida deles.

Sistemas desenvolvidos com foco na área médica são considerados e tratados como críticos, pois lidam com vidas, tendo um elevado índice de risco que, no caso de falha, podem acarretar desde uma lesão à morte de um paciente. Sistema crítico é aquele que pode causar danos a vida ou ao ambiente (GOVE et. al.,1991). Esses sistemas são regulados por normas e certificações de qualidade que atestam a segurança do software desenvolvido. Mas, atingir os padrões de segurança exigidos não é tarefa simples. Esses órgãos exigem documentação extensa para atestar a integridade e qualidade do software (WARREN, 2011).

O uso de processos de desenvolvimento de software seguindo o modelo tradicional para este tipo de sistema ainda sofre percalços (GARCIA et. al., 2005). Apesar de o desenvolvimento fazer uso de extensa documentação, alguns processos, a exemplo do modelo em cascata, só realizam testes no fim do desenvolvimento e só se tem uma versão funcional do sistema após muito tempo de desenvolvimento, dificultando a gestão dos riscos e a realização da validação (SCOTT et. al., 2011).

[1]Aluno graduando em Computação – Universidade Estadual da Paraíba - UEPB
[2] Orientador – Universidade Estadual da Paraíba - UEPB

Em contrapartida, as metodologias ágeis, a exemplo do XP (eXtreme Programming), geralmente se preocupam com o desenvolvimento de testes à medida que o sistema é desenvolvido e sempre há uma versão funcionando a cada iteração, otimizando o *time-to-market* (AFONSO et. al., 2008). As metodologias ágeis têm como características a adaptabilidade e agilidade em tratar erros ou requisitos inesperados no ciclo de vida do projeto. Com este desígnio ágil, requisitos como eficácia, qualidade, flexibilidade, facilitam o alcance de resultados satisfatórios e aceitáveis em projetos (SOMMERVILLE, 2007). Entretanto, não é dada significativa atenção à documentação e especificação formal que é importante num sistema crítico. Com isso, na visão de SIPONENA et. al. (2005), faz-se necessário um processo de desenvolvimento de software que atenda as propriedades inerentes a sistemas médicos e de uma maneira dedicada e ágil.

Nesse contexto, o desenvolvimento de sistemas na área da saúde requer uma abordagem dirigida que, além de auxiliar na condução do processo de desenvolvimento, também se enquadre no tratamento de requisitos específicos que estes produtos de software necessitam. Dessa forma, os processos podem auxiliar de modo ativo todas as particularidades de sistemas críticos minimizando o trabalho da equipe de desenvolvimento. Assim, levanta-se a questão da pesquisa: Como os processos de software dão suporte ao desenvolvimento de sistemas críticos com foco em saúde? Com isso, o presente trabalho busca fazer análises de processos na literatura existente para identificar e avaliar dentre as metodologias, quais processos melhor se relaciona no foco em riscos, ressaltando o foco em risco de produto, que atenta para a prevenção e segurança de software, e se os mesmos necessitam ser remodelados para serem aceitos por órgãos reguladores do sistema de saúde, caso não haja nenhuma proposta adequada para esta linha de desenvolvimento.

O artigo se estrutura por meio de seções que embasam cada uma delas. Assim a seção 2 expõe um pouco a cerca dos Processos de desenvolvimento de software, como surgiram e alguns exemplos. Na seção 2.1 e 2.2, é diferenciado a Metodologia Tradicional da Metodologia ágil respectivamente. Já na seção 3, detalha características importantes quanto ao desenvolvimento de sistemas com características críticas, é feita uma introdução e posteriormente são elencados critérios que estes sistemas devem se enquadrar, conforme verificado na literatura pesquisada. Na seção 4, faz-se um comparativo, unindo as metodologias apresentadas, com os critérios destacados, identificando quais processos inerentes a cada metodologia, se enquadram nesses critérios estabelecidos. E na seção 5, é apresentada a conclusão do trabalho realizado.

2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Os processos de desenvolvimento de software surgiram numa época de crise de software, onde se fez necessário sistematizar o trabalho de forma consistente para solucionar problemas cada vez maiores e mais complexos. Assim, houve a necessidade de tornar o processo de desenvolvimento estruturado e unificado, atendendo a todas as necessidades de uma forma compensadora (MAINART et. al., 2010).

Historicamente, um dos primeiros modelos de processos de desenvolvimento de software foi o modelo de código e correção, que se baseava em duas etapas: escrever o código e em seguida corrigir os possíveis problemas do código (MISRA et. al., 2012). Adiante, um processo que se firmou durante anos foi o método em cascata de desenvolvimento suprindo limitações de outros modelos. O modelo em cascata, mostrado na Figura 1, é uma abordagem sugerida por Royce em 1970 (SANTOS, 2004), onde a ideia principal é a sequência de etapas definidas por meio de atividades fundamentais como análise e definição de requisitos, projeto de sistema e software, implementação e testes de unidade, integração e testes de sistemas, operação e manutenção, na qual cada etapa compreende em um documento aprovado e uma etapa só se inicia após o término da anterior (SOMMERVILLE, 2007).

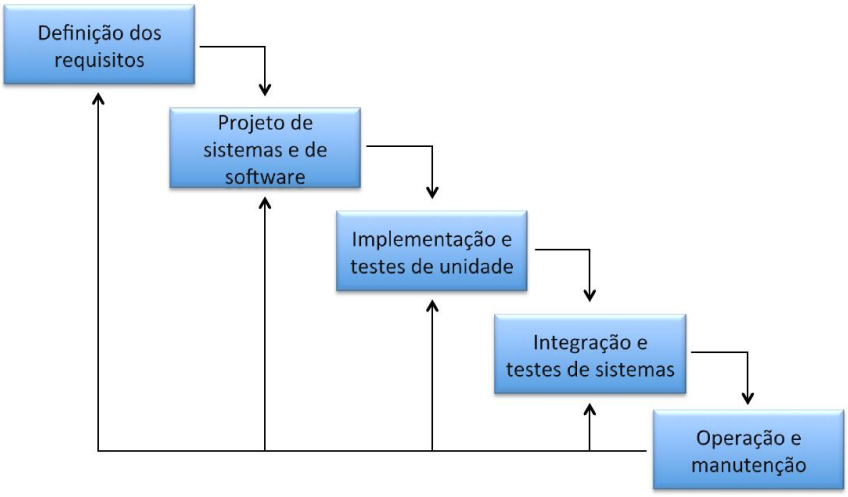

Figura 1. Modelo em Cascata (SOMMERVILLE, 2007).

Mas, com o tempo, notou-se que o modelo também tinha suas restrições, porém funcionou de modo satisfatório no desenvolvimento de sistemas operacionais e compiladores. Em seguida, o método espiral ganhou notoriedade (MISRA et. al., 2012). Este modelo, observado na Figura 2, é representado como um espiral, onde cada iteração representa uma fase, num total de quatro, que envolve definição de objetivos, avaliação e redução de riscos, desenvolvimento e validação e planejamento (SOMMERVILE, 2007). Esses modelos descrevem um método de desenvolvimento, a chamada metodologia tradicional.

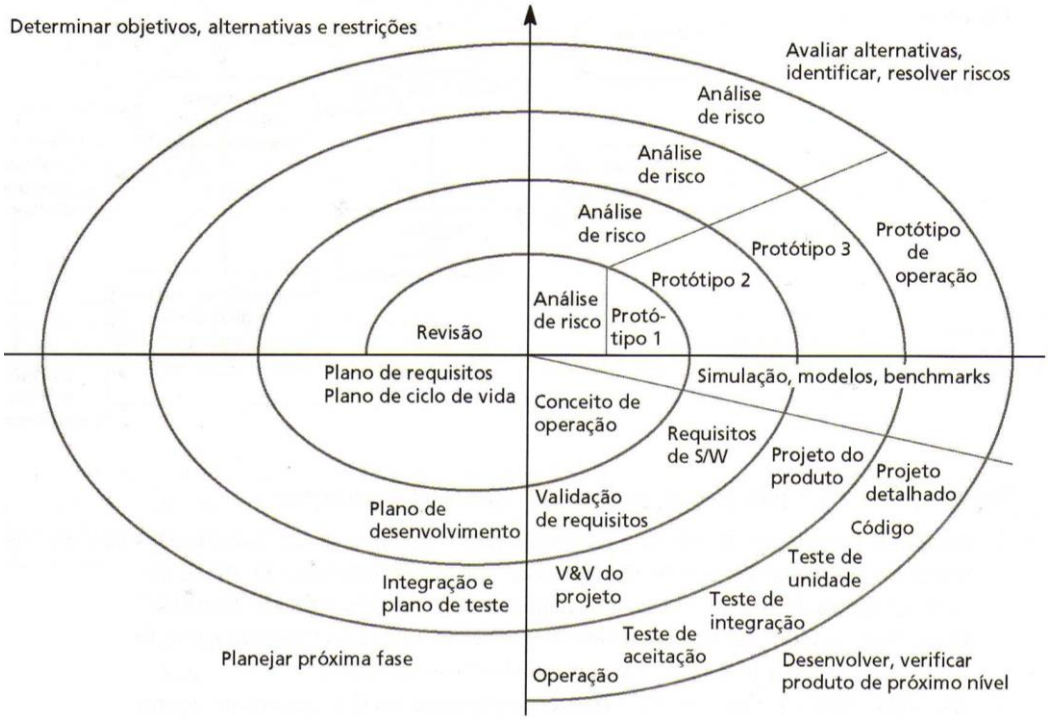

Figura 2. Modelo em Espiral (SOMMERVILLE, 2007).

Em evolução a esta linha de desenvolvimento, surge à metodologia ágil, caracterizando-se pelo modo adaptativo e partilhando a ideia de que o cliente identifica suas necessidades à medida que manipula o que já foi desenvolvido (GARCIA et. al., 2005).

O uso de processos no desenvolvimento de software enfrenta grandes pressões, considerando que a compreensão, adaptação e uma gestão ativa do processo de desenvolvimento são

fatores importantes para o êxito em projetos. Isso melhora a qualidade, reduz os custos, tempo e ajuda as partes interessadas (SOMMERVILLE, 2004). Neste contexto, o processo é um arcabouço bem definido e inter-relacionado que auxilia na construção de um software, de forma a atender todos os requisitos estabelecidos em um projeto com qualidade e eficiência (MADAN, 1997). Ainda segundo MADAN (1997), processo é definido como tarefas que constroem os requisitos estabelecidos pelo usuário em software.

2.1 METODOLOGIA TRADICIONAL

O modelo tradicional é marcado pela rigidez, resultado da definição de passos sequenciais bem definidos. Distingue-se pela formalidade, controle e rigor, onde o sucesso alcançado é fruto do que foi planejado (PRESSMAN, 2006). Segundo MAINART e SANTOS (2010) apud Oliveira, seu foco principal é conhecer todos os requisitos antes do projeto iniciar, possibilitando um melhor planejamento, facilitando a gerência e mantendo o processo rigoroso. Ainda segundo MAINART e SANTOS (2010):

> *As metodologias consideradas tradicionais, também chamadas de "pesadas", têm como característica marcante serem divididas em etapas e/ou fases. Essas fases são muito bem definidas e englobam atividades como Análise, Modelagem, Desenvolvimento e Testes.*

Um processo que marca essa proposta e é muito utilizado até hoje é o Modelo em Cascata ou Modelo Clássico (SANTOS, 2004), em que a sequência é priorizada, de forma que uma etapa só se inicie após o término da anterior e a cada etapa finalizada, uma documentação deve ser aprovada para início da seguinte. Em suma, é um processo disciplinado de tarefas na construção de um projeto buscando garantir as exigências do cliente no tempo e custos previstos (EDUARDO et. al., 2007).

A seguir, encontra-se a síntese de alguns dos principais processos inerentes à metodologia tradicional segundo as pesquisas realizada por (MAINART e SANTOS, 2010), (SANTOS, 2004) e (EDUARDO e MASSAKI, 2007):

Tabela 1. Principais Métodos Tradicionais

Método Tradicional	**Descrição**
Espiral	O Modelo Espiral é um modelo iterativo. É muito utilizado no desenvolvimento de softwares em conjunto com o paradigma da orientação a objetos, onde o desenvolvimento em módulos, somado ao processo de integração, se encaixa nos conceitos do paradigma. Possui ainda, uma fase que trata de riscos.
Incremental	Assume que o software desenvolvido pode sempre crescer e agregar novas funcionalidades, sendo que cada uma dessas funcionalidades, ou o conjunto delas, será desenvolvido por um incremento que segue todas as etapas descritas no modelo linear.
Cascata	É um modelo sequencial dividido em fases distintas sem possibilidades de alteração, ideal para projetos com requisitos bem definidos e compreendidos. Cada fase inicia-se após o término da anterior.

Fonte: (MAINART et. al.,2010), (SANTOS, 2004) e (EDUARDO et. al., 2007).

2.2 METODOLOGIA ÁGIL

A metodologia ágil surgiu em 2001 através do Manifesto Ágil, proposto por BECK et. al. (2001), um grupo de dezessete especialistas em processos, que juntos propuseram conceitos e princípios comuns entre os métodos de desenvolvimento, doze no total, que regem e auxiliam no desenvolvimento ágil. Destes princípios, destacam-se: a maior prioridade é satisfazer o cliente, mudanças inesperadas nos requisitos são bem vindas e entregar frequentemente o software funcionando.

Nos métodos ágeis busca-se constantemente o desenvolvimento de práticas que minimizem os índices de erros em projetos por meio de técnicas ágeis (GARCIA et. al., 2005). Têm como objetivo a satisfação do cliente por meio de entregas parciais do produto proporcionando uma visibilidade maior do software e feedbacks simultâneos pelo mesmo (MISRA et. al., 2012).

A seguir, encontra-se a descrição dos principais métodos ágeis de desenvolvimento de acordo com a pesquisa realizada por DYBA e DINGSOYR (2008):

Tabela 2. Principais Métodos Ágeis

Método Ágil	Descrição
Crystal	O método centra-se na comunicação em pequenas equipes de desenvolvimento de software. O desenvolvimento tem sete características: entrega frequente, melhoria reflexiva, comunicação osmótica (aberta para ouvir e intervir no ambiente de comunicação), segurança pessoal, o foco, fácil acesso a usuários experientes e os requisitos para o ambiente técnico.
Feature-driven development (Desenvolvimento orientado a funcionalidade)	Combina desenvolvimento ágil orientado por modelo com ênfase no modelo inicial de objeto, a divisão do trabalho em recursos, e design interativo para cada recurso. Uma iteração de um recurso consiste em duas fases: concepção e desenvolvimento.
Lean software development	Uma adaptação dos princípios da produção enxuta. Consiste em sete princípios: eliminar o desperdício, ampliar o aprendizado, decidir o mais tarde possível, entregar o mais rápido possível, capacitar à equipe, construir integridade e ver o todo.
Scrum	Concentra-se em gerenciamento de projetos em situações em que é difícil de planejar com antecedência, com mecanismos para "controle de processos empírico"; onde iterações de feedback constituem o elemento central. O software é desenvolvido por uma equipe de auto-organização por meio de fases chamadas de "sprints". Recursos a serem implementados no sistema estão registrados em uma carteira. Em seguida, o proprietário do produto decide quais itens do backlog deve ser desenvolvido na seguinte sprint. Os membros da equipe coordenam o seu trabalho em uma reunião diária, chamada "stand-up". Um membro da equipe, o scrum master, é responsável por resolver os problemas que impedem a equipe de trabalhar de forma eficaz.
Extreme Programming – XP	Concentra-se nas melhores práticas para o desenvolvimento. Consiste em doze práticas: o jogo de planejamento, fases pequenas, metáfora, design simples, teste, refatoração, programação em pares, propriedade coletiva, integração contínua, semana de 40hrs, clientes no local e os padrões de codificação.

Fonte: DYBA et. al. (2008)

Confrontando as duas metodologias vistas na seção 2.1 e 2.2, identificou-se na pesquisa realizada por DYBA et. al. (2008), um resumo de algumas das principais diferenças entre o desenvolvimento tradicional e o desenvolvimento ágil, conforme é apresentado na Tabela 3, a seguir:

Tabela 3. Principais diferenças entre Metodologias

	Desenvolvimento Tradicional	**Desenvolvimento Ágil**
Pressuposto Fundamental	Os sistemas são plenamente determináveis, previsíveis e são construídos através de um planejamento meticuloso e extenso.	Software adaptativo de alta qualidade é desenvolvido por pequenas equipes que usam o princípio do design contínuo de aperfeiçoamento e testes com base no feedback rápido e mudanças.
Estilo de Gestão	Comando e controle	Liderança e colaboração
Gestão de Conhecimento	Explícito	Implícito
Comunicação	Formal	Informal
Modelo de Desenvolvimento	Ciclo de vida do modelo (cascata, espiral ou alguma variação).	Modelo de entrega evolutiva.
Pretensão Organizacional Forma / Estrutura	Mecanicistas (burocrática com a alta formalização), destinada a grandes organizações.	Orgânico (flexível e participativa, incentivando a ação cooperativa social), destinada a pequenas e médias organizações.
Controle de Qualidade	Planejamento denso e controle rigoroso. Testes.	Controle contínuo dos requisitos de design e soluções. Teste contínuo.

Fonte: DYBA et. al. (2008)

3 SISTEMAS CRÍTICOS

O desenvolvimento de sistemas críticos está sendo amplamente empregado em diversas áreas, a exemplo de sistemas de aviação, usinas nucleares e na assistência à saúde, principalmente em dispositivos médicos (LINDHOLM et. al., 2011). Segundo RENATO (2011):

> *Estamos em uma época de rápido avanço tecnológico da comunicação móvel, da computação embarcada e miniaturização dos dispositivos e sensores eletrônicos. A aplicação destas tecnologias na área médica poderá trazer diversos benefícios aos pacientes e otimizar a prestação de serviços dos profissionais da saúde.*

Ainda segundo RENATO (2011), o uso adequado de novas tecnologias é essencial nos cuidados de pacientes para amenizar erros médicos, trazendo mais segurança no tratamento de doentes. Como exemplo de sistemas na área da saúde existe: marca passo, desfibriladores, máquina

de ressonância, dispositivos médicos em geral. Desenvolver esse tipo de sistema necessita de uma preocupação especial quanto ao processo de desenvolvimento, devido à segurança que exige, pois deve atentar para requisitos como segurança, confiabilidade e integridade (MADAN, 1997). Essencialmente, as operações de um sistema crítico sempre devem ser seguras, de modo que nunca cause danos às pessoas ou ao ambiente, mesmo em caso de falhas (SOMMERVILLE, 2007). A importância no rigor em tratar certos requisitos nestes sistemas se dá pelo fato de que suas consequências de fracasso podem gerar danos elevados, desde danos ao meio ambiente, como a morte em humanos (GOVE et. al.,1991). Um exemplo real de acidente com sistemas críticos conhecido foi o Therac-25, que ocorreu entre os anos 85 e 87, envolvendo uma máquina de radioterapia através de overdoses, desencadeando mortes e ferimentos graves (PHANI et. al., 2010).

Uma característica importante que deve ser levada em conta no desenvolvimento de um sistema crítico é a confiabilidade, pois se o mesmo não oferecer confiança, seus usuários não irão querer fazer uso dele, como também irão generalizar a qualidade de outros produtos da mesma empresa e em decorrência disso, os custos dessa falha serão muito maiores (SOMMERVILLE, 2007). Neste sentido, o uso de critérios bem definidos buscando garantir a eficácia de um sistema se faz necessário.

3.1. CRITÉRIOS DE SEGURANÇA

De acordo com PROJECT MANAGEMENT INSTITUTE (2008), o risco é a incerteza que, caso ocorra, afeta um dos objetivos do projeto. Esses objetivos envolvem custos, cronograma, escopo e qualidade. Com isso, o risco pode ter diversas causas e impactos. Neste sentido, o processo de desenvolvimento de um sistema crítico requer análises e conceitos mais estruturados para lidar com esta proposta (WUNRAM, 1993). Assim, com base na literatura pesquisada, verificou ser uma constante no tratamento de risco, alguns critérios base. Desse modo, utilizou-se o guia PMBOK[3] como referência principal e foram elencados cinco critérios, que compreendem em:

- **Identificação de riscos:** Identificam-se eventos que possam ameaçar ou afetar a qualidade e funcionalidades do software.
- **Análise de Riscos e ameaças:** Etapa em que se reflete sobre os riscos identificados, seus possíveis danos e como o trabalho deve seguir diante deles, priorizando os de maior probabilidade.
- **Estimativa de segurança:** Estabelece associações entre um evento/risco e uma ação de segurança.
- **Integridade de segurança:** É o critério mais importante, onde o sistema deve ser capaz de realizar operações automaticamente, visando prever, detectar e eliminar um risco.
- **Monitoramento:** Verificar e controlar os riscos conhecidos, além de produzir relatórios que comprovem esta constatação.

A seguir será discorrido sobre cada critério.

3.1.1 IDENTIFICAÇÃO DE RISCOS

Segundo COYLE et. al. (2009), esta é a fase mais importante no tratamento de risco e tem como finalidade perceber as possíveis ameaças de um sistema com antecedência, de modo que,

[3] Um Guia do Conhecimento em Gerenciamento de Projetos

minimize seus efeitos ou até os anule, como também se evite maiores erros, custos e o fracasso do projeto.

É importante que cada risco identificado receba uma breve descrição de risco para facilitar a percepção no momento em que o recurso for tratado pela equipe de desenvolvimento. Sugere-se que haja uma lista de verificação para identificar os pontos de alto risco. A identificação de um risco pode ocorrer nas reuniões de planejamento e pode gerar muito tempo. Aconselha-se, ainda, classificar o risco (baixo, médio, alto) para visibilidade da equipe por meio de notas com cores, vermelha, por exemplo, para riscos de alto impacto e probabilidade (YLIMANNELA, 2012).

É monitorada através de análises de segurança de software durante todo o processo. Assim, precisam ser reconhecidos todos os riscos para serem considerados durante todo o desenvolvimento (PHANI et. al., 2010).

3.1.2 ANÁLISE DE RISCOS E AMEAÇAS

De acordo com PROJECT MANAGEMENT INSTITUTE (2008), esta análise avalia o grau e as consequências de ocorrência de um risco. Faz-se a junção dos riscos e o planejamento sobre que decisão tomar no decorrer do projeto tomando como referência todos os possíveis riscos. Nesse contexto, a segurança é um fator predominante que deve estar incluso durante todo o ciclo de vida do software, pois garante que o sistema não oferecerá perigo aos usuários, e a atividade que dirige se essa garantia é a gestão de riscos (MCCAFFERY et. al., 2009).

Vale ressaltar que é importante identificar os principais requisitos de segurança, classificá-los, bem como apontar as pessoas autorizadas que possam interferir na segurança do software. Ou seja, definir pessoas específicas a lidar com determinados assuntos garantindo a segurança do todo, onde certos assuntos ficam sob sigilo dos principais responsáveis do software (SIPONENA et. al., 2005).

Segundo BOEHM et. al. (2003), também é necessário prover de uma base para tomar decisões quanto à estratégia de desenvolvimento. São identificados os candidatos a risco possibilitando a análise e servindo como guia no auxílio ao planejamento. Assim, revisões extensivas são essenciais durante o processo de desenvolvimento orientado à segurança (PHANI et. al., 2010).

3.1.3 ESTIMATIVA DE SEGURANÇA

De acordo com (ISO, 2007), a ISO 14971 detalha um processo que identifica perigos associados ao desenvolvimento de produtos destinados a saúde e estima os riscos destes perigos. Assim, a estimativa de risco é a redução da incerteza, onde se estabelece uma associação entre um evento e uma ação de segurança determinando a extensão do risco (COYLE et. al., 2009).

Para isso, uma equipe de risco, de preferência multidisciplinar, analisa os riscos de um sistema e ao término da analise terá uma ordem de riscos priorizando os mais graves dos mais leves, e em seguida é proposto medidas de controle determinando um limite aceitável entre os riscos identificados. Na visão de PROJECT MANAGEMENT INSTITUTE (2008), é importante destacar que as medidas tomadas devem se ajustar a relevância do risco.

Dessa forma, uma placa de risco pode ser utilizada para evidenciar os riscos de maior e menor prioridade, identificados por notas vermelhas e amarelas respectivamente, e com um “X”, marcando a nota de risco já tenha sido tratada. Isto permite que a equipe de desenvolvimento tenha maior visibilidade dos riscos que já foram ou ainda serão tratados (YLIMANNELA, 2012).

3.1.4 INTEGRIDADE DE SEGURANÇA

Preferencialmente os sistemas devem ser simples, mas de segurança atestada, evitando ao máximo a possibilidade de erro. Segundo EDUARDO et. al. (2007):

> *Segurança é considerada um requisito não funcional ou a qualidade de um sistema e que deve ser considerado em todo o ciclo de desenvolvimento.*

De acordo com PHANI et. al. (2010), segurança de software envolve:

- Integrar a segurança no ciclo de vida do software;
- Analisar o software, sistema e interfaces do começo ao fim;
- Planos de segurança de documentação, decisões, processos e resultados;
- Rastrear os requisitos de segurança de software em todas as fases de software;
- Relatórios e resolução de problemas e divergências;

Segundo SOMMERVILLE (2007), a chave para garantir a segurança é assegurar que o acidente não ocorra ou que as consequências de sua ocorrência sejam ínfimas, e isso pode ser conseguido de três maneiras:

- Prevenção de perigos: O sistema é projetado para que os perigos sejam evitados. Por exemplo, um sistema de corte, que requer que o operador pressione dois botões separados simultaneamente para operar a máquina, evita o perigo de as mãos do operador estarem no caminho da lâmina.
- Detecção e remoção de perigos: Perigos devem ser detectados e removidos antes de causarem um acidente. Por exemplo, um sistema de uma indústria química pode detectar a pressão excessiva e abrir uma válvula de alívio para reduzir a pressão, antes que ocorra uma explosão.
- Limitação de danos: Deve incluir soluções de proteção que minimizem os danos resultantes de um acidente. Por exemplo, um motor de aeronave normalmente inclui um extintor automático de incêndio. Se ocorrer um incêndio, ele poderá ser controlado antes que ameace a aeronave.

3.1.5 MONITORAMENTO

Comumente, o foco está na parte da engenharia de software na busca de entender e especificar ao máximo o software e minimizar os erros, mas se dá pouca atenção a constante verificação de software e é neste ponto que sistemas complexos necessitam de atenção. Assim, rever ou adaptar o processo de desenvolvimento se faz necessário para aplicações críticas (WUNRAM, 1993).

Segundo GOVE et al. (1991), idealmente o sistema crítico deve se enquadrar num estado seguro, porém, ainda que o sistema não esteja em um estado propriamente dito seguro, este pode ser considerado seguro se ele pode se recuperar de um estado perigoso. Mas atingir este patamar em um projeto não é tarefa fácil. É importante também que haja uma avaliação do Software com relação a sua segurança, para certificar e atestar que o sistema atende a todos os requisitos de segurança. Essas informações ficam registradas num relatório ou documentação contendo todos os resultados do processo de testes (PHANI et. al., 2010). Esta documentação é necessária e será apresentada posteriormente a órgãos reguladores, a fim de obter certificação do sistema desenvolvido.

De acordo com PROJECT MANAGEMENT INSTITUTE (2008), é importante também controlar os riscos fazendo acompanhamentos e avaliações contínuas durante todo o projeto, desde riscos já identificados até novos riscos, facilitando a adoção de medidas corretivas.

4 COMPARATIVO DE METODOLOGIAS

O comparativo entre processos tem como objetivo avaliar o progresso do processo de desenvolvimento de software, identificando vantagens e desvantagens que sirvam de base na redução de erros e qual a melhor abordagem a ser utilizada no tratamento de riscos em sistemas críticos na área da saúde.

Como pôde ser visto anteriormente nesta pesquisa na seção 3, alguns critérios apresentados se fazem necessários para realizar esta comparação:

- **Identificação de Riscos:** A metodologia tradicional prioriza o conhecimento de todos os requisitos pertinentes ao projeto e a partir deles é estruturado o planejamento que define o desenrolar da execução das etapas do processo. No tocante a identificação do risco propriamente dito, há o reconhecimento de riscos, já no desenvolvimento ágil o foco mais importante é o desenvolvimento e entregas rápidas, com isso, riscos que possam surgir no decorrer do projeto são tratados em reuniões rápidas, chamadas de *stand up meetings*.
- **Análise de Riscos e Ameaças:** No desenvolvimento tradicional o controle do projeto se baseia nos requisitos já estabelecidos, o risco é visto como algo que pode ameaçar o projeto ou dar errado, dessa forma, uma providência é tomada para reduzir o risco identificado, porém, se surgir no decorrer do desenvolvimento um requisito ou um risco inesperado, o todo tem que ser revisto. No desenvolvimento ágil a abordagem utilizada possibilita que novos requisitos sejam inseridos durante o ciclo de vida sem afetar o todo, uma vez que, dá suporte as mudanças que possam surgir. E se tratando do tratamento de riscos, a metodologia não enfoca a análise de riscos, esta ocorre de forma muito sucinta, pois almeja entregas rápidas de software, destinando menos tempo a análises e planejamento.
- **Estimativa de Segurança:** De acordo com o que foi exposto na seção 3.2.2, a estimativa é a redução da incerteza estabelecendo uma ação de segurança para cada risco identificado. Nesse sentido, identifica-se que apenas a metodologia tradicional realiza esse tratamento, entretanto, de um modo geral, as medidas tomadas são com foco em risco de projeto, ou seja, as iniciativas tomadas estão associadas ao sucesso ou fracasso do projeto, desviando-se do foco do presente trabalho que é voltado a riscos de produto.
- **Integridade de Segurança:** Em vista ao exposto na seção 3.2.3, nenhuma das metodologias vistas preocupa-se com segurança no que diz respeito a arquitetar o sistema para que perigos sejam evitados, detectados, removidos ou até mesmo que inclua soluções de proteção minimizando os resultados de um acidente, caso aconteça.
- **Monitoramento:** Ambas as metodologias monitoram o que está sendo desenvolvido a partir de testes, possibilitando um maior feedback do software desenvolvido, e como observado na seção 3, este é um critério favorável no que diz respeito ao controle do software. Outro questionamento que pode ser levantado em relação ao enfoque ágil é quanto a sua utilização em sistemas críticos pelo fato de destinar pouca atenção a documentação. Será que com uma documentação menor é possível obter a certificação de software? Assim, sob a ótica de sistemas críticos voltados a área médica, há controvérsias quanto a sua utilização em processos ágeis, pois há uma necessidade considerável de documentação extensa para apresentação e comprovação junto a órgãos reguladores atestando detalhadamente a eficácia e qualidade do software crítico.

A tabela a seguir, apresenta o resumo baseado nos critérios abordados neste comparativo entre de metodologias:

Tabela 4. Resumo Comparativo com base em Critérios de Risco

Critérios de comparação	Desenvolvimento Tradicional		Desenvolvimento Ágil	
Identificação de riscos	✓	A identificação ocorre de forma mais criteriosa por meio de planejamento antes que o projeto inicie.	✘	O risco é visto como um requisito normal. Tenta minimizar os riscos com desenvolvimento em releases curtos, ou seja, é identificado durante o projeto.
Análise de riscos e ameaças	✓	Reconhecimento explícito de risco. É visto como algo que pode ameaçar o projeto ou dar errado.	✘	Direcionamento menor em análises. Risco é considerado um elemento cotidiano.
Estimativa de segurança	✓	Providências são tomadas para reduzir o risco identificado.	✘	Não há medidas de segurança.
Integridade de segurança	✘	A segurança não é integrada no ciclo de vida do projeto.	✘	A segurança não é integrada no ciclo de vida do projeto.
Monitoramento	✓	A verificação é feita por meio de testes. Facilita o controle dos riscos. A documentação é priorizada a cada etapa finalizada durante o projeto.	✓	O projeto passa por testes frequentes possibilitando o reconhecimento de erros previamente e tomando as devidas resoluções, evitando custos adicionais. Isso facilita o controle dos riscos. Com relação a documentação, a preocupação é menor, uma vez que, o foco está no desenvolvimento e na entrega do produto de maior valor para o cliente no menor tempo possível.

Uma observação a ser feita no presente trabalho, é com relação ao critério exposto na tabela anterior que pode ser considerado o mais importante e relevante no tocante a sistemas médicos: Integridade de Segurança, pois além de integrar a segurança em todo o ciclo de vida do projeto, o sistema deve ser estruturado de forma que seja capaz de prevenir, identificar e remover ameaças. Assim, não houve nenhuma preocupação com integridade de segurança nas metodologias abordadas neste trabalho, evidenciando a necessidade de rever ou adequar este quesito em um processo de desenvolvimento de software.

Diante da conjuntura atual em que tecnologias estão sendo inseridas para facilitar as atividades do dia a dia e por outro lado à atenção está voltada aos cuidados com a saúde e a qualidade de vida, esta constatação é importante, visto que proporciona uma contribuição considerável a futuras pesquisas, pois evidencia a necessidade de que sistemas médicos sejam arquitetados para realizar operações automáticas e imediatas, incluindo: prevenção, identificação e remoção, que limitem os prejuízos caso um dano venha a ocorrer. Integrar tais características em um processo de desenvolvimento de software é um passo inicial e primordial para assegurar que pacientes submetidos a tratamentos por meio de sistemas médicos, estarão seguros.

5 CONCLUSÃO E TRABALHOS FUTUROS

Como observado na análise realizada, o trabalho visava identificar como os processos de desenvolvimento de software existentes se aplicam no desenvolvimento sistemas na área da saúde, uma vez que, requerem enfoques mais estruturados e direcionados para lidar com esta proposta, onde o quesito segurança deve ser tratado durante todo o ciclo de vida do software. Dessa forma, foi possível perceber que os processos existentes não tratam em específico do requisito de segurança, este é considerado apenas como um requisito normal e sendo tratado dessa forma não é o suficiente para assegurar a qualidade de um sistema médico, nem obter certificação objetivando a adequação de normas destinadas a este tipo de dispositivo. Mas em vista aos dados observados, os processos ágeis são mais eficazes no que se refere ao *time-to-market*, entretanto apresenta-se inferior no comparativo entre metodologias, enquanto o rigor encontrado na metodologia tradicional se apresenta mais adequado para desenvolver um sistema médico. Com isso, conclui-se, que os processos de um modo geral, atendem parcialmente ao tratamento de riscos, sendo necessárias algumas reformulações para atender de forma eficiente a perspectiva esperada para esta linha de desenvolvimento. Assim, como sugestão de trabalhos futuros, propõe-se abordar dois processos, um tradicional e um ágil, fazendo um misto das duas metodologias, moldando-a a fim de cobrir o tratamento de riscos e de segurança, integrando a característica de *time-to-market* da metodologia ágil, buscando a disponibilização mais rápida do sistema e adequando-o para aprovação por parte dos órgãos reguladores de software médicos.

ABSTRACT

The use of software systems in healthcare have been increasingly common in order to improve the delivery of service to patients. Systems like these are considered as critical by the fact that the consequences of a failure can cause considerable harms, from environment damaged to death of people. Therefore, developing a software with these features isn't too easy, because it's necessary to use a structured process that meets all characteristics inherent to this purpose. In this sense, this paper identifies the key available methodologies, shows the priorities to be observed in critical systems and, finally, conduct a comparision between methodologies focusing on critical systems, identifying how each one is applicable on this kind of project.

KEYWORDS: Processes of software development. Traditional Methodology. Agile Methodology. Critical Systems.

REFERÊNCIAS

AFONSO, Paulo; NUNES, Manuel; PAISANA, Antônio; BRAGA, Ana. **The influence of time-to-market and target costing in the new product development success.** *International Journal of Production Economics*, 2008. v. 115, n. 2, October 2008, pp. 559-568, ISSN 0925-5273.

BECK, Kent; BEEDLE, Mike; VAN, Arie Bennekum; COCKBURN, Alistair; CUNNINGHAM, Ward; FOWLER, Martin; GRENNING, James; HIGHSMITH, Jim; HUNT, Andrew; JEFFRIES, Ron; KERN, Jon; MARICK, Brian; MARTIN, Robert C.; MELLOR, Steve; SCHWABER, Ken; SUTHERLAND, Jeff; THOMAS, Dave. **Manifesto for Agile Software Development.** (2001). Disponível em: <http://www.agilemanifesto.org/> Acesso em: 07 de Agosto de 2013.

BOEHM, Barry; TURNER, Richard. **Using Risk to Balanc Agile and Plan- Driven Methods.** *Computer*, 2003, v. 36, n. 6 , June 2003, pp 57-66, ISSN 0018-9162.

COOPER, Alecia. **Initiatives to Improve Patient Safety**. *Perioperative Nursing Clinics*, 2008, v. 3, n. 4, December 2008, pp 287-295.

COYLE, Sharon; CONBOY, Kieran. **A Case Study of Risk Management in Agile Systems Development.** (2009). *ECIS - 17th European Conference on Information Systems*. Disponível em: <http://aisel.aisnet.org/ecis2009/3/> Acesso em: 21/10/2013.

DYBA, Tore; DINGSOYR, Torgeir. **Empirical Studies of Agile Software Development: a Systematic Review.** *Information and Software Technology*, 2008, v. 50, n. 9-10, pp 833-859, ISSN 0950-5849.

EDUARDO, Carlos de Barros Paes; MASSAKI, Celso Hirata. **RUP extension for the development of secure systems.** *Emerald - International Journal of Web Information Systems*, 2007, v. 3 n. 4, pp. 293-314.

GARCIA, Edes da Costa Filho; PENTEADO, Rosângela; COUTINHO, Júnia Anacleto Silva; VACCARE, Rosana Teresinha Braga. **Padrões e Métodos Ágeis: agilidade no processo de desenvolvimento de software.** (2005). Disponível em: <http://sugarloafplop2005.icmc.usp.br/papers/9673.pdf> Acessado em: 07/08/2013.

GOVE, R. A.; HEINZMAN, J. L. **Safety criteria and model for mission-critical embedded software systems.** *Computer Assurance, 1991. COMPASS '91, Systems Integrity, Software Safety and Process Security. Proceedings of the Sixth Annual Conference on*, 1991, pp 69-73, Jun 1991.

ISO (2007). **ISO 14971. Medical devices - Application of risk management to medical devices**, 2nd Edition.

IWOHARA, Steven K.; LIU, Dar-Biau. **A Verification Tool to Measure Software in Critical Systems**. *Reliability and Maintainability Symposium, 1995. Proceedings., Annual*, Jan 1995, pp 315-320, ISSN 0149-144X.

LINDHOLM, Christin; NOTANDER, Jesper Pedersen; HÖST, Martin. **Software Risk Analysis in Medical Device Development**. *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, 2011, Sept 2011. p. 362-365.

MADAN, Sanjay. **Techniques to facilitate development of safety critical software systems.** *Electrical and Computer Engineering, 1997. Engineering Innovation: Voyage of Discovery. IEEE 1997 Canadian Conference on*, 1997, v. 1, May 1997, pp 249-252, ISSN 0840-7789.

MAINART, Domingos de A.; SANTOS, Ciro M. **Desenvolvimento de Software: Processos Ágeis ou Tradicionais? Uma visão crítica.** (2010). Disponível em: <http://www.enacomp.com.br/2010/cd/artigos/completos/enacomp2010_4.pdf> Acesso em: 05/08/2013.

MCCAFFERY, F.; BURTON, J.; RICHARDSON, I. **Improving Software Risk Management in a Medical Device Company**. *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, 2009, May 2009, pp 152-162.

MISRA, Subhas; KUMAR, Vinod and Uma Kumar; FANTAZY, Kamel; AKHTER, Mahmud. **Agile software development practices: evolution, principles, and criticisms.** *Emerald - International Journal of Quality & Reliability Management*, 2012, v. 29 n. 9, pp. 972-980.

PHANI, S. Kumar; SEETHA, P. Ramaiah; KHANAA, V. **A Methodology for Building Safer Software Based Critical Computing Systems.** *Advance Computing Conference (IACC), 2010 IEEE 2nd International*, Feb 2010, pp 422-429.

PROJECT MANAGEMENT INSTITUTE (PMI). **Um guia do conhecimento em gerenciamento de projetos (PMBOK Guide).** 4ª ed. 2008. ISBN: 978-1-933890-70-8.

PRESSMAN, R. S. **Engenharia de Software**. 6ª ed. McGraw-Hill, 2006.

RENATO, Alexandre Rodrigues de Souza. **Desafios e Oportunidades no Desenvolvimento de Produtos Eletrônicos com Software Embarcado na Área Médica.** (2011). Disponível em: <http://ubiq.inf.ufpel.edu.br/arrsouza/lib/exe/fetch.php?media=artigo_topicos.pdf> Acesso em: 15/01/2014.

SANTOS, Michel Soares. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.** (2004). Disponível em: < https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCoQFjAA&url=http%3A%2F%2Fxps-project.googlecode.com%2Fsvn%2F!svn%2Fbc%2F6%2Ftrunk%2Foutros%2FMet._Ageis.pdf&ei=CG8XU7zIF4jKkAfD3oCgDg&usg=AFQjCNFQAzq2EGASz8JiIenpBOA488UgmA&bvm=bv.62286460,d.eW0&cad=rja> Acesso em: 15/01/2014.

SCOTT, G. Gazelle; SANTOS, Isa C.T.; ROCHA, Luís A.; TAVARES, João Manuel R.S. **Desenvolvimento de Dispositivos Médicos: Vantagens de uma Metodologia Dedicada.** (2011). In: CIBEM. Disponível em: <http://paginas.fe.up.pt/~tavares/downloads/publications/artigos/CIBEM_2011_IsaCTSantos.pdf> Acesso em: 11/11/2013.

SIPONENA, Mikko; BASKERVILLEB, Richard; KUIVALAINENA, Tapio. **Integrating Security into Agile Development Methods**. *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, Jan 2005, pp 185a, ISSN 1530-1605.

SOMMERVILLE, I. **Software Engineering.** (2004). Addison-Wesley, Reading, MA.

SOMMERVILLE, I. **Engenharia de Software.** (2007). Addison-Wesley, Reading, 8ª Edição, MA.

WARREN, C. Axelrod. **Applying Lessons from Safety-Critical Systems to Security-Critical Software.** *Systems, Applications and Technology Conference (LISAT), 2011 IEEE Long Island*, May 2011, pp 1-6.

WUNRAM, Jurgen. **A strategy for identification and development of safety critical software embedded in complex space systems.** *Acta Astronautica,* 1993, v. 29, n. 3, March 1993.

YLIMANNELA, Ville. **A Model for Risk Management In Agile Software Development.** (2012). Disponível em: <http://www.cloudsw.org/under-review/a6f468c9-4857-4206-96ee-f67df0583d41/file_initial_version> Acesso em: 16/09/2013.