Universidade Estadual de Londrina

BRUNO CARAZATO DE OLIVEIRA

GAIA PROTÓTIPO: UM MODELO DE PROTOTIPAÇÃO PARA PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

LONDRINA–PR

2018

BRUNO CARAZATO DE OLIVEIRA

GAIA PROTÓTIPO: UM MODELO DE PROTOTIPAÇÃO PARA PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rodolfo Miranda de Barros

LONDRINA–PR

2018

Bruno Carazato de Oliveira

Gaia Protótipo: Um Modelo de Prototipação para Processos de Desenvolvimento de Software/ Bruno Carazato de Oliveira. – Londrina–PR, 2018-

56 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Rodolfo Miranda de Barros

– Universidade Estadual de Londrina, 2018.

1. Processo de Desenvolvimento de Software. 2. Prototipação. I. Prof. Dr. Rodolfo Miranda de Barros. II. Universidade Estadual de Londrina. III. Gaia Protótipo: Um Modelo de Prototipação para Processos de Desenvolvimento de Software

CDU 02:141:005.7

BRUNO CARAZATO DE OLIVEIRA

GAIA PROTÓTIPO: UM MODELO DE PROTOTIPAÇÃO PARA PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Prof. Dr. Rodolfo Miranda de Barros
Universidade Estadual de Londrina
Orientador

Prof. Dr. Adilson Luiz Bonifácio
Universidade Estadual de Londrina

Prof. Dr. Vitor Valério de S. Campos
Universidade Estadual de Londrina

Londrina–PR, 22 de janeiro de 2018

Este trabalho é dedicado à minha família,
em especial aos meus pais, por todo apoio, confiança e por nunca medir esforços ao me ajudar em toda minha formação.

AGRADECIMENTOS

Agradeço primeiramente a Deus por todas as oportunidas que constantemente coloca na minha vida e pela proteção por onde quer que eu vá. Agradeço imensamente aos meus pais, Fábio Wagner de Oliveira e Rosângela Ap. Carazato de Oliveira, por sempre investirem na minha formação pessoal e acadêmica e não deixarem faltar amor e carinho na minha vida. Também agradeço ao meu irmão Murilo pelo companheirismo e pelos momentos agradáveis que temos juntos.

A minha namorada Maria Paula por todo apoio em todas as situações, dentro e fora da universidade, tendo sempre uma paciência imensa comigo e por esse amor fraternal que nossa relação ao longo dos anos se tornou.

A todos os amigos que caminham comigo, que também foram de suma importância

Ao meu orientador Prof. Rodolfo Miranda de Barros, por confiar no meu trabalho, me incentivar e por me guiar nessa reta final.

Por fim, agradeço a todos os professores pelo conhecimento e experiências compartilhados, a Universidade Estadual de Londrina e o Departamento de Computação por me proporcionarem durante esses 4 anos um ótimo ambiente de estudo.

"The things that we love tell us what we are."
(Thomas Aquinas)

OLIVEIRA, B. C.. **Gaia Protótipo: Um Modelo de Prototipação para Processos de Desenvolvimento de Software**. 56 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina–PR, 2018.

RESUMO

Possuir um processo de desenvolvimento de *software* adequado é imprescindível para que a máxima qualidade do *software* seja o objetivo, uma vez que a indústria desta área esta em constante crescimento e aprimoramento, tornando o mercado mais competitivo. Sendo assim, o presente trabalho propõe o desenvolvimento de um modelo para os PDS com foco na prototipação evolutiva e também descartável, visto que a prototipação é uma das práticas que minimiza vários riscos no processo de desenvolvimento de um *software*. As melhores práticas de metodologias já existentes também foram aplicadas de forma empirica para a construção do modelo proposto, tendo em vista os variados tipos de equipes e as diferentes culturas organizacionais.

Palavras-chave: Prototipação. Processo de Desenvolvimento de *Software*. Modelo.

OLIVEIRA, B. C.. **Gaia Protótipo: A Prototyping Model for Software Development Process**. 56 p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina–PR, 2018.

ABSTRACT

An appropriate software development process is essential when the quality of software is the primary goal, since in this area, the industry is constantly growing and improving, making the market more competitive. Thus, this work propose the development of a new model of software development process, using prototyping as base, once prototyping is a technique that minimizes a lot of risks on software development processes, using also the best practices observed in existing methodologies, taking into account the different types of teams and organizational cultures.

Keywords: Prototyping. Software Development Process. Model.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ISO	*International Organization for Standardization*
PDS	Processo de Desenvolvimento de *Software*
UP	*Unified Process*
XP	*Extreme Programming*
NPI	Núcleo de Projetos em Informática
DNIT	Departamento Nacional de Infraestutura de Transportes
UML	*Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Ano após ano cresce a importância do *software* na sociedade. Desde os variados setores de negócio até mesmo os aspectos mais cotidianos da nossa vida, como: atividades pessoais ou de trabalho, infra-estruturas civis e industriais, política, educação e entretenimento são influênciados por algum tipo de *software*. Como consequência, o desenvolvimento de *software* tornou-se uma atívidade a ser cuidadosamente estudada e melhorada [7].

Para se obter um produto de *software* de qualidade, é imprescindível estabelecer um Processo de Desenvolvimento de *Software* (PDS) adequado. Um PDS consiste em etapas parcialmente ordenadas com um conjunto de atividades a serem desenvolvidas e de forma genérica, um modelo para o PDS deve compreender cinco atividades, podendo ser descritas como [3]:

- **Comunicação**: compreender os objetivos das partes interessadas e fazer o levantamento correto das necessidades e requisitos;
- **Planejamento**: definir as tarefas técnicas a serem conduzidas, os possíveis riscos, os recursos necessários, o que deve ser produzido ao fim do processo e um cronograma de trabalho;
- **Modelagem**: esboçar de forma refinada ou não, a fim de visualizar uma idéia mais concreta de como cada parte se encaixará;
- **Construção**: gerar código e executar testes para que possíveis erros sejam solucionados;
- **Emprego**: entrega do *software* ao cliente que fornece um *feedback*.

Dado a importância e a necessidade do *software* na sociedade, fica evidente que qualquer melhoria em seu processo de produção pode trazer um aumento do lucro para as empresas que desenvolvem *software*. Como resultado de um PDS adequado, tem-se um produto de melhor qualidade. Sendo assim, constata-se o interesse na melhoria dos processos de desenvolvimento de *software*.

Com base nas experiências dos próprios profissionais, novas idéias surgem, fazendo com que o mercado de *software* se tornasse mais competitivo e exigente. A partir de então, as cinco atividades citadas anteriormente têm sido adaptadas e otimizadas para cada formato de empresa e equipe, e com isso, novos mecanismos e modelos são desenvolvidos a fim de obter um *software* de melhor qualidade, minimizar custo, tempo e retrabalho.

A prototipação nesse contexto é relevante justamente por ser uma técnica candidata à minimizar o retrabalho, e consequententemente, diminuir o custo e reduzir atrasos na entrega do produto final, pois mudanças e/ou erros durante o processo podem acontecer.

Existem dois tipos diferentes de protótipos no desenvolvimento de *software*: os protótipos de levantamento (*Throw-away*), cuja finalidade é ajudar a entender determinados requisitos de difícil compreensão pelo analista; protótipos evolutivos, que oferecem uma projeção funcional viável do sistema para o cliente e muitas vezes se tornam parte do sistema final [8].

O objetivo do presente trabalho é desenvolver um modelo baseado nas práticas de prototipação. Sabendo que são técnicas candidatas a minimizar riscos no PDS, constata-se que se bem exploradas, as práticas de prototipação podem trazer benefícios relevantes, minimizando problemas no decorrer do PDS. Fez-se uso também das melhores práticas observadas das metodologias denominadas Ágeis e Evolucionárias, promovendo a melhoria contínua no PDS.

A motivação para o presente trabalho se deu ao constatar a importância do uso de um modelo adequado para o PDS. Assim, iniciou-se um estudo na literatua a fim de justificar e embasar os benefícios para o PDS ao utilizar modelo que explora as práticas de prototipação e agrega as boas práticas já existentes. Dessa forma, o modelo foi aplicado no desenvolvimento de uma aplicação móvel.

A organização segue da seguinte maneira: No Capítulo 2, são apresentados conceitos e definições usados neste trabalho e o levantamento de trabalhos correlatos. O Capítulo 3 aborda o modelo proposto, a aplicação do modelo em um estudo de caso, uma análise comparativa com os trabalhos correlatos e algumas considerações sobre o capítulo. No Capítulo 4, encontram-se as conclusões finais sobre o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Segundo a ISO 12207 [9], o ciclo de vida de um produto de *software* é modelado em estágios. Os modelos podem ser utilizados para representar todo o ciclo de vida, desde o conceito, até a cessão ou parte dele. O foco deste trabalho será no PDS, que é o estágio responsável por conceber o produto no ciclo de vida do *software*.

Em [10], o autor afirma que o sucesso das organizações que atuam na indústria de *software* se deve pela qualidade do seu produto, que está diretamente ligado à qualidade no processo de desenvolvimento do mesmo. Além disso, como desenvolver está relacionado à criatividade e habilidade dos desenvolvedores de *software*, a força de trabalho é o principal ativo organizacional, o que se confirma e é ressaltado em [11], quando o autor apresenta a influência direta da qualidade do trabalho em equipe na qualidade do *software* desenvolvido.

2.1 Processo de Desenvolvimento de *Software*

Um Processo ou Metodologia de Desenvolvimento de *Software* consiste em um conjunto de atividades e resultados associados que auxiliam na produção de uma aplicação.

Segundo [1], muitas organizações no cenário brasileiro atual, mesmo cientes da importância de desenvolver *software* de qualidade, ainda o fazem sem utilizar nenhum processo e ainda segundo o autor, isso se deve pois, em particular, as organizações de pequeno e médio porte não possuem recursos suficiente para fazer uso das metodologias tradicionais, ditas “pesadas”. Outras empresas, por desconhecerem novas metodologias e há também as que conhecem porém têm receio da mudança, optando por permanecer com o mesmo modelo tradicional.

O resultado da falta de um processo adequado na produção de *software* é a baixa qualidade do produto final, a dificuldade de cumprir prazos e custos predefinidos e possivelmente inviabilizar a futura evolução do mesmo.

2.1.1 Qualidade de Software

Segundo [12], a palavra *qualidade* na Engenharia de *Software* soa como ambígua, visto que, dependendo do ponto de vista, pode estar relacionada a diferentes fatores, e, as seguintes definições são consistentes e adotadas por muitos profissionais responsáveis pela qualidade:

- Da perspectiva profissional, em [13], o autor define qualidade como “conformidade com os requisitos”, o que implica no elevado grau de compreenção dos requisitos

de *software*, e então, no processo de desenvolvimento do *software*, medições são tomadas regularmente a fim de determinar a conformidade com esses requisitos. Em [14], por sua vez, o autor define *qualidade* como "adequado ao uso". Essa segunda definição leva em consideração não só os requisitos levantados, mas também as expectativas do consumidor, tendo em vista que os consumidores podem usar os produtos para diferentes finalidades, ou seja, "adequado ao uso" pode ser classificado por determinados parâmetros.

- Da perspectiva do consumidor por sua vez, a qualidade é o valor agregado ao comprar determinado produto, com base em variáveis, como: preço, desempenho, confiabilidade e satisfação.

2.2 Metodologias Tradicionais

As metodologias tradicionais, também são chamadas de "pesadas" [1], pois têm como principal característica a documentação total do *software* antes de sua implementação e não são sucetíveis a mudanças no decorrer da produção, ou seja, será desenvolvido de início ao fim o que foi planejado [3]. Esta é a principal limitação dos modelos tradicionais, pois o foco desses modelos é a previsibilidade dos requisitos do sistema, que por um lado torna mais fácil a gerência dos projetos, visto que são completamente planejados, mas por outro, torna a especificação de requisitos uma etapa fundamental, onde todas as necessidades do cliente devem ser muito bem definidas e documentadas.

Dessa maneira, segundo [15], os processos de análise e projeto tornam-se bastante demorados e de difícil manutenção caso ocorra alguma mudança nas especificações.

2.2.1 Cascata

O primeiro processo tradicional publicado foi o chamado *Cascata* [3]. Esse modelo é considerado a principal metodologia tradicional, sendo muito utilizado em sua forma original até hoje [1].

O modelo Cascata descreve um método linear e sequencial de fases, sendo que quando uma determinata fase é completada, segue-se para a próxima sem a opção de voltar, pular ou refazer a etapa atual.

Uma vantagem do modelo é a possibilidade que ele proporciona de se realizar um controle departamental e gerencial, visto que cada fase é muito bem definida. Por outro lado, ele não permite muita flexibilidade ou revisão. Se algo não foi bem pensado no estágio conceitual, será muito dificil fazer alterações no decorrer do desenvolvimento.

De maneira geral, são estabelecidas as etapas descritas na Figura 1:

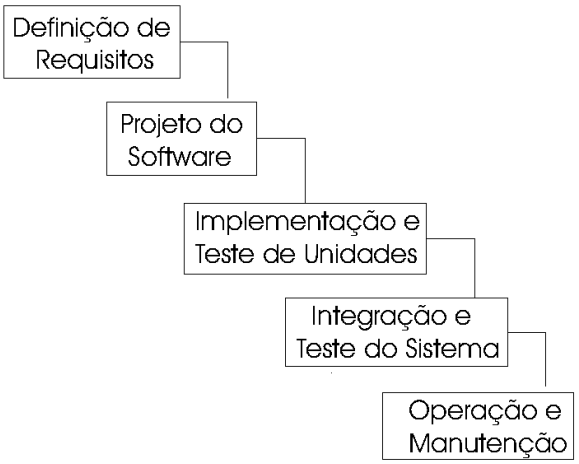

Figura 1 – Modelo Cascata. (Fonte: [1])

Em [3], o autor elenca alguns problemas que o modelo enfrenta:

- Os projetos reais raramente seguem o fluxo seqüencial que o modelo propõe. Geralmente alguma iteração acontece, trazendo problemas na aplicação do paradigma;
- É difícil para o cliente declarar todas suas necessidades explicitamente. O ciclo de vida dos modelos clássicos exige isso e as incertezas naturais não são acomodadas com facilidade;
- O cliente precisa ter paciência visto que uma versão funcional do *software* não estará disponível tão cedo. Ponto importante, que se não detectado cedo, pode ser desastroso.

2.3 Metodologias Evolucionárias

Ao longo do tempo, o *software* evolui, com isso, novas necessidades de negócio e produto surgem frequentemente. Assim, para os modelo de desenvolvimento de linha reta (sem iterações), as limitações citadas na sessão anterior se agravam, e então, são criados os modelos evolucionários.

Os modelos evolucionários começam a introduzir a idéia de como lidar com mudanças, pois os requisitos do negócio e do produto podem mudar freqüentemente à medida que segue o desenvolvimento da aplicação. Tem-se então a concepção da possibilidade densenvolver produtos que evoluam ao longo do tempo.

2.3.1 O Processo Unificado

O UP, Processo Unificado (*Unified Process*) é um modelo de estrutura genérica de processos que pode ser personalizado, adicionando ou eliminando atividades segundo

as necessidades particulares do cliente, epigrafadas nos recursos orçados para um projeto [16].

O Processo Unificado faz uso extensivo da UML, a fim de especificar, modelar e documentar artefatos. É guiado por casos de uso com foco em riscos e centrado na arquitetura, visando a construção de sistemas orientados a objetos.

Caracteriza-se também por ser um processo iterativo e adaptativo de desenvolvimento de software e suas fases bem definidas trazem a consistência na organização e condução de um projeto.

O Processo Unificado organiza suas iterações em quatro fases principais:

- **Concepção:** Definir o escopo do projeto;
- **Elaboração:** Definir os requisitos e a arquitetura;
- **Construção:** Desenvolver o sistema;
- **Transição:** Implantar o sistema.

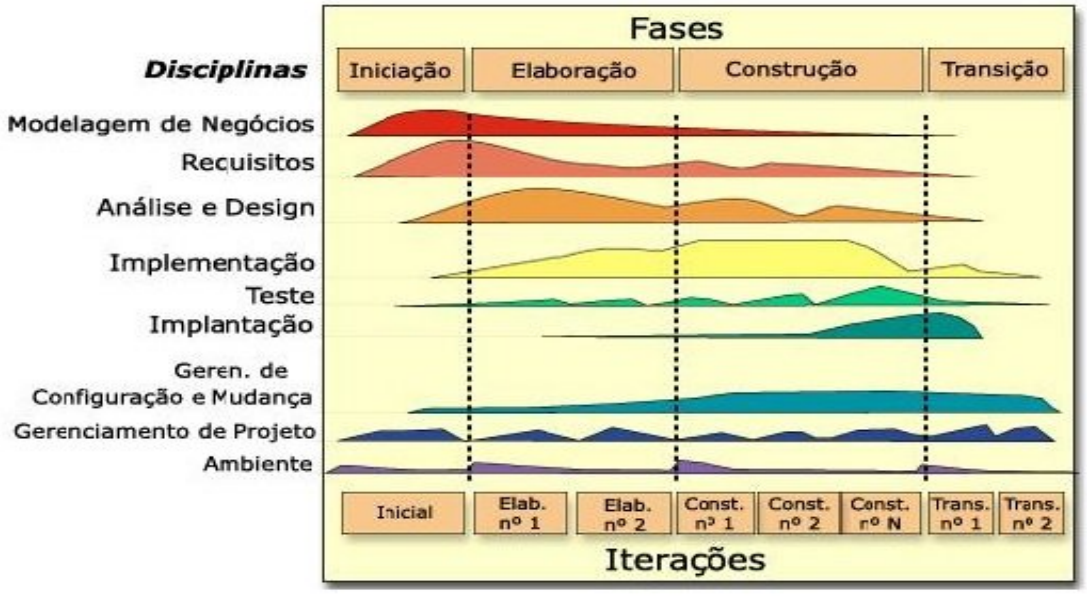

Figura 2 – Esboço das fases de um Processo Unificado. (Fonte: [2])

Ao ser constatado que não existe um modelo único suficiente para cobrir todos os aspectos do sistema, o UP suporta múltiplas visões e modelos arquiteturais.

2.3.2 Prototipação

Segundo Pressman [3], a prototipação costuma ser a melhor escolha de abordagem quando:

- O cliente define uma série de objetivos gerais para o *software*, mas não identifica, de forma detalhada, os requisitos para funções e recursos;
- O desenvolvedor encontra-se inseguro quanto à eficácia de um algoritmo ou quanto à adaptabilidade de um sistema operacional;
- Quando há dúvida quanto à forma em que deve ocorrer a interação homem/máquina.

Embora a prototipação possa ser utilizado como um modelo de processo isolado, é mais comumente utilizada como uma opção passível de ser implementada no contexto de qualquer metodologia.

Em [17], o autor descreve um estudo de caso feito com duas empresas de desenvolvimento de *software*. Essas empresas foram escolhidas seguindo os seguintes critérios:

- Tempo de atividade de no mínimo quatro anos;
- Mais de vinte profissionais com vínculo direto com a empresa;
- Mais de 50 clientes em carteira.

Uma dessas empresas, chamada Educom, utiliza a prototipação como uma de suas práticas para desenvolvimento de seus produtos, para entender melhor as necessidades de seus clientes, pois ao idealizar os protótipos (versão beta), são levados aos clientes para a verificação e validação de sua aceitação.

Em geral, os protótipos podem ser construídos como "descartáveis" ou então evolucionários, no sentido de que evoluem lentamente até se transformarem no produto final.

2.3.3 Espiral

Apesar da possibilidade de ser classificado como Evolutivo, em [6] o autor diz que o modelo Espiral nasce beaseado na experiência obtida com vários refinamentos do modelo Cascata, porém com uma grande diferença, este cria uma abordagem orientada a riscos para o processo de desenvolvimento de *software*, já o Cascata é primariamente orientado a documentação ou a codificação.

A principal característica do modelo Espiral é o fato de ser um modelo incremental, tornando formal o conceito de iterações orientadas a risco e deixando claro a necessidade de avaliação dos riscos a cada iteração.

A Figura 3, ilustra algumas etapas que constituem o modelo:

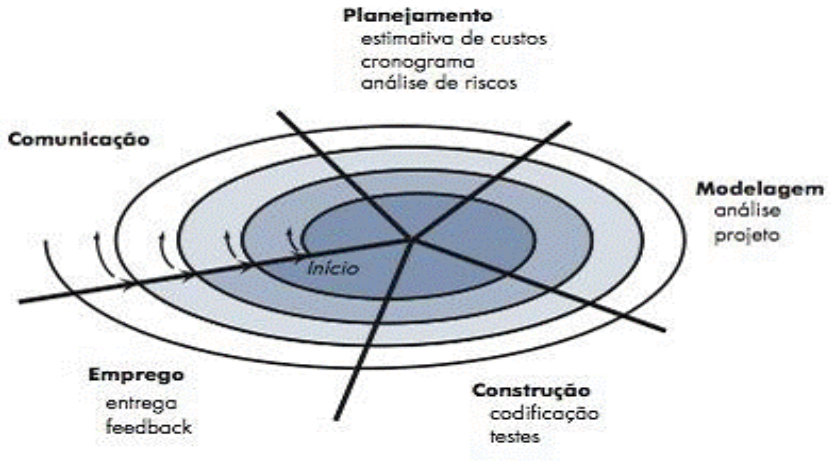

Figura 3 – Modelo espiral de desenvolvimento de *software*. (Fonte: [3])

Portanto, pode-se dizer que a metodologia é de bom uso para aqueles que defendem a utilização de planejamento extensivo, processos codificados e rigorosos reusos para tornar o processo uma atividade preditiva e eficiente que vai gradualmente amadurecendo em direção ao objetivo, mesmo assim, o modelo ainda enfrenta algumas limitações herdadas dos modelos tradicionais. Visto isso, com o objetivo de solucionar ou minimizar tais limitações, revolucionando a maneira de desenvolver *software*, surgem as metodologias Ágeis.

2.4 Metodologias Ágeis

Em 2001, acontece o Manifesto Ágil, uma declaração de valores e princípios essenciais para o desenvolvimento de *software*, onde se reuniram dezessete especialistas em PDS e profissionais da área (alguns já praticavam certos conceitos que seriam estabelecidos nessa reunião) descontentes com as metodologias até então existentes, a fim de levantar os pontos de sucesso das metodologias que cada um utilizava e com base nas suas experiências profissionais criam o Manifesto para Desenvolvimento Ágil de *Software*, as ditas "Metodologias Ágeis", termo que se popularizou mais a diante.

Os conceitos chave deste grupo são:

- **Indivíduos e interações** ao invés de processos e ferramentas.
- ***Software* executável** ao invés de documentação.
- **Colaboração do cliente** ao invés de negociação de contratos.
- **Respostas rápidas a mudanças** ao invés de seguir planos.

Em [18], o autor conclui que as Metodologias Ágeis formam um grupo de métodos incrementais e iterativos mais eficazes, pois implementam mecanismos de *feedback*

e melhoria contínua nos incrementos e iterações, além de representar o escopo do projeto de uma forma mais clara. Por isso, estas metodologias têm sido usadas também no gerenciamento de projetos em geral, não somente no desenvolvimento de *software*.

É valido a ressalva de que as Metodologias Ágeis não rejeitam os processos e ferramentas, bem como documentações e negociações, porém os atribuem uma importância secundária. Em [19], o autor ressalta a transferência de conhecimento entre os membros da equipe visto que essas metodologias utilizam de uma constante comunicação e colaboração entre os mesmos, especialmente através da chamada *face-to-face* (comunicação mais informal e direta).

2.4.1 *Extreme Programming*

Segundo [20], o XP é a metodologia ágil mais amplamente utilizada, voltado para projetos cujos requisitos são vagos e podem mudar com frequência, desenvolvimento de sistemas orientados a objeto, equipes pequenas, preferencialmente até 12 (doze) desenvolvedores, e desenvolvimento incremental (ou iterativo), onde o sistema começa a ser implementado logo no início do projeto e vai ganhando novas funcionalidades ao longo do tempo.

A Tabela 1 descreve 13 (treze) práticas importantes que compõem o XP, que segundo [5], são:

Prática no XP	**Descrição**
1. Cliente Presente	O XP trabalha com a premissa de que o cliente deve conduzir o desenvolvimento a partir do *feedback* que recebe do sistema.
3. Jogo do Planejamento	No início de cada iteração ocorre o jogo do planejamento. Trata-se de uma reunião onde o cliente avalia as funcionalidades que serão implementadas.
3. *Stand Up Meeting*	A equipe se reune a cada manhã para avaliar o trabalho que foi executado no dia anterior e priorizar aquilo que será implementado no dia que se inicia .
4. Programação em Par	Os desenvolvedores implementam as funcionalidades em pares, ou seja, diante de cada computador, existem sempre dois desenvolvedores que trabalham juntos para produzir o mesmo código.
5. Desenvolvimento Guiado p/ Testes	Testes são escritos para cada funcionalidade antes de codificá-las. Fazendo isso, eles aprofundam o entendimento das necessidades do cliente.

6. Refactoring	O refactoring é o ato de alterar um código sem afetar a funcionalidade que ele implementa. O objetivo é tornar o *software* mais simples de ser mantido.
7. Código Coletivo	Os desenvolvedores têm acesso a todas as partes do código e podem alterar aquilo que julgarem importante sem pedir autorização de outra pessoa.
8. Código Padronizado	Para facilitar a manutenção no código por parte de toda equipe, padrões de codificação são definidos tornando o sistema mais homogêneo e permitir que qualquer membro da equipe tenha condições de dar manutenção no sistema.
9. Design Simples	Para que o cliente possa obter feedback logo, a equipe precisa ser ágil no desenvolvimento, o que leva a optar pela simplicidade do design.
10. Metáfora	Para facilitar a criação de um design simples, a equipe de desenvolvimento utiliza metáforas, já que elas têm o poder de transmitir ideias complexas de forma simples.
11. Ritmo Sustentável	Para garantir que a equipe tenha sempre o máximo de rendimento e produza *software* com melhor qualidade possível, o XP recomenda que os desenvolvedores trabalhem apenas oito horas por dia e evitem fazer horas-extras, visto que é essencial estar descansado a cada manhã, de modo a utilizar a mente na sua plenitude.
12. Integração Contínua	Prática utilizada com o objetivo de checar/testar a aplicação, sempre que uma nova funcionalidade é implementada, seja de forma manual ou automática, forma esta que se utiliza de ferramentas especializadas para tal.
13. Releases Curtos	O XP tem como objetivo gerar um fluxo contínuo de valor para o cliente. Sendo assim, ele trabalha com releases curtos, ou seja, a equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente.

Tabela 1 – As treze práticas importantes que compõem o XP. (Fonte: [5])

2.4.2 Scrum

O *Scrum*, foi desenvolvido por Jeff Sutherland e sua equipe no inicio de 1990. Segundo [18], o *Scrum* consiste em um time, eventos, artefatos e regras. As regras são essenciais para criar equipes, eventos e artefatos durante o projeto.

Neste modelo, são artefatos do processo:

- *Scrum Team*: a equipe;
- *Scrum Master*: o responsável por coordenar o processo e garantir que tudo ocorra corretamente, sempre promovento a adaptação da equipe;
- *Product*: o produto;
- *Product Backlog*: as respectivas funcionalidades do produto a serem desenvolvidas;
- *Product Owner*: o cliente, que possui participação importante no processo.

Figura 4 – Componentes do *Scrum*. (Fonte: [4])

A fase de desenvolvimento do *software* ocorre nas *Sprints*, que são vistas como o coração do processo. De forma sucinta, considera-se uma *Sprint* como um pequeno projeto, sendo parte do projeto original, cuja duração é definida inicialmente e dentro do prazo definido, espera-se atingir ou se aproximar o máximo possível de desenvolver todas as tarefas, que são priorizadas pelo PO (*Product Owner*).

Os objetivos de desenvolvimento de cada *Sprint* e o *Scrum Team* não devem ser alterados durante o curso da *Sprint*. Porém, o PO (*Product Owner*) e o *Scrum Team* podem redefinir o escopo do projeto conforme necessário. O PO também pode cancelar *Sprint* se ocorrer alguma alteração na direção da empresa, necessidades de mercado ou de tecnologia.

O trabalho conduzido dentro de uma *Sprint*, é adaptado ao problema em mãos, podendo ser frequentemente modificado pela equipe sem qualquer tipo de problema. Para que isso aconteça, reuniões rápidas ocorrem diariamente a fim de que os membros da equipe apresentem problemas, palpites e qualquer tipo de troca de informação que seja relevante ao processo [18].

Basicamente, ao final de cada *Sprint* são feitas reuniões mais detalhadas para apresentar o que foi produzido durante a *Sprint*, bem como um *feedback* do ocorrido, e, com as experiências obtidas, planejar uma nova *Sprint*.

2.5 Comparação entre Metodologia Ágeis e Tradicionais

Ambos os tipos de metodologias possuem pontos fortes e que podem ser combinados a fim de estabelecer um PDS que melhor se adeque ao estilo da equipe, produzindo assim bons resultados. A Tabela 2 faz uma comparação entre os principais pontos dos dois tipos de modelos.

	Tradicional	**Ágil**
Premissas Fundamentais	Sistemas são bem especificáveis e pode ser construídos através de planejamento meticuloso e extensivo.	*Software* de alta qualidade e adaptativo pode ser desenvolvido por times pequenos que usam princípios de melhoria continua em design e testes, sendo baseados em feedback rápidos e possíveis mudanças.
Controle	Centrado em processos.	Centrado em pessoas.
Estilo da gestão	Comando-e-controle.	Liderança-e-colaboração.
Gestão do conhecimento	Explícito.	Tácito.
Atribuição de papéis	Individuais – favorizam especialização.	Times autogeridos-encorajam troca de papéis.
Comunicação	Formal.	Informal.
Papel do cliente	Importante.	Crítico.
Ciclo do projeto	Guiado por tarefas ou atividades.	Guiado por funcionalidades do produto.
Modelo de desenvolvimento	Modelo de ciclo-de-vida (Cascata, espiral ou alguma variação destes).	Modelo de entrega evolutiva.

Estrutura ou forma organizacional desejada	Mecânica (burocrática, com alta formalização).	Orgânica (flexível e participativa, encorajando ação social cooperativa).
Tecnologia	Sem restrições.	Favorece tecnologia orientada a objetos.

Tabela 2 – Tabela comparativa entre características das metodologias tradicionais e ágeis. (Fonte: [6])

Ao analisar a Tabela 2, conclui-se que as Metodologias Tradicionais visam o processo, a documentação, a formalidade e a individualidade, enquanto as Metodologias Ágeis priorizam as pessoas, a adaptação à mudanças, uma comunicação mais informal e direta e a cooperação entre os membros da equipe.

Uma particularidade interessante é o papel do cliente em cada metodologia, sendo que nas Metodologias Ágeis, o cliente está em constante contato com desenvolvimento do produto, já nas metodologias Tradicionais, este contato só ocorre nos dois extremos do processo.

2.6 Riscos, problemas e possíveis soluções para o desenvolvimento de *software*

No decorrer de um PDS, são enfrentados diversos problemas, ainda que este tenha sido muito bem planejado, segundo [21], os principais são:

- Definição e cumprimento dos prazos;
- Custo do projeto;
- Produtividade;
- Qualidade;

A solução ou a minização destes problemas, pode garantir o sucesso ou fracasso de um projeto, por isso, a Engenharia de *Software* segue em constante estudo, afim de encontrar formas de aprimorar os PDS.

Em [6], o autor lista os dez maiores itens de risco para *software* e possíveis técnicas para enfrentá-los, são eles:

Item de risco	**Técnicas de gerenciamento de riscos**
1. Perda de pessoas	Elencar talentos que combinem com o trabalho; treinamento cruzado; pré-seleção de pessoas-chave.

2. Prazos e orçamentos não-realísticos	Estimativa detalhada dos prazos e custos por várias fontes, desenvolvimento incremental; reuso de *software*; enxugamento de requisitos.
3. Desenvolvimento das funções erradas de *software*	Análise organizacional; análise da missão; formulação de conceitos de operação; questionário com usuários; prototipapação; antecipação de manuais do usuário.
4. Desenvolvimento de interface de usuário errada	Análise das tarefas; prototipação; cenários; caracterização do usuário (funcionalidade, estilo, carga de trabalho).
5. "Gold plating" (ir além do esforço útil)	Enxugamento de requisitos, prototipação, análise do custo-benefício, "deisgn to cost" (desenhar a custo ótimo).
6. Onda contínua de requisições de mudanças	Limiar para muita quantidade em mudanças; ocultação de informação; no desenvolvimento incremental, deferir mudanças nos últimos incrementos.
7. Perdas devido a componentes externos	Benchmarking; inspeções; checagem de referências; análise de compatibilidades.
8. Perdas devido a tarefas externas	Checagem de referências; auditorias "pre-award"; contratos "award-fee"; design competitivo ou prototipação; construção em equipe.
9. Perdas devido ao desempenho em tempo real	Simulação; benchmarking; modelagem; prototipação; instrumentação; configuração de cenários.
10. Problemas com capacidades computacionais	Análise técnica; análise do custo-benefício; prototipação; checagem de referências.

Tabela 3 – Tabela priorizada dos dez maiores itens de risco para *software* e técnicas para enfrentá-los. (Fonte: [6])

Nota-se que pela Tabela 3, a prototipação é uma opção para enfrentar diversos itens de risco, sendo assim, no presente trabalho foram estudadas técnicas e abordagens para desenvolver um modelo que possa ser implantado em diferentes processos de desenvolvimento de *software*, a fim de minimizar tais riscos enfrentados e otimizar o processo como um todo.

2.7 Trabalhos correlatos

É possível encontrar na literatura vários trabalhos, alguns que ressaltam a importância dos modelos para o PDS [3] e outros que descrevem metodologias próprias, como

em [22, 23, 24].

Em [22], o autor propõe o *Qualitas*, um modelo de processo de desenvolvimento de *software* orientado a modelos. A metodologia proposta foi estudada experimentalmente através da implementação de funcionalidades relacionadas ao Sistema de Triagem Neonatal do Hospital Universitário da Universidade Federal de Sergipe. Resumidamente, o *Qualitas* foi desenvolvido seguindo o modelo Espiral, citado na seção 2.3.3, sendo que cada volta na espiral representa uma fase do PDS e sua execução segue os conceitos evolucionário e incremental.

O autor em [23], descreve um modelo desenvolvido para o Núcleo de Projetos em Informática (NPI) da Universidade Federal de Santa Catarina para o desenvolvimento de sistemas que são comercializados, o modelo pode ser chamado de MPDS-NPI. O MPDS-NPI foi proposto com base em um modelo próprio existente, até então utilizado pelo NPI, sendo esse um dos pontos iniciais do autor. O objetivo é melhorar o processo utilizado pela empresa júnior, respeitando algumas de suas práticas e necessidades burocráticas. O autor conclui pontos importantes a serem melhorados após sua pesquisa na literatura. Apesar do modelo desenvolvido ser específico para o NPI da UFSC, é possível identificar algumas características herdadas dos modelos tradicionais, ainda que algumas fases ocorram em iterações.

Já em [24], o autor descreve a metodologia de desenvolvimento de *software* utilizado pelo Departamento Nacional de Infraestutura de Transportes (DNIT). A metodologia, chamada também de *Software Novo*, tem como característica principal a definição clara e total dos processos, seguindo a metodologia UP, citado na seção 2.3.1. Faz-se também o uso de processos iterativos e incrementais para novos desenvolvimentos e evoluções. Mesmo assim, o autor afirma que as abordagens ágeis são práticas ainda insipientes na organização e não são tratadas no trabalho.

Estes trabalhos descrevem modelos próprios desenvolvidos a fim de melhorar o PDS de cada um. Vale ressaltar que esses modelos foram desenvolvidos sempre levando em consideração as características dos seus respectivos projetos. Já o Gaia Protótipo, busca ter como característica a possibilidade de ser aplicado no PDS de qualquer tipo de projeto.

3 GAIA PROTÓTIPO: UM MODELO DE PROTOTIPAÇÃO PARA PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

O GAIA Protótipo tem como objetivo descrever um modelo capaz elicitar a melhoria contínua no PDS, utilizando como base as práticas de prototipação, a fim de minimizar os problemas e riscos enfrentados pelos PDS, citados na seção anterior, que resultam em gastos desnecessários, baixa produtividade, má qualidade do *software*, entre outros.

Na composição do modelo, fez-se o uso de algumas características de dois outros meodelos, Scrum (Ágil) e Espiral (Evolucionária), unindo empíricamente os pontos positivos de cada um para que o objetivo fosse atingido.

Vale ressaltar que, embora a prototipação possa ser utilizada como um modelo isolado, as práticas de prototipação foram utilizadas no presente trabalho.

O modelo foi desenvolvido de forma iterativa e incremental, assim como são suas próprias características. As melhores práticas e técnicas foram escolhidas e moldadas utilizando uma abordagem empírica para alcançar o resultado desejado. Espera-se que em futuros trabalhos, o modelo desenvolvido seja aplicado nos variados tipos de equipes e nas diferentes culturas organizacionais.

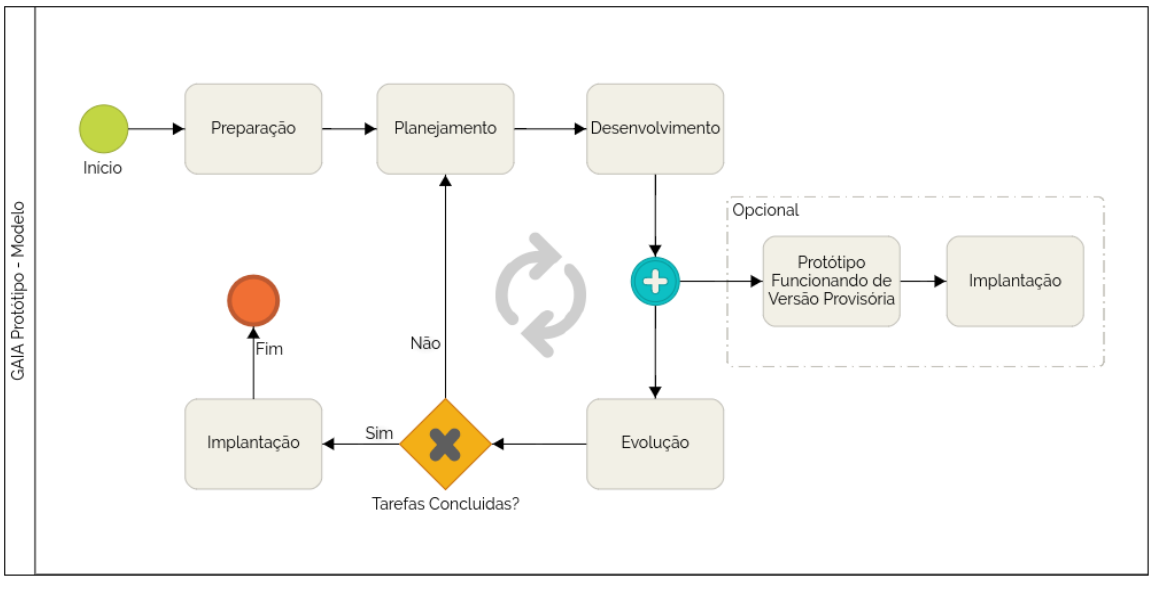

Figura 5 – GAIA Protótipo. (Fonte: Autor)

A Figura 5 ilustra todas as fases do modelo, sendo que, cada vez que o ciclo passa pela etapa de desenvolvimento, o protótipo evolui, assim, é possivel que seja enviado ao cliente um protótipo provisório funcional do sistema, enquanto paralelamente segue o

desenvolvimento do produto final.

O GAIA Protótipo é formado por poucos processos, de caráter evolutivo pois vão amadurecendo em direção ao objetivo, mas sempre visando a adequação às mudanças, tratando o papel do cliente como imprescindível no processo, assim como no Scrum. Também se faz necessária a etapa de *Ánalise de Riscos*, pois dessa forma, as decisões para as próximas etapas são tomadas ponderando as diferentes alternativas. Tais caracteríscas são encontradas também no modelo Espiral [6].

O objetivo do modelo proposto no presente trabalho, é o *software* de alta qualidade, podendo ser aplicado por qualquer tipo de times, porém, que façam uso de princípios de melhoria contínua e tratem o papel do cliente como imprescindível no processo, baseados em *feedbacks* rápidos e possíveis mudanças. Segundo [6], tais características também estão presentes nas metodologias Ágeis, como por exemplo, o Scrum.

3.1 Preparação

Na primeira fase do processo, o objetivo é elicitar os requisitos funcionais e não funcionais de tal forma que descrevam com fidelidade as necessidades do cliente, auxiliar o responsável pelo levantamento das funcionalidades a serem implementadas a entender melhor o que esta sendo pedido pelo cliente e também validar a plataforma e as tecnologias definidas pela equipe.

A Figura 6 retrata as seguintes etapas:

1. **Levantamento de Requisitos**: Nesta etapa, é estabelecida uma comunicação com o cliente a fim de fazer o levantamento de requisitos do *software*.

2. **Definição das Tecnologias**: Aqui, a equipe define a plataforma alvo e as tecnologias a serem utilizadas, tomando como base os recursos que o cliente possui para utilizar o sistema, bem como a capacitação técnica da equipe diante as tecnologias candidatas, podendo desenvolver um rápido Protótipo Teste de caráter descartável para testar na plataforma do cliente. Paralelamente, é desenvolvido a primeira versão do Protótipo de Interface de caráter evolutivo.

3. **Verificação e Validação**: Das etapas anteriores, tem-se o Protótipo de Interface desenvolvido com base no levantamento de requisitos realizado. O protótipo é levado para o cliente a fim de ser verificado e validado juntamente com o que se tem de requisitos até o presente momento, pois, empiricamente, percebe-se a maior facilidade e clareza do cliente para elicitar as necessidades que visa em seu produto ao se deparar com uma candidata interface. Caso o Protótipo de Interface e os requisitos não sejam aprovados, repete-se a etapa anterior aplicando as atualizações necessárias, caso contrário, começa então a próxima fase.

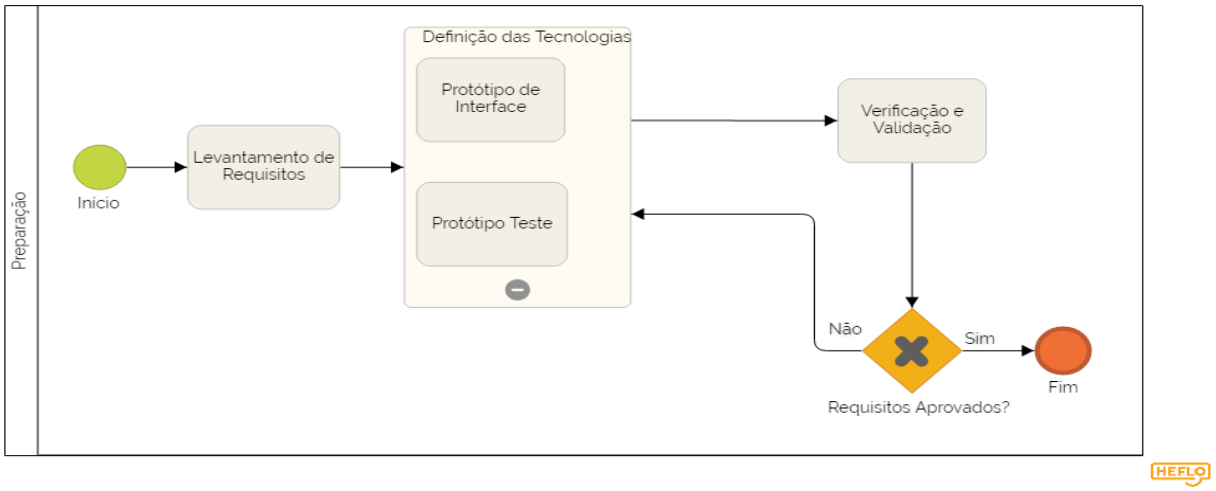

Figura 6 – Preparação. (Fonte: Autor)

3.2 Planejamento

A fase de Planejamento acontece a cada *Sprint*, dessa forma, o desempenho da fase de Desenvolvimento está diretamente relacionado com um bom planejamento. Do modelo Cascata, foi herdado uma de suas principais etapas para o GAIA Protótipo, a Análise de Riscos, que é a primeira etapa da fase de Planejamento. Em seguida, o responsável pelo processo divide as equipes (sempre aberto à sugestões do time), elencando um líder por equipe, estes por sua vez, juntamente com o responsável pelo processo distribuem as tarefas para as equipes e definem a *Sprint* de cada equipe.

A Figura 7 ilustra as sequintes etapas:

1. **Análise de Riscos**: Esta etapa merece dedicação diferenciada, visto que este é o momento em que são analisados os riscos das decisões de etapas anteriores.

2. **Divisão da Equipe**: Em seguida, a equipe é dividida de acordo com a necessidade do projeto, podendo ser também sub-dividida em várias equipes, porém, é importante que um membro de cada equipe seja o responsável por ela.

3. **Distribuição de Tarefas**: O responsável pelo processo, juntamente com os líderes de cada time, se reunem a fim de dividir as tarefas, levando em consideração o conhecimento que possuem sobre a capacidade dos membros de sua equipe.

4. **Definição das *Sprints***: Os mesmos responsáveis da etapa anterior também definem o tempo aproximado necessário para cada *Sprint* e os horários que as reuniões acontecerão. Neste trabalho, a chamada *Sprint* consistirá a fase de Desenvolvimento, bem como suas tarefas a serem desenvolvidas de forma ágil.

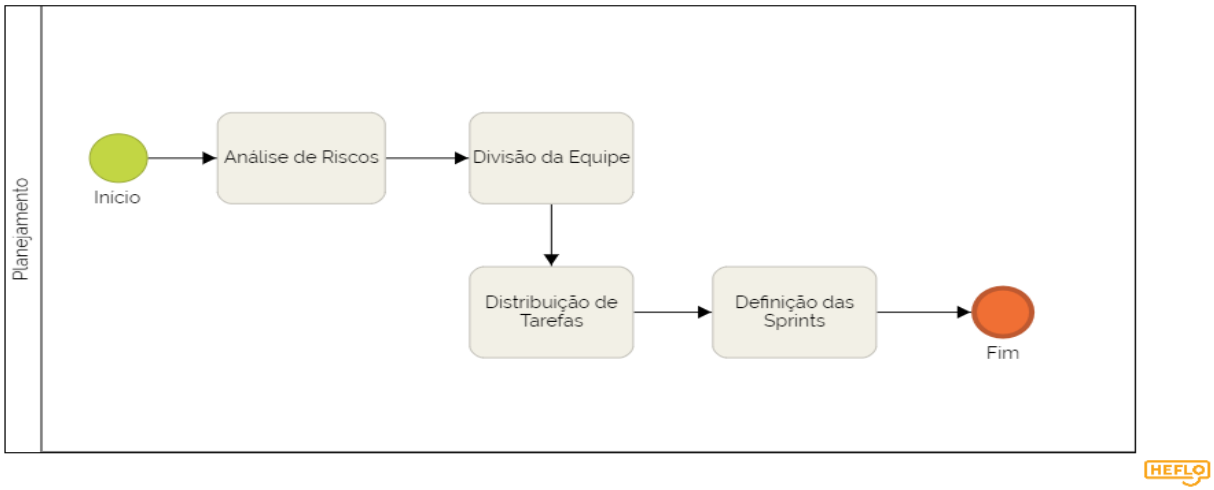

Figura 7 – Planejamento. (Fonte: Autor)

3.3 Desenvolvimento

Como terceira fase, tem-se a fase de Desenvolvimento, sendo em torno desta que todo o modelo se projeta. Nesta etapa, valorizam-se as reuniões diárias, promovendo a troca de experiência e cooperação entre os membros da equipe. Dessa forma, as iterações ocorrem de maneira ágil, atualizando as tarefas conforme são desenvolvidas, e ao fim de cada *Sprint*, atualiza-se também o protótipo até então desenvolvido.

A Figura 8 ilustra as sequintes etapas:

1. **Codificação**: Neste etapa, cada integrante da equipe possui tarefas pelas quais está responsável à desenvolver. Como um determinado período é definido na fase anterior, cada integrante deve fazer seu máximo a fim de atingir o objetivo, porém não será penalizado caso algum imprevisto ocorra, afinal, lidar com mudanças e imprevistos é um dos focos deste modelo.

2. **Reuniões Diárias**: Ao término ou antes de iniciar uma etapa de Codificação, devem ser realizadas Reuniões Diárias, sendo que cada integrante do time tem um determinado tempo para falar o que desenvolveu, bem como as dificuldades que teve e o que não conseguiu desenvolver.

3. **Troca de Experiências**: Com base no que foi dito nas Reuniões Diárias, os integrantes devem conversar e trocar experiências a fim de resolver possíveis problemas encontrados ou melhorar determinados aspectos da codificação.

4. **Atualização das Tarefas**: A equipe faz a atualização do que foi realizado e o responsável distribui as próximas tarefas. A cada atualização do que foi realizado, verifica-se:

    - Se todas as tarefas já foram finalizadas, caso a equipe termine antes do prazo;

- Se o prazo foi atingido, para enviar um protótipo funcional ou voltar para a codificação.

5. **Protótipo**: Quando a *Sprint* atinge a duração estipulada na fase de Planejamento, a equipe deve apresentar um protótipo rápido, de caráter evolutivo, mas funcional, ou seja, um protótipo funcionando porém como parte do projeto, sendo atualizado a cada *Sprint*.

   Caso as tarefas sejam finalizadas antes da duração estipulada, gera-se o protótipo e o desenvolvimento caminha para a próxima etapa.

   Se a equipe, não conseguir um protótipo minimamente funcional (o que deve ser medido pela própria equipe), cabe ao responsável pelo projeto conceder um tempo extra mínimo para que um protótipo funcional seja desenvolvido e passar para a próxima fase de desenvolvimento.

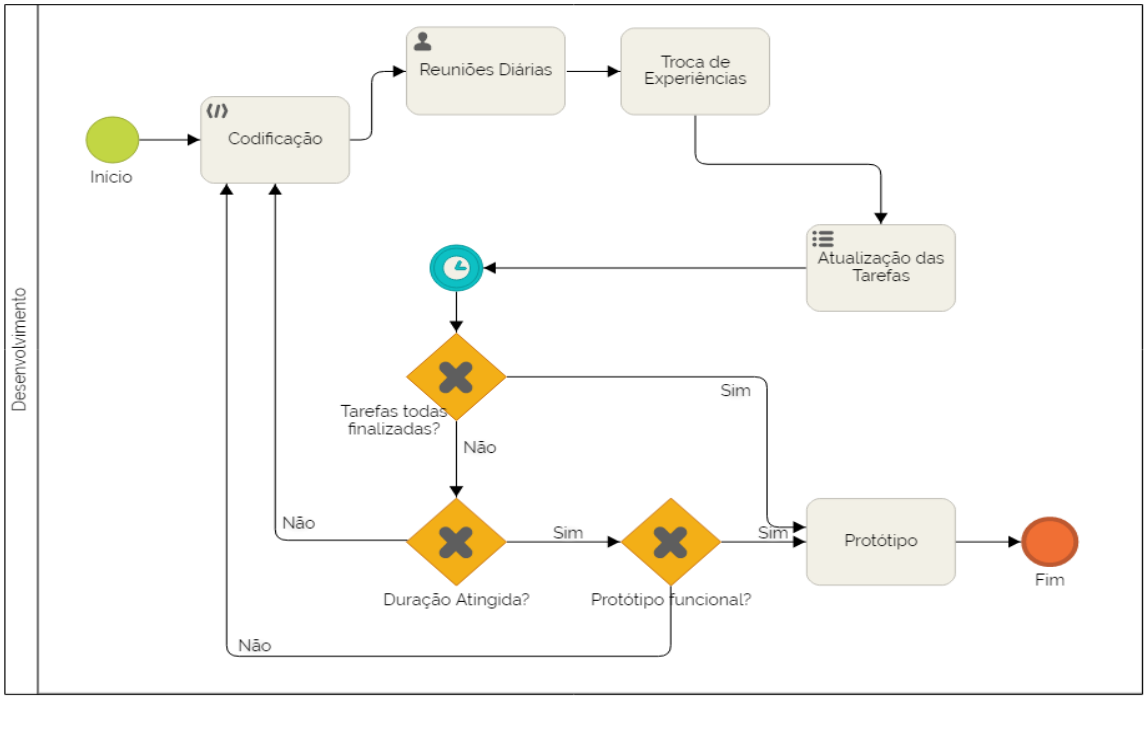

Figura 8 – Desenvolvimento. (Fonte: Autor)

3.4 Evolução

Nesta fase, o principal objetivo é envolver o cliente no projeto para obter um *feedback* e assim evoluir o produto. Após o desenvolvimento, tem-se um protótipo funcionando porém de caráter evolutivo que será apresentado ao cliente para uma verificação e validação do que foi desenvolvido. Paralelo à Evolução, mediante ao que foi acordado com o

cliente, é possivel enviar um protótipo funcionando de versão provisória a fim de que seja colocado em operação porém com as devidas restrições.

A Figura 9 ilustra as sequintes etapas:

1. **Verificação e Validação**: Nesta etapa é estabelecido uma comunicação com o cliente, a fim de apresentar o protótipo de versão mais recente para que os requisitos sejam verificados e validados.

2. **Reunião**: Uma reunião é feita com a equipe para que o *feedback* recebido anteriormente direcione os integrantes para a próxima etapa.

3. **Atualização das Tarefas**: Após a Reunião, o responsável pela equipe faz a atualização do que foi realizado, aplicando as alterações necessária para adequar as tarefas à qualquer tipo de mudança.

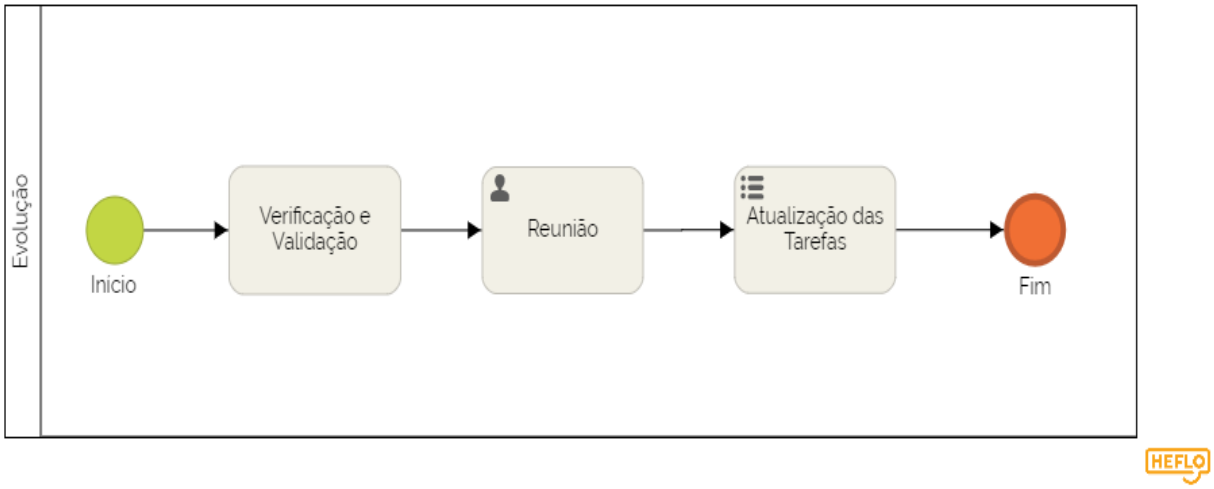

Figura 9 – Evolução. (Fonte: Autor)

3.5 Estudo de Caso

O modelo proposto no presente trabalho foi aplicado em um projeto cuja finalidade foi desenvolver uma aplicação para dispositivos móveis.

Resumidamente, a aplicação traz informações sobre pontos de venda de cerveja, bem como preços, avaliações e rotas até os estabelecimentos. Em sua versão final permite que o usuário, ao logar no aplicativo com uma conta de sua rede social, obtenha acesso para utilizar as funcionalidades da aplicação, são elas:

- Verificar as informações de todos os estabelecimentos cadastrados;

- Buscar por produtos no banco de dados do sistema e verificar os preços encontrados em cada estabelecimento;
- Gerar uma rota para um determinado estabelecimento, visto que todos os estabelecimentos são marcados no mapa da aplicação;
- Ver e enviar avaliações e comentários sobre os estabelecimentos cadastrados.

Na fase de **Preparação**, após o Levantamento de Requisitos, o Protótipo de Interface foi essencial para o projeto, pois seu uso possibilitou o descarte de certas ferramentas que poderiam ser utilizadas, visto que as limitações dessas acarretariam em não satisfazer alguns requisitos, e, juntamente com o Protótipo de Teste, foram validados e auxiliaram na Definição das Tecnologias.

Feita a preparação, seguem as iterações do modelo. No decorrer de cada iteração, as *Sprints* de desenvolvimento foram definidas, e ao fim, o protótipo foi evoluindo sempre com a constante comunicação com o cliente, a fim de verificar e validar os requisitos levantados.

Neste caso em específico, a ultima *Sprint*, foi finalizada na fase de Desenvolvimento, antes do prazo estipulado, visto que todo o processo ocorreu de forma positiva e todas as mudanças no decorrer do desenvolvimento foram aplicadas ao protótipo, assim, chegou-se em um Protótipo Final adequado.

3.6 Análise Comparativa

A fim de comparar as outras metodologias com o modelo proposto no presente trabalho, foram levantados alguns critérios para representar características gerais das metodologias abordadas.

Critérios	**GAIA Protótipo**	***Qualitas***	**MPDS-NPI**	***Software Novo***
Planejamento Predefinido	Não	Não	Sim	Sim
Planejamento Iterativo	Sim	Sim	Não	Sim
Interações ou Processos	Interações	Processos	Processos	Processos
Documentação	Pouca Documentação	Pouca Documentação	Bastante Documentação	Documentação Completa
Iterativo	Sim	Sim	Não	Sim

Comunicação com o Cliente	Essencial	Importante	Básica	Básica
Aberto a Mudanças	Sim	Parcialmente	Não	Parcialmente
Versões Parciais	Sim	Sim	Não	Sim
Testes Unitários	Sim	Sim	Sim	Sim
Testes Finais	Sim	Sim	Sim	Sim

Tabela 4 – Tabela comparativa GAIA Protótipo vs Modelos Correlatos. (Fonte: Autor)

A partir da Tabela 4, ao analisar as metodologias apresentadas, é possível inferir o seguinte:

- O MPDS-NPI é um modelo de caráter Tradicional, diferentemente dos outros, pois ainda que possua incrementos em sua fase de desenvolvimento, seu planejamento é predefinido do inicio ao fim do projeto.

- O *Software Novo* possui caráter Evolucionário, pois mesmo contando com um planejamento de fases predefinido, o planejamento das atividades de cada fase é realizado em cada iteração. A documentação completa, faz-se necessário devido à particularidade dos projetos que são desenvolvidos.

- Tanto o MPDS-NPI quanto o *Software Novo* possuem dificuldade para lidar com mudanças, sendo o primeiro fechado para mudanças no decorrer do desenvolvimento.

- O modelo *Qualitas* e o GAIA Protótipo são os que mais se assemelham, pois são Iterativos e também possuem características das Metodologias Ágeis. Entretanto, o GAIA Protótipo se mostrou mais aberto à mudanças, priorizando uma maior comunicação com o cliente, visto que ele é papel fundamental dentro do modelo. Outro importante diferencial é o foco em Interações ao invés de Processos, característica presente nas Metodologias Ágeis.

Vale ressaltar que o MPDS-NPI e o *Software Novo*, foram feitos especificamente para os projetos de caracteristicas particulares que desenvolvem, sendo respectivamente, o primeiro desenvolvido para uma Empresa Júnior do Núcleo de Projetos em Informática da UFSC e o segundo para o Departamento Nacional de Infraestutura de Transportes (DNIT).

3.7 Considerações Finais do Capítulo

Pode-se dizer que não existe uma melhor metodologia para o desenvolvimento de software, cada modelo possui caracteristicas próprias e etapas diferentes a serem seguidas. O modelo deve ser escolhido de acordo com as necessidades, aplicação e recursos do projeto.

O modelo proposto no presente trabalho, tem como base do seu desenvolvimento as práticas de prototipação e segundo [6], referente a Tabela 3, a prototipação é uma prática que pode minimizar ou solucionar 60% dos *Itens de Risco*, são eles:

- Desenvolvimento das funções erradas de software;
- Desenvolvimento de interface de usuário errada;
- "Gold plating"(ir além do esforço útil);
- Perdas devido a tarefas externas;
- Perdas devido ao desempenho em tempo real;
- Problemas com capacidades computacionais.

Sendo também uma metodologia mista de iterações e conceitos ágeis, o GAIA Protótipo apresenta uma maior facilidade para lidar com mudanças no decorrer do desenvolvimento do software, este que é também um *Item de Risco* no PDS. A grande diferença do modelo proposto para os modelos dos trabalhos relacionados citados anteriormente, é que o GAIA Protótipo possui como foco os indivíduos e interações, ao invés de processos, pois, segundo [11], a força de trabalho é o principal ativo organizacional, visto a influência direta da qualidade do trabalho em equipe na qualidade do *software* desenvolvido.

A capacidade e experiência da equipe são os fatores que devem direcionar a escolha da metodologia de desenvolvimento ideal. Não se faz necessária a escolha de um modelo isolado, visto que a combinação de metodologias mostra-se interessante para o desenvolvimento de determinados projetos.

4 CONCLUSÃO

Ao analisar o cenário atual, constata-se que, a sociedade sofre mudanças constantemente, não sendo diferente para as desenvolvedoras de *softwares*. Destaca-se então a importância de se utilizar um Processo de Desenvolvimento de *Software* a fim de que seja possível lidar com essas mudanças, tendo como principal ativo organizacional o trabalho em equipe, a fim de obter um *software* de melhor qualidade.

Por ser uma metodologia de poucas fases e de caráter simples, é possivel que, para futuros trabalhos, o modelo seja adaptado possibilitando a opção de ser englobado em qualquer outro modelo.

Construído a partir de premissas das Metodologias Ágeis, tem como foco a Interação e o *Software* em funcionamento ao invés de Processos e Documentação excessiva. Sua fase de desenvolvimento tem como objetivo a Prototipação Evolutiva, porém faz-se uso de protótipos de testes e interface, a fim de melhorar o levantamento de requisitos e comunicação com o cliente.

Por fim, com o intuito de minimizar os riscos recorrentes nos PDS, citados na Tabela 3, fez-se o uso das práticas de prototipação, descartáveis e evolutivos, juntamente com determinadas práticas do modelo Espiral e do *Scrum*.

Sendo assim, é possivel dizer que modelo mostra-se relevante para o PDS, visto também que o GAIA Protótipo obteve sucesso quando aplicado ao estudo de caso citado na Seção 3.5 do presente trabalho,.

Como trabalhos futuros, pretende-se aplicar o modelo proposto no processo de desenvolvimento de outros softwares com variadas equipes, tornar o modelo capaz de ser integrado em outros modelos de PDS e elicitar as melhores tecnologias existentes para facilitar a fase de desenvolvimento dos projetos e a comunicação entre o time.

REFERÊNCIAS

[1] SOARES, M. dos S. Comparação entre metodologias ágeis e tradicionais para o desenvolvimento de software. *INFOCOMP Journal of Computer Science*, v. 3, n. 2, p. 8–13, 2004.

[2] Palpite Digital. *RUP – Rational Unified Process Fases*. 2017. [Online; accessed December 22, 2017]. Disponível em: <https://www.palpitedigital.com/i/2300/disciplinas-fases-rup-511.jpg>.

[3] PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016.

[4] Catarina Gomes. *Processo Scrum*. 2017. [Online; accessed December 22, 2017]. Disponível em: <https://cdn2.hubspot.net/hubfs/2896172/imagens/blog/lean/processo-scrum.png>.

[5] TELES, V. M. Extreme programming. *São Paulo: Novatec*, 2004.

[6] ALMEIDA, G. A. M. d. *Fatores de escolha entre metodologias de desenvolvimento de software tradicionais e ágeis.* Tese (Doutorado) — Universidade de São Paulo, 2017.

[7] FOSE 2014: Proceedings of the on Future of Software Engineering. New York, NY, USA: ACM, 2014. ISBN 978-1-4503-2865-4.

[8] PAETSCH, F.; EBERLEIN, A.; MAURER, F. Requirements engineering and agile software development. In: *WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.* [S.l.: s.n.], 2003. p. 308–313. ISSN 1080-1383.

[9] STANDARDIZATION, I. O. *ISO 12207 (Standard for the Information Technology - Software Life Cycle Process)*. [S.l.], 2013. Disponível em: <http://www.iso.org/iso/catalogue_detail?csnumber=43447>. Acesso em: 20.07.2017.

[10] SALIM, G. A. Guia de implantação de processos de gerenciamento de pessoas para organizações de desenvolvimento de software. Universidade Estadual Paulista (UNESP), 2017.

[11] WEIMAR, E. et al. The influence of teamwork quality on software team performance. *arXiv preprint arXiv:1701.06146*, 2017.

[12] KAN, S. H. *Metrics and models in software quality engineering*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002.

[13] CROSBY, P. B. Quality is free: The art ofmaking quality certain. *New York*, 1979.

[14] JURAN, J.; JR, G. *FM Quality planning and analysis: from product evelopment though usage*. [S.l.]: New York: McGraw-Hill, 1970.

[15] ROCHA, T. Á. da; OLIVEIRA, S. R. B.; VASCONCELOS, A. M. L. de. Adequação de processos para fábricas de software. *Anais do Simpósio Internacional de Melhoria de Processo de Software (SIMPROS)*, 2004.

[16] SCOTT, K. *O processo unificado explicado*. [S.l.]: Bookman, 2003.

[17] CHERUBIN, P. F. Estratégias de negócio em software-houses. *Revista da FAE*, v. 3, n. 2, 2017.

[18] LEI, H. et al. A statistical analysis of the effects of scrum and kanban on software development projects. *Robotics and Computer-Integrated Manufacturing*, Elsevier, v. 43, p. 59–67, 2017.

[19] BRITO, M. F. D. et al. Knowledge transfer in a management process for outsourced agile software development. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2017.

[20] NUNES, R. D. A implantação das metodologias ágeis de desenvolvimento de software scrum e extreme programming (xp): uma alternativa para pequenas empresas do setor de tecnologia da informação. *ForScience*, v. 4, n. 2, 2017.

[21] PRIKLADNICKI, R. Problemas, desafios e abordagens do processo de desenvolvimento de software. 2002. *Trabalho Individual I, FACIN-PPGCC, PUCRS, Porto Alegre*, 2002.

[22] ALMEIDA, C. C. d. J. et al. Qualitas: uma modelo de processo de desenvolvimento de software orientado a modelos. Ciência da Computação, 2014.

[23] SOUZA, M. B. de. Modelo de processo de software: aplicação em uma empresa júnior.

[24] FINANÇAS-DAF, D. D. A. E. Metodologia de desenvolvimento de software (mds) do dnit.

Apêndices

Anexos