UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
COORDENADORIA DO CURSO DE ENGENHARIA DE SOFTWARE

MAICON JUNIOR SILVEIRA

PRIORIZAÇÃO AUTOMÁTICA DE ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

TRABALHO DE CONCLUSÃO DE CURSO

DOIS VIZINHOS

2018

MAICON JUNIOR SILVEIRA

PRIORIZAÇÃO AUTOMÁTICA DE ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia de Software, da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Msc. André Roberto Ortoncelli

DOIS VIZINHOS

2018

TERMO DE APROVAÇÃO

Priorização Automática de Atividades de Desenvolvimento de Software: Um estudo de Caso.

por

Maicon Junior Silveira

Este Trabalho de Conclusão de Curso foi apresentado em 04 de Dezembro de 2018 como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software. O(a) candidato(a) foi arguido(a) pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Andre Roberto Ortoncelli
Presidente da Banca

Andre Luiz Marasca
Membro Titular

Franciele Beal
Membro Titular

* A Folha de Aprovação assinada encontra-se na Coordenação do Curso

Dedico este trabalho aos meus pais, que não mediram esforços para que eu chegasse até esta etapa da minha vida.

AGRADECIMENTOS

A Deus por ter me dado saúde e força para superar as dificuldades.

A esta universidade, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior, eivado pela acendrada confiança no mérito e ética aqui presentes.

Ao meu orientador André Roberto Ortoncelli, pelo suporte no pouco tempo que lhe coube, pelas suas correções e incentivos.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

"Quando algo é importante o suficiente, você realiza mesmo que as chances não estejam a seu favor". Elon Musk

RESUMO

SILVEIRA, MAICON. PRIORIZAÇÃO AUTOMÁTICA DE ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO. 40 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2018.

A priorização de atividades de desenvolvimento em empresas de software é comumente realizada de forma arbitrária e subjetiva, com base unicamente na experiência dos funcionários da empresa. Nesse contexto, este trabalho tem por objetivo apresentar um método de priorização automática de atividades de desenvolvimento de software, por meio de um algoritmo que combina o uso de uma Rede Neural Artificial (RNA) com um método de ordenação. Para validar a acurácia do método proposto, foi conduzido um estudo de caso, com dados de atividade fornecidos pela empresa Visual Software. Os experimentos permitiram avaliar a acurácia do método proposto qualitativamente e quantitativamente, gerando resultados satisfatórios, que indicam a viabilidade da implementação do método proposto na empresa alvo do estudo de caso.

Palavras-chave: Inteligência Artificial, Redes Neurais Artificias, Aprendizagem de Máquina, Priorização de Atividades, Processo de Desenvolvimento de Software

ABSTRACT

SILVEIRA, MAICON. TITLE IN ENGLISH. 40 f. Trabalho de Conclusão de Curso – Coordenadoria do Curso de Engenharia de Software, Universidade Tecnológica Federal do Paraná. Dois Vizinhos, 2018.

The prioritization of development activities in software companies commonly is done in an arbitrary and subjective way, based solely on the experience of the company's employees. In this context, this work aims to present a method of automatic prioritization of software development activities, through an algorithm that combines the use of an Artificial Neural Network (RNA) with an ordering method. To validate the accuracy of the proposed method, a case study was conducted, with activity data provided by Visual Software. The experiments allowed to evaluate the accuracy of the proposed method qualitatively and quantitatively, generating satisfactory results, which indicate the feasibility of implementing the proposed method in the company targeted by the case study.

Keywords: Artificial Intelligence, Artificial Neural Networks, Machine Learning, Prioritizing Activities, Software Development Process

LISTA DE FIGURAS

FIGURA 1 – Modelo Simplificado de Neurônio Artificial 17
FIGURA 2 – Representação de RNA 18
FIGURA 3 – Erro Médio Quadrático do método proposto 27
FIGURA 4 – Processo Geral 34
FIGURA 5 – Processo de Analise de Viabilidade 35
FIGURA 6 – Processo de Análise Orçamentária 35
FIGURA 7 – Processo de Elaborar Proposta Comercial 36
FIGURA 8 – Processo de Análise de Requisitos 36
FIGURA 9 – Processo de Validação de Requisitos 37
FIGURA 10 – Processo de Planejar Desenvolvimento 37
FIGURA 11 – Processo de Desenvolver os Requisitos 38
FIGURA 12 – Processo de Verificar e Validar as Funcionalidades 38
FIGURA 13 – Processo de Verificar Defeitos Registrados 39
FIGURA 14 – Processo de Corrigir Defeitos 39
FIGURA 15 – Processo de Liberar Versão 40

LISTA DE SIGLAS

RNA	Rede Neural Artificial
EQM	Erro Médio Quadrático

LISTA DE SÍMBOLOS

Σ	Letra Sigma
φ	Letra Fi

SUMÁRIO

1 INTRODUÇÃO 12
1.1 OBJETIVOS 13
1.1.1 Objetivo Geral 13
1.1.2 Objetivos Específicos 13
1.2 ORGANIZAÇÃO DO TEXTO 13
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 APRENDIZAGEM DE MÁQUINA 15
2.1.1 Redes Neurais Artificiais 16
2.2 ALGORITMO DE ORDENAÇÃO 18
2.3 PROCESSO DE PRODUÇÃO DE SOFTWARE 19
2.4 PRIORIZAÇÃO DE ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE COM APRENDIZAGEM DE MÁQUINA 20
2.5 PRIORIZAÇÃO DE ATIVIDADES NA VISUAL SOFTWARE 21
3 MÉTODO DE PRIORIZAÇÃO PROPOSTO 22
3.1 EXTRAÇÃO DOS DADOS 24
3.2 TREINAMENTO 24
3.3 CLASSIFICAÇÃO 25
3.4 AVALIAÇÃO 25
4 RESULTADOS EXPERIMENTAIS 26
5 CONCLUSÕES 28
REFERÊNCIAS 29
Apêndice A – PROCESSO DE DESENVOLVIMENTO 32
A.1 DESCRIÇÃO DA EMPRESA 32
A.2 ATIVIDADES DO PROCESSO DE DESENVOLVIMENTO 32

1 INTRODUÇÃO

Ambientes de desenvolvimento de software se caracterizam por envolverem um alto dinamismo. Mesmo que exista um processo muito bem estruturado, ele ainda estará suscetível a perturbações internas e externas, portanto é importante que o processo se mantenha altamente adaptativo (RIBEIRO et al., 2017).

Em tais ambientes, a priorização de atividades que irão consumir recursos do projeto pode causar um impacto direto no resultado do processo e do produto a ser desenvolvido, pois direciona as pessoas e equipes de desenvolvimento.

Nesse contexto, o projeto apresentado nesse trabalho surgiu de uma demanda apresentada pela empresa Visual Software, em relação a priorização automática de atividades de desenvolvimento de software.

Ressalta-se que a priorização de atividades em empresas de software é realizada na maioria das vezes de forma arbitrária e subjetiva e a proposta do desenvolvimento de uma técnica determinística com critérios bem definidos é de grande importância (RIBEIRO et al., 2017).

A Visual Software possui hoje um processo de desenvolvimento de software definido, priorizando atividade de acordo com a opinião de especialistas, porém tem interesse em um sistema para automatização dessa tarefa.

Como a empresa possui o histórico de todas as atividades, optou-se por utilizar esses dados para treinar um classificador com uso de uma Rede Neural Artificial (RNA). Destaca-se que a empresa permitiu o uso e divulgação dos dados.

O método de priorização proposto utiliza uma RNA que é aplicada em conjunto com método de ordenação, de modo similar ao método apresentado em (PAETZOLD; SPECIA, 2017), que escolhe a melhor simplificação para um determinado termo.

Considerando um conjunto de n termos, selecionar um termo de maior prioridade equivale a dar a um termo, a maior prioridade, dessa forma, o problema apresentado em

(PAETZOLD; SPECIA, 2017) é similar ao método de priorização de atividade proposto nesse trabalho.

1.1 OBJETIVOS

Essa seção apresenta o objetivo geral e os objetivos específicos do trabalho proposto.

1.1.1 OBJETIVO GERAL

O objetivo geral desse projeto é criar um método de priorização automática de atividades de desenvolvimento de software.

1.1.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral desse trabalho, os seguintes objetivos específicos também foram alcançados:

- Definir os critérios de priorização das atividades.
- Desenvolver um algoritmo baseado em redes neurais, capaz de priorizar automaticamente atividades de desenvolvimento de software.
- Criar uma base de dados para validação do método do método proposto, com base nos dados fornecidos pela Visual Software.
- Avaliar os resultados do método de priorização proposto, com os dados fornecidos pela Visual Software.
- Avaliar a aplicabilidade do método de priorização proposto no processo de desenvolvimento de software da Visual Software.

1.2 ORGANIZAÇÃO DO TEXTO

O restante do texto encontra-se organizado da seguinte forma: O Capítulo 2 apresenta uma fundamentação teórica, sobre os temas necessários para o entendimento e contextualização do trabalho proposto em relação ao estado da arte. Detalhes sobre o método de priorização proposto são apresentados no Capítulo 3. Os resultados dos experimentos são apresentados no Capítulo 4. Por fim, a conclusão do trabalho é apresentada no Capítulo 5.

Complementando o conteúdo do trabalho, o Apêndice A, apresenta detalhes do processo de desenvolvimento utilizado na Visual Software.

2 FUNDAMENTAÇÃO TEÓRICA

Esse Capítulo visa apresentar conceitos importantes para o entendimento do trabalho proposto, e também para sua contextualização em relação ao estado da arte. Com esse objetivo, o restante do Capítulo encontra-se organizado da seguinte forma: Conceitos sobre aprendizagem de máquina com foco em RNA são apresentados na Seção 2.1. A Seção 2.2 define o conceito de algoritmo de ordenação. Conceitos de processo de produção de software são apresentados na Seção 2.3, com foco na importâncias da priorização de atividade e requisitos.

Uma vez introduzida a importância da priorização de tarefas dentro de um processo de desenvolvimento de software, a Seção 2.4 descreve técnicas de priorização aplicadas em processos de produção de software desenvolvidas com uso de Aprendizagem de Máquina. Por fim, a Seção 2.5 apresenta técnicas de priorização que já foram desenvolvidas para Visual Software.

2.1 APRENDIZAGEM DE MÁQUINA

A Aprendizagem de Máquina é uma área da Inteligência Artificial que integra conceitos de computação e da estatística para a construção de sistemas que são capazes de aprender com os dados disponibilizados (dados de treinamento), e em alguns casos, são capazes de aprender até com suas próprias decisões (utilizando as decisões tomadas para retroalimentar a base de treinamento) (LUGER, 2013).

Um sistema de aprendizado pode ser definido como um sistema computacional que toma decisões baseadas em experiências acumuladas através da resolução de problemas anteriores (WEISS; KULIKOWSKI, 1991).

O aprendizado de máquina pode ser dividido em duas categorias: aprendizado supervisionado e aprendizado não supervisionado (MONARD; BARANAUSKAS, 2003). Tais categorias serão detalhados a seguir.

No aprendizado supervisionado é fornecido ao sistema de aprendizado um conjunto de exemplos, tal conjunto é denominado como conjunto de treinamento, que será utilizado para treinar o classificador. Para cada exemplo do conjunto é necessário associar um rótulo que irá definir a classe a qual o exemplo pertence. (BATISTA et al., 2003).

No aprendizado não supervisionado é fornecido um conjunto de exemplos contendo apenas vetores, sem nenhum dado de classe. O objetivo é construir um modelo que busque regularidades nos exemplos, formando agrupamento de exemplos que contenham características similares (BATISTA et al., 2003).

2.1.1 REDES NEURAIS ARTIFICIAIS

Compreender o funcionamento de um cérebro biológico e reproduzi-lo de forma artificial, vem sendo durante gerações a ambição de uma infinidade de pesquisadores. O cérebro humano processa as informações de forma totalmente diferente de um computador convencional, é altamente complexo, não linear e paralelo (HAYKIN, 2007).

As RNA são modelos matemáticos que têm capacidade computacional adquirida por meio de aprendizagem e generalização (BRAGA; CARVALHO; LUDERMIR, 2000). Todos esses modelos almejam recriar habilidades de processamento de informação, aprendizado e adaptação que são presentes em seres vivos. Uma rede neural pode ser vista como uma máquina adaptativa, constituída de unidades simples de processamento, com vocação natural para armazenar conhecimento e torná-lo disponível para uso (HAYKIN, 2007).

O modelo matemático de um neurônio é composto por três elementos básicos, conforme ilustrado pela Figura 1, em seguida são detalhados cada um dos elementos.

- **Entrada**: O primeiro elemento é constituído por um conjunto de n conexões de entradas $(x1, x2, ..., xn)$, essas entradas são caracterizadas por pesos $(p1, p2, ..., pn)$;
- **Somador**: O somador (Σ) tem por objetivo acumular os sinais de entrada;
- **Função de Ativação**: A função de ativação (φ) limita o intervalo permissível de amplitude do sinal de saída (y) a um valor fixo;

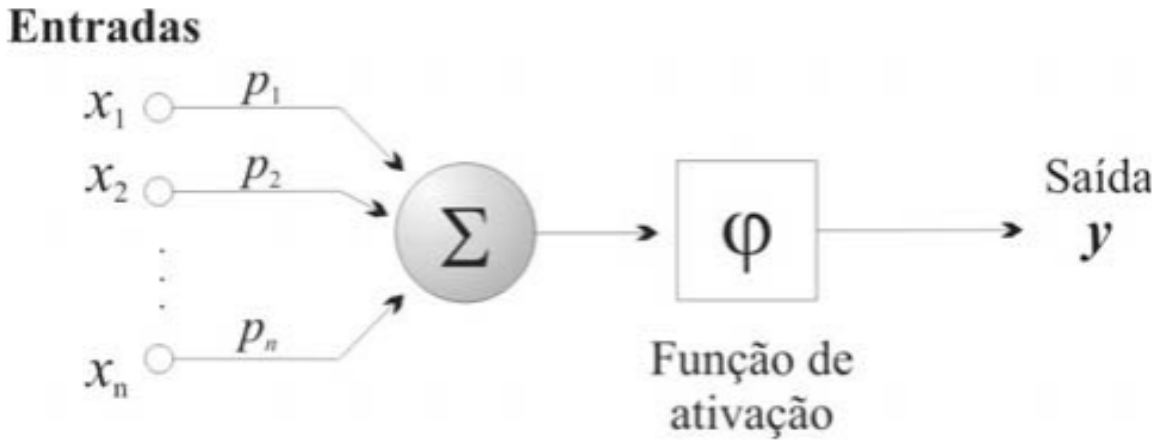

Figura 1: Modelo de Neurônio Artificial.
Fonte: Haykin (2007)

Com a combinação de diversos neurônios, obtemos a formação de uma RNA. As RNA podem ser definidas como modelos que buscam simular o processamento de informação que é efetuado pelo cérebro humano. A união dos neurônios ocorre por meio de conexões sinápticas. (FERNEDA, 2006).

A RNA pode ser exemplificada com a utilização de um grafo, onde os nós são os neurônios e as ligações fazem a função das sinapses (FERNEDA, 2006), conforme exemplificado na Figura 2.

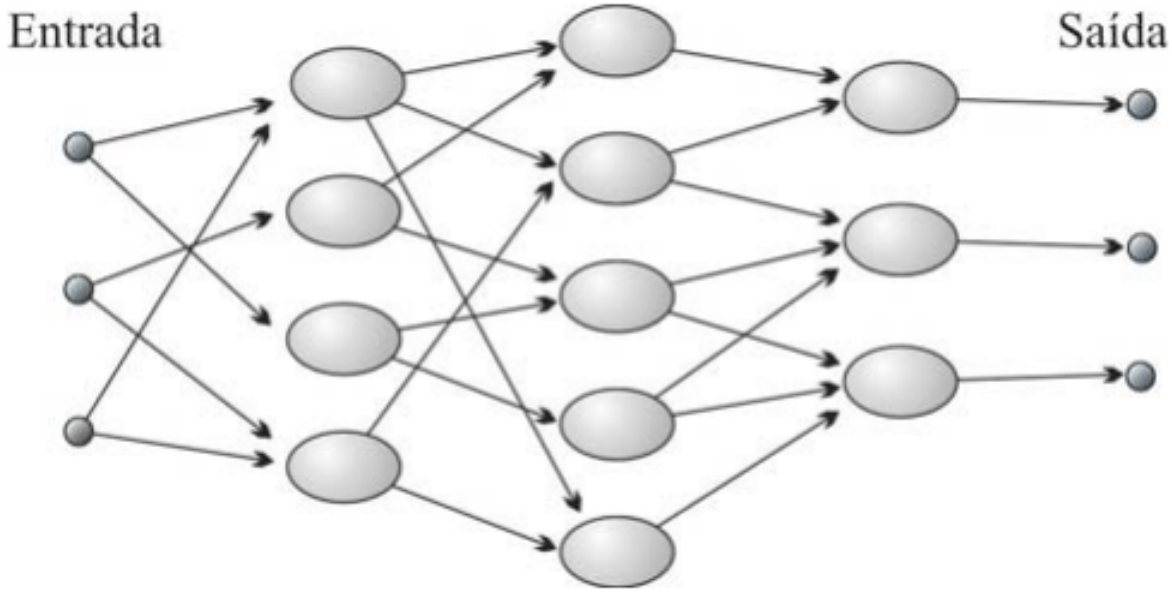

Figura 2: Representação de RNA.
Fonte: Haykin (2007)

A RNA possui várias propriedades, sendo uma das mais importantes a capacidade de aprender por meio de exemplos, e a partir disso realizar inferências sobre o que aprendeu, tendo melhora gradativa em seu desempenho. As RNA utilizam algoritmo de aprendizagem tal qual tem como tarefa ajustar os pesos de suas conexões (BRAGA; CARVALHO; LUDERMIR, 2000).

2.2 ALGORITMO DE ORDENAÇÃO

Considerando uma sequência qualquer de n elementos S, ordenada ou não, um algoritmo de ordenação pode ser representado por uma função $\psi(S)$ que recebe como parâmetro uma sequência S e retorna uma sequência ordenada S_O, tal que todos os elementos contidos em S_O estão ordenados com base em uma ordem lógica, sendo que comumente são utilizados critérios de ordenação numérica e a lexicográfica.

Existem diferentes implementações de métodos de ordenação, com desempenho diferente, quanto ao número de operações realizadas para ordenar um conjunto, por exemplo, para um conjunto de n elementos, os métodos *Bubble Sort* e *Insertion Sort* executam n^2 operações, enquanto métodos mais eficientes, tal como o *Meger Sort* e o *Quick Sort*, irão executar apenas $nlog_2n$ operações (CORMEN et al., 2009).

Existem também métodos de ordenação híbridos, tal como o *Tim Sort*, que

consegue ordenar uma sequência parcialmente ordenadas com menos de $log_{n!}$ operações, combinado de modo híbrido os algoritmos *Insertion Sort* e *Merge Sort* (AUGER; NICAUD; PIVOTEAU, 2015). Destaca-se que pela eficiência do algoritmo TimSort, ele foi o escolhido para implementação de métodos de ordenação padrão nas linguagens de programação Python [1].

2.3 PROCESSO DE PRODUÇÃO DE SOFTWARE

Um processo de desenvolvimento de software pode ser visto como um conjunto de atividades organizadas, usadas para definir, desenvolver, testar e manter um software. É uma atividade complexa (CLARKE; O'CONNOR; LEAVY, 2016) que é altamente sensível à interação humana e ao trabalho em equipe (YILMAZ; O'CONNOR; CLARKE, 2012).

Muitas abordagens diferentes para o desenvolvimento de software foram propostas, cada uma delas oferecendo algo especial, único ou novo, que representa uma melhoria em relação aos demais modelos de processo (CLARKE; O'CONNOR; LEAVY, 2016).

Existem autores que defendem a premissa de que nenhum modelo de processo de desenvolvimento de software é perfeitamente adequado para todas as necessidades específicas de uma empresa (CLARKE et al., 2015), sendo necessário fazer adaptações dos processos modelos de processo utilizados (COLEMAN; O'CONNOR, 2008).

Duas das etapas importantes em qualquer processo de desenvolvimento de software são o levantamento de requisitos e a definição de atividades de desenvolvimento de software, que são conceitos relacionados, porém distintos. As atividades são planejadas com base nos requisitos, um requisito pode ser implementado com o uso de uma ou mais atividades.

Tais atividades são importantes, pois falhas no entendimento dos requisitos e no planejamento das atividades podem causar retrabalho e gastos adicionais, além de impactos no cronograma do projeto.

Portanto, torna-se necessário dispor de processos adequados para planejar, classificar e priorizar requisitos e atividades de modo eficiente, evitando quebras de confiança, contrato ou acordo com cliente.

Nesse contexto, a priorização tem o objetivo de classificar os requisitos/atividades em sua ordem de importância, impactando as liberações subsequentes da implementação. É um passo importante na tomada de decisões cruciais para aumentar o valor econômico de

[1] https://bugs.python.org/file4451/timsort.txt

um sistema e também na qualidade e eficácia do processo de desenvolvimento de software (ACHIMUGU et al., 2014).

Métodos de priorização também podem ser definidos como processo de identificar os requisitos mais valiosos em sua ordem de importância ou preferência (AGUIAR et al., 2015; SHAO et al., 2017).

Diferentes técnicas têm sido propostas na literatura por autores e acadêmicos, com o objetivo de classificar os requisitos baseando-se em sua ordem de importância (ACHIMUGU et al., 2014; AGUIAR et al., 2015; SHAO et al., 2017), porém nenhum trabalho relacionado a priorização de atividades foi encontrado na literatura (com exceção de trabalhos relacionados a Visual Software, que serão apresentados na Seção 2.5).

2.4 PRIORIZAÇÃO DE ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE COM APRENDIZAGEM DE MÁQUINA

Métodos de priorização vêm sendo aplicados em diferentes áreas, tais como, na área médica para priorização de pacientes (MYLAVARAPU; THOMAS, 2017), na priorização de estudantes com risco de não se graduarem no tempo correto (AGUIAR et al., 2015), e também diferentes aplicações em relação ao processo de desenvolvimento de software, conforme detalhado a seguir.

Quanto aos métodos de priorização aplicados no processo de desenvolvimento de software com uso de algoritmos de aprendizagem de máquina, poucos trabalhos foram encontrados em relação a priorização de atividades de desenvolvimento de software, porém um volume considerável de trabalhos é relacionado a priorização de requisitos, que é atividade diretamente ligada com a priorização de atividades (PERINI; SUSI; AVESANI, 2013; TONELLA; SUSI; PALMA, 2013; KARLSSON; WOHLIN; REGNELL, 1998; SHAO et al., 2017).

Além disso, a partir desta pesquisa, observa-se que, a maioria das técnicas existentes de aprendizado de máquina sofre com a questão das atualizações de *rank*. As técnicas proeminentes de aprendizado de máquina que sofrem com essa limitação são o ranking da base de casos (PERINI; SUSI; AVESANI, 2013); técnica de priorização de algoritmos genéticos interativos (TONELLA; SUSI; PALMA, 2013); Árvore de busca binária (KARLSSON; WOHLIN; REGNELL, 1998), EVOLVE (GREER; RUHE, 2004), *DRank*, baseado no algoritmo DE *PageRank* (SHAO et al., 2017).

Além dos algoritmos de priorização de atividades e requisitos, algoritmos de

priorização também são explorados para priorização de casos de teste (SANTOS, 2017; LACHMANN et al., 2016; CHEN et al., 2017; SPIEKER et al., 2017) e também em relação a priorização de *BUGS* (UDDIN et al., 2017; KANWAL; MAQBOOL, 2010; YANG; ZHANG; LEE, 2014).

Deve-se destacar também, que no processo de desenvolvimento de software, os métodos de aprendizagem de máquina vêm sendo explorados não somente para priorizar atividades, mas também para classificação em diferentes contextos, tais como, a classificação da qualidade de requisitos (PARRA et al., 2015) e perfis de programadores (BEAL; BASSI; PARAISO, 2017).

2.5 PRIORIZAÇÃO DE ATIVIDADES NA VISUAL SOFTWARE

Dois algoritmos de priorização já foram criados com base no processo de desenvolvimento da Visual Software (SILVEIRA et al., 2017; SANTOS, 2017). (SANTOS, 2017) Trata da priorização de atividade de teste de software. Neste trabalho foram avaliados os históricos de tempo dos testes de cada um dos testadores. O objetivo principal é determinar qual o melhor testador para destinar a atividade, de modo que o testador que tenha mais eficiência em determinado módulo receba esta atividade automaticamente.

Já (SILVEIRA et al., 2017) trata de atividades de desenvolvimento de software, este trabalho foi desenvolvido como uma etapa preliminar ao desenvolvimento do trabalho apresentado nesta monografia. Em (SILVEIRA et al., 2017) foi criada uma árvore de decisão com base no conhecimento de especialistas da empresa, porém ao estabelecer pesos fixos, em certo ponto da classificação as atividades com características totalmente diferentes começar a ter o mesmo peso, dessa forma o algoritmo não consegue categorizar corretamente qual atividade deve ser realizada primeiro.

3 MÉTODO DE PRIORIZAÇÃO PROPOSTO

Considerando o conjunto de atividade $A = a_1, a_2, ..., a_n$ composto por n atividades, tal que cada atividade a_x é um vetor composto por 6 atributos: i) Categoria da Tarefa; ii) Origem da Tarefa; iii) Categoria do Cliente; iv) Média de Satisfação; v) Tempo em Espera; e vi) Cliente em Implantação. Detalhes sobre cada um dos atributos das atividades que compõe o conjunto A são apresentados a seguir:

1. Categoria da Tarefa: Esta informação está relacionada ao tipo de intervenção que será realizada na atividade, as intervenções possíveis são: i) BUG; ii) Customização Paga; iii) Melhoria e iv) Customização não Paga. Cada uma das possíveis categorias de tarefas é detalhada a seguir:

    - BUG: Corresponde a todas as falhas manifestadas nos softwares.
    - Melhoria: Corresponde a novas funcionalidades nos softwares que geram benefícios a todos os clientes que o utilizarem.
    - Customização: Corresponde a novas funcionalidades que possuam muitas particularidades de determinado cliente, tornando o recurso útil apenas a ele. Nessa categoria é avaliada a possibilidade de realizar o desenvolvimento mediante o pagamento do cliente solicitante.

2. Origem da Tarefa: Esta informação está relacionada ao ponto de partida de cada atividade, que pode ser externa, se a mesma foi registrada para algum dos clientes ou interna, se partiu de uma investigação de um dos colaboradores da própria empresa.

3. Categoria do Cliente: Esta informação está relacionada ao grupo em que o cliente pertence, podendo ser do grupo *premium* (que geram um retorno financeiro fundamental para a Visual Software) ou do grupo normal (Todo o restante dos clientes que não se encaixarem na faixa *premium*).

4. Cliente em Implantação: Esta informação identifica se o cliente está ou não em processo de implantação, caso sim, as solicitações dele devem ser priorizadas, pois o

objetivo é tornar o cliente autossuficiente na utilização dos softwares o mais rápido possível, diminuindo assim custos da Visual Software no processo de adequação do cliente.

Estar em processo de implantação significa que o cliente adquiriu a licença de utilização do software recentemente, e precisa de uma atenção especial, para que seu primeiro contato com a empresa não seja insatisfatório.

5. Média de Satisfação: As notas de satisfação são coletadas pelos softwares da Visual Software, os usuários são notificados mensalmente para atribuir uma nota de 0 a 5 sobre sua satisfação com relação a qualidade dos softwares. É realizado uma média trimestral entre todos os usuários e tal informação é utilizada para priorizar as atividades dos clientes com menor média de satisfação.

6. Tempo em Espera: Esta informação está relacionada ao tempo em que uma atividade está parada sem nenhuma interação, avaliando apenas as horas úteis da semana. Tal informação auxilia a priorizar atividades que estejam a muito tempo sem uma interação, na qual podem haver clientes insatisfeitos pela demora excessiva de uma entrega.

O conjunto A é dividido em dois subconjuntos $A_{treinamento}$ e A_{teste}, tal que um subconjunto de α atividades do conjunto A são utilizadas para criar o conjunto $A_{treinamento}$ e as demais atividades do conjunto A são usadas para criar o conjunto A_{teste}.

O conjunto $A_{treinamento}$ é um conjunto de pares de atividades utilizado para treinar a RNA. Para cada combinação possível de pares de atividade $a_x, a_y \in A$ é criado um registro $at_z \in A_{treinamento}$. Cada registro at_z é composto por 13 atributos: os seis atributos da atividade $a_x \in A$, mais os seis atributos da atividade $a_y \in A$ e mais ainda um rótulo atribuído a um especialista que indica qual atividade tem maior prioridade em relação a outra.

Se a atividade a_x possui maior prioridade que a atividade a_y é atribuído o rótulo -1, se atividade a_y possui maior prioridade é atribuído o rótulo 1 e o rótulo zero é atribuído caso as atividades tenham a mesma prioridade.

O conjunto $A_{treinamento}$ é utilizado para treinar uma RNA com um sistema de regressão. Em seguida tal RNA é utilizada para priorizar as atividades de desenvolvimento software contidas no conjunto A_{teste}, em conjunto com o algoritmo de ordenação TimSort.

Já o conjunto A_{teste} é passado como parâmetro para o método de ordenação

TimSort, que compara pares de atividades $a_x, a_y \in A_{teste}$ por meio do *callback* de RNA, que por meio de uma regressão, retorna um valor entre -1 e 0, se a prioridade de a_x é menor que a de a_y, um valor entre 0 e 1, se a prioridade de a_y é menor que a de a_x, ou retorna zero, se a_x e a_y possuem a mesma prioridade.

Dessa forma é possível comparar pares de atividade em relação a sua prioridade, podendo então aplicar um método de ordenação para ordenar as atividade de acordo com o nível de prioridade.

Destaca-se que, para executar o método de priorização descrito, quatro etapas principais devem ser seguidas para aplicação do método proposto. A seguir mais detalhes sobre cada uma dessas etapas são apresentados: i) Extração dos Dados; ii) Treinamento; iii) Classificação e iv) Avaliação. Mais detalhes de cada uma das etapas serão apresentados a seguir.

3.1 EXTRAÇÃO DOS DADOS

Uma vez obtida a base de dados fornecida pela empresa de Visual Software (composta por 69535 atividades), se dá início a etapa de extração dos dados, na qual é produzido o conjunto de atividades A.

Nessa etapa, para cada atividade $a_x \in A$, foram extraídos os 6 atributos necessários: i) Categoria da Tarefa; ii) Origem da Tarefa; iii) Categoria do Cliente; iv) Média de Satisfação; v) Tempo em Espera e vi) Cliente em Implantação.

O conjunto de atividades A, extraído dos dados fornecidos pela Visual Software foi disponibilizado no link: https://1drv.ms/t/s!AtoUA1clU7Nrip8ZS6YUu5jPdvZVGQ. Tais dados estão disponibilizados na forma de um *script* de criação e povoamento de um banco de dados em Mysql.

Qualquer pessoa pode utilizar tais dados, desde que citem esse trabalho, ou artigo(s) que poderão publicados posteriormente pelo autor deste trabalho, com o mesmo tema.

3.2 TREINAMENTO

Após a execução da atividade de extração dos dos dados, obtendo-se o conjunto A, tal conjunto foi dividido em dois subconjuntos $A_{treinamento}$ e A_{teste}, sendo que o conjunto $A_{treinamento}$ foi composto por 100 atividades da base de dados, referentes as tarefas

desenvolvidas pela empresa em um determinado período (por exemplo, um mês). Já o conjunto A_{teste} foi composto pelas demais $n - m$ atividades.

Cada par de atividades do conjunto $A_{treinamento}$ foi priorizado manualmente por um especialista da empresa.

3.3 CLASSIFICAÇÃO

Após a criação dos conjuntos $A_{treinamento}$ e A_{teste}. Na etapa de classificação, o conjunto $A_{treinamento}$ foi utilizado para treinar a RNA que posteriormente foi utilizada para classificar a prioridade de pares de atividades do conjunto A_{teste}.

As atividade do conjunto A_{teste} foram classificadas em pares, de acordo com o algoritmo TimSort, que produziu um conjunto ordenado de atividade, sendo a posição da atividade neste conjunto, referente ao valor do atributo prioridade dessa atividade.

Destaca-se que foi utilizado o método de ordenação TimSort devido a sua eficiência e também devido a tal método já estar implementado na linguagem de programação Python, que foi utilizada para o desenvolvimento do algoritmo.

3.4 AVALIAÇÃO

A aplicação da etapa de avaliação consistiu em duas abordagens para avaliar os resultados: Uma avaliação qualitativa, na opinião de especialistas da empresa e uma avaliação quantitativa, com base na comparação com dados rotulados manualmente, também por especialistas da empresa.

Para conduzir a avaliação quantitativa foi utilizado o Erro Quadrático Médio (EQM) (MORETTIN; BUSSAB, 2017), entre a prioridade estimada pelo método proposto e a prioridade definida por um especialista da empresa.

Mais detalhes sobre a avaliação do método proposto e do do experimento realizado para tal, são apresentados no Capítulo 4.

4 RESULTADOS EXPERIMENTAIS

Para que fosse possível mensurar a acurácia da abordagem de priorização automática de atividades proposta nesse trabalho, o método apresentado no Capítulo metodo foi implementado na linguagem de programação Python (o código fonte é disponibilizado em: https://github.com/maiconsilveira/Trabalho-de-Conclus-o-de-Curso/tree/master). Após a implementação do método, foi conduzido um estudo de caso com a base de dados disponibilizada pela empresa Visual Software, composta por 69535 registros de atividades de desenvolvimento de software.

Do conjunto de atividade fornecidas pela Visual Software, 100 atividades (referentes as atividades do mês de Janeiro de 2018) foram utilizadas para produzir o conjunto de treinamento ($A_{treinamento}$). Já quanto ao conjunto de teste (A_{teste}), varias variações desse conjunto foram criadas para permitir avaliar o método proposto de duas formas: i) Qualitativa e ii) Quantitativa.

Para avaliação qualitativa, o conjunto de teste, foi composto por um conjunto de 100 atividades de desenvolvimento de software, com exceção das atividades que foram utilizadas para criar o conjunto de treinamento. Os resultados foram apresentados a dois especialistas da empresa, que consideraram a ordenação das atividades adequada e condizente com o processo subjetivo realizado atualmente, entendendo assim como algo aplicável para a empresa.

Já para avaliação quantitativa, foi mensurado o EQM, considerando subconjuntos de A_{teste}, denotados por $A_{testex} \in A_{teste}$, tal que cada subconjunto A_{testex} é composto por 100 atividades.

Cada um dos subconjuntos A_{testex} teve suas atividades priorizadas por um especialista da empresa e também pelo método automático de priorização proposto, sendo possível assim calcular o EQM da priorização, usado para definir a acurácia do método.

Os experimentos foram feito considerando 5 configurações diferentes para o conjunto A_{testex}, cada um com 100 atividades distintas. Destaca-se que o método foi aplicado

considerando as 100 atividades de cada conjunto A_{testex}, mas também as primeiras 25, 35, 50 e 75 atividades, permitindo visualizar o nível de aumento do EQM, em relação ao aumento do tamanho do conjunto de atividades. A média dos resultados obtidos com esse experimentos é apresentada na Figura 3.

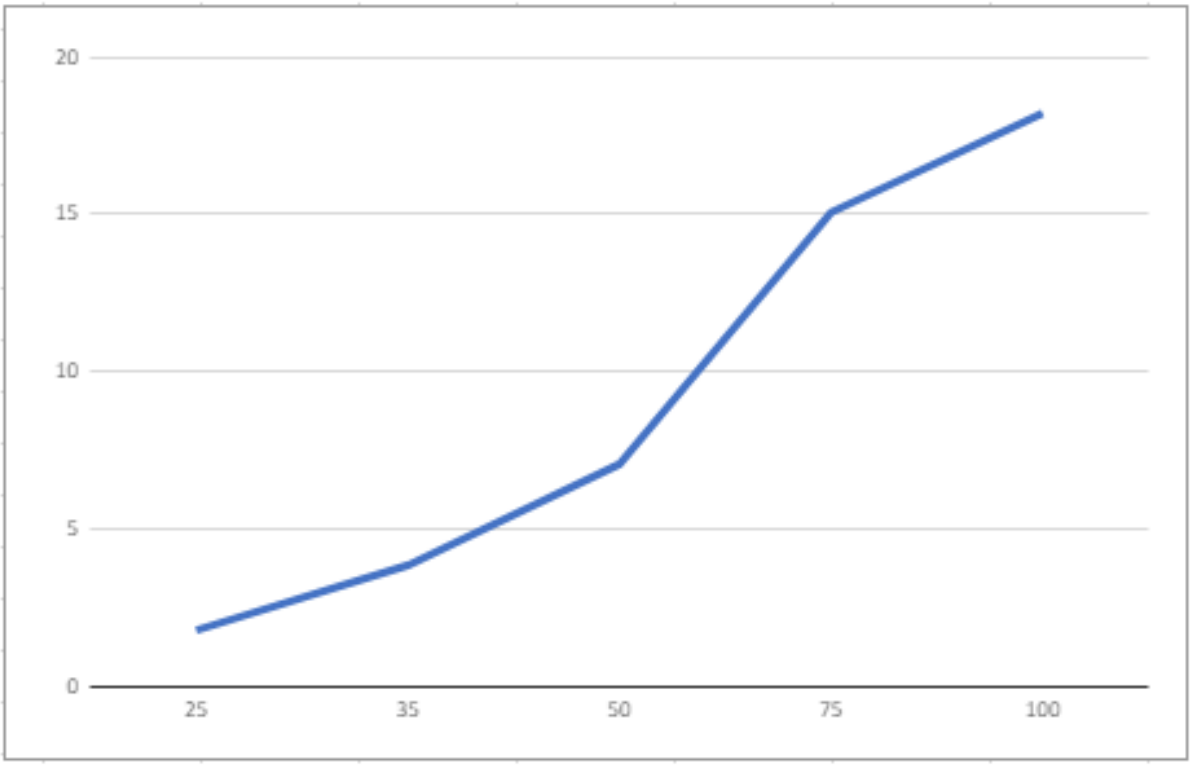

Figura 3: Erro Médio Quadrático do método proposto
Fonte: Autoria Própria.

5 CONCLUSÕES

A priorização correta das atividades de desenvolvimento de software é fundamental para o cumprimento das estratégias das empresas deste ramo e também para melhorar o desempenho das equipes de desenvolvimento. Porém, muitas vezes essa atividade acontece de maneira subjetiva, já que a fila de priorização é normalmente determinada por algum especialista que utiliza sua experiência para realizar essa ordenação. A ausência dessa figura na equipe ou a ausência de uma metodologia bem definida faz com que os desenvolvedores muitas vezes não tenham bons resultados de ordenação, impactando na qualidade do processo de desenvolvimento utilizado.

Visando contribuir com esse problema, esse trabalho apresentou método priorização automático de atividades de desenvolvimento de software, utilizando uma RNA e um método de ordenação.

O método proposto foi avaliado de forma qualitativa e quantitativa, por meio de estudos de casos com dados fornecidos pela empresa Visual Software. Os resultados do experimento comprovaram a acurácia do método proposto.

Espera-se que o método proposto possa ser utilizado ou servir como base para criação de métodos de priorização automática que possam ser utilizados em outras empresas.

REFERÊNCIAS

ACHIMUGU, P. et al. A systematic literature review of software requirements prioritization research. **Information and software technology**, Elsevier, v. 56, n. 6, p. 568–585, 2014.

AGUIAR, E. et al. Who, when, and why: a machine learning approach to prioritizing students at risk of not graduating high school on time. In: ACM. **Proceedings of the Fifth International Conference on Learning Analytics And Knowledge**. [S.l.], 2015. p. 93–102.

AUGER, N.; NICAUD, C.; PIVOTEAU, C. Merge strategies: from merge sort to timsort. 2015.

BATISTA, G. E. d. A. P. et al. **Pré-processamento de dados em aprendizado de máquina supervisionado**. Tese (Doutorado) — Universidade de São Paulo, 2003.

BEAL, F.; BASSI, P.; PARAISO, E. Developer modelling using software quality metrics and machine learning. **International Conference on Enterprise Information Systems**, p. 424–432, 2017.

BRAGA, A. d. P.; CARVALHO, A.; LUDERMIR, T. B. **Redes neurais artificiais: teoria e aplicações**. [S.l.]: Livros Técnicos e Científicos Rio de Janeiro, 2000.

CHEN, J. et al. Learning to prioritize test programs for compiler testing. In: IEEE PRESS. **Proceedings of the 39th International Conference on Software Engineering**. [S.l.], 2017. p. 700–711.

CLARKE, P.; O'CONNOR, R. V.; LEAVY, B. A complexity theory viewpoint on the software development process and situational context. In: ACM. **Proceedings of the International Conference on Software and Systems Process**. [S.l.], 2016. p. 86–90.

CLARKE, P. et al. Exploring the relationship between software process adaptive capability and organisational performance. **IEEE Transactions on Software Engineering**, IEEE, v. 41, n. 12, p. 1169–1183, 2015.

COLEMAN, G.; O'CONNOR, R. Investigating software process in practice: A grounded theory perspective. **Journal of Systems and Software**, Elsevier, v. 81, n. 5, p. 772–784, 2008.

CORMEN, T. H. et al. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, SciELO Brasil, v. 35, n. 1, 2006.

GREER, D.; RUHE, G. Software release planning: an evolutionary and iterative approach. **Information and software technology**, Elsevier, v. 46, n. 4, p. 243–253, 2004.

HAYKIN, S. **Redes neurais: princípios e prática**. [S.l.]: Bookman Editora, 2007.

KANWAL, J.; MAQBOOL, O. Managing open bug repositories through bug report prioritization using svms. In: **Proceedings of the International Conference on Open-Source Systems and Technologies, Lahore, Pakistan**. [S.l.: s.n.], 2010.

KARLSSON, J.; WOHLIN, C.; REGNELL, B. An evaluation of methods for prioritizing software requirements. **Information and software technology**, Elsevier, v. 39, n. 14-15, p. 939–947, 1998.

LACHMANN, R. et al. System-level test case prioritization using machine learning. In: IEEE. **Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on**. [S.l.], 2016. p. 361–368.

LUGER, G. F. **Artificial intelligence:**. [S.l.]: Pearson education, 2013.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas Inteligentes-Fundamentos e Aplicações**, v. 1, n. 1, p. 32, 2003.

MORETTIN, P. A.; BUSSAB, W. O. **Estatística básica**. [S.l.]: Editora Saraiva, 2017.

MYLAVARAPU, G.; THOMAS, J. P. A multi-task machine learning approach for comorbid patient prioritization. In: IEEE. **Big Data (Big Data), 2017 IEEE International Conference on**. [S.l.], 2017. p. 3877–3881.

PAETZOLD, G.; SPECIA, L. Lexical simplification with neural ranking. In: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers**. [S.l.: s.n.], 2017. v. 2, p. 34–40.

PARRA, E. et al. A methodology for the classification of quality of requirements using machine learning techniques. **Information and Software Technology**, Elsevier, v. 67, p. 180–195, 2015.

PERINI, A.; SUSI, A.; AVESANI, P. A machine learning approach to software requirements prioritization. **IEEE Transactions on Software Engineering**, IEEE, v. 39, n. 4, p. 445–461, 2013.

RIBEIRO, S. A. et al. A síndrome do deadline: Origem, causas e implicações no processo de desenvolvimento de software. **iSys-Revista Brasileira de Sistemas de Informação**, v. 10, n. 2, p. 30–47, 2017.

SANTOS, R. M. **Descoberta de Conhecimento em Bases de Dados para Direcionamento de Tarefas de Teste em Empresas de Software**. Monografia (Gradução) — Faculdade Educacional de Dois Vizinhos, Dois Vizinhos, 2017.

SHAO, F. et al. Drank: A semi-automated requirements prioritization method based on preferences and dependencies. **Journal of Systems and Software**, Elsevier, v. 126, p. 141–156, 2017.

SILVEIRA, M. J. et al. Uma técnica de priorização de atividades de desenvolvimento de software baseada em árvore de decisão. In: **Congresso de Ciência e Tecnologia da UTFPR Câmpus Dois Vizinhos**. [S.l.: s.n.], 2017. p. 460–462.

SPIEKER, H. et al. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In: ACM. **Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis**. [S.l.], 2017. p. 12–22.

TONELLA, P.; SUSI, A.; PALMA, F. Interactive requirements prioritization using a genetic algorithm. **Information and software technology**, Elsevier, v. 55, n. 1, p. 173–187, 2013.

UDDIN, J. et al. A survey on bug prioritization. **Artificial Intelligence Review**, Springer, v. 47, n. 2, p. 145–180, 2017.

WEISS, S. M.; KULIKOWSKI, C. A. **Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems**. [S.l.]: Morgan Kaufmann Publishers Inc., 1991.

YANG, G.; ZHANG, T.; LEE, B. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: IEEE. **2014 IEEE 38th Annual Computer Software and Applications Conference (COMPSAC)**. [S.l.], 2014. p. 97–106.

YILMAZ, M.; O'CONNOR, R. V.; CLARKE, P. A systematic approach to the comparison of roles in the software development processes. In: SPRINGER. **International Conference on Software Process Improvement and Capability Determination**. [S.l.], 2012. p. 198–209.

APÊNDICE A – PROCESSO DE DESENVOLVIMENTO

Essa Apêndice visa apresentar o processo de desenvolvimento de software utilizado na empresa Visual Software. Com esse objetivo, essa seção encontra-se organizada da seguinte forma: A seção A.1 apresenta a empresa. Detalhes do processo de desenvolvimento são apresentados na subseção A.2.

A.1 DESCRIÇÃO DA EMPRESA

As técnicas de priorização automática de atividades, propostas nesse projeto, serão validadas com base no processo de desenvolvimento da Visual Software, que é uma empresa de desenvolvimentos de software, da cidade de Quedas do Iguaçu-PR, que está a mais de dezesseis anos no mercado.

Atualmente a empresa, possuí mais de dois mil clientes ativos em quase todas as unidades federativas do país e cerca de cinquenta colaboradores diretos, tendo como foco as áreas de varejo, distribuidoras, supermercados, laticínios e indústrias.

A.2 ATIVIDADES DO PROCESSO DE DESENVOLVIMENTO

A empresa conta com uma lista de atividades a serem desenvolvidas, chamada de *backlog*, na qual novos itens podem ser adicionados a qualquer momento.

Para o desenvolvimento dessas atividades, utilizam-se ciclos mensais (iterações), durante os quais são desenvolvidas as atividades encaminhadas pelo coordenador de desenvolvimento e a equipe de testes, sendo que ao termino de cada iteração é realizada uma entrega formal.

Como novas atividades podem ser adicionadas a qualquer momento, se torna inviável re-priorizar o *backlog* de atividades sempre que tal fato ocorre, já que para isso seria necessário revisar toda a lista de tarefas, e ordena-las manualmente, e isso é necessário já que novas atividades podem possuir maior prioridade que as já inclusas anteriormente.

As atividades são separadas em 3 diferentes categorias, que no processo de desenvolvimento seguirão fluxos diferentes, mas em determinado momento estarão todas disponíveis no mesmo *backlog* de atividades, são elas:

- **BUG**: Corresponde a todas as falhas manifestadas nos softwares.
- **Melhoria**: Corresponde a novas funcionalidades nos softwares que geram benefícios a todos os clientes que o utilizarem.
- **Customização**: Corresponde a novas funcionalidades que possuam muitas particularidades de determinado cliente, tornando o recurso útil apenas a ele. Nessa categoria é avaliada a possibilidade de realizar o desenvolvimento mediante o pagamento do cliente solicitante.

Destaca-se que, os resultados de priorização obtidos com a técnica proposta nesse trabalho, serão utilizados em duas das atividades apresentadas: i) Desenvolver os Requisitos; e ii) Corrigir Defeitos.

O processo geral e todas as etapas do processo de desenvolvimento de software da empresa são apresentados detalhadamente a seguir:

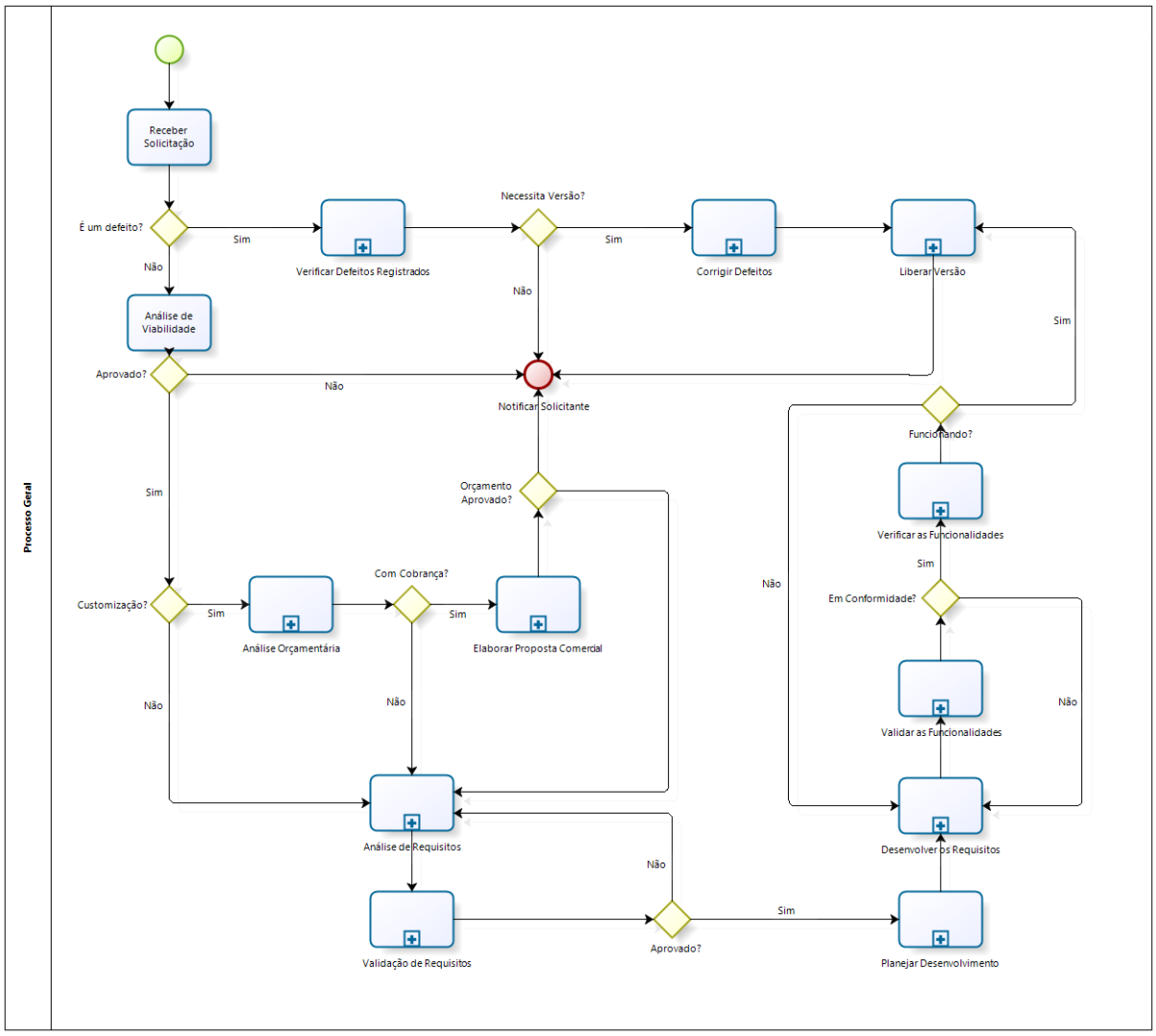

Figura 4: Processo Geral.

Fonte: Autoria Própria.

- **Análise de Viabilidade**: Nesta etapa, o analista de requisitos verificar se as necessidades apresentadas pelo cliente podem ser resolvidas com alguma funcionalidade já existente, caso não seja possível, é analisada viabilidade técnica, resultando na aprovação ou recusa da solicitação. A Figura 5 ilustra detalhes do processo de análise de viabilidade.

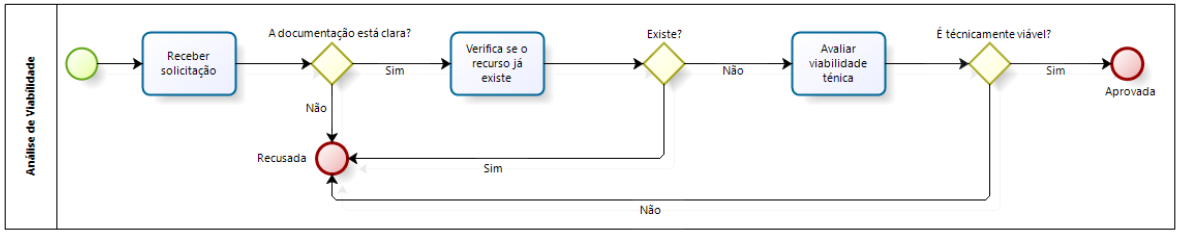

Figura 5: Processo de Análise de Viabilidade.
Fonte: Autoria Própria.

•**Analise Orçamentária**: Nesta etapa o coordenador de desenvolvimento identifica as particularidades de cada nova funcionalidade. Caso se trate se algo que não irá gerar valor para outros clientes, é realizando juntamente com a equipe de desenvolvimento uma estimativa de tempo para ser desenvolvido a funcionalidade, os custos são repassado ao departamento comercial para confecção da proposta comercial. A Figura 6 ilustra detalhes do processo de analise orçamentária.

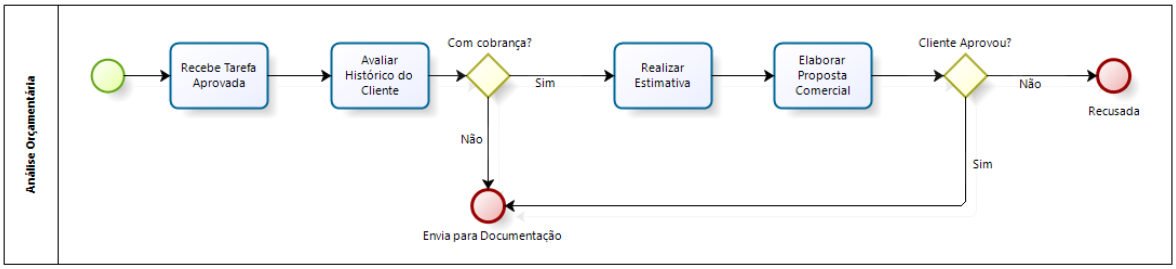

Figura 6: Processo de Análise Orçamentária.
Fonte: Autoria Própria.

•**Elaborar Proposta Comercial**: Nesta etapa o agente comercial responsável pela região do cliente elabora um documento de proposta comercial para a alteração do software, com base no custo repassado pelo coordenador de desenvolvimento de sistemas. Após, o documento é encaminhado ao cliente, caso seja aprovado ou recusado a tarefa é encaminhada aos seus respectivos status. A Figura 7 ilustra detalhes do processo de elaborar proposta comercial.

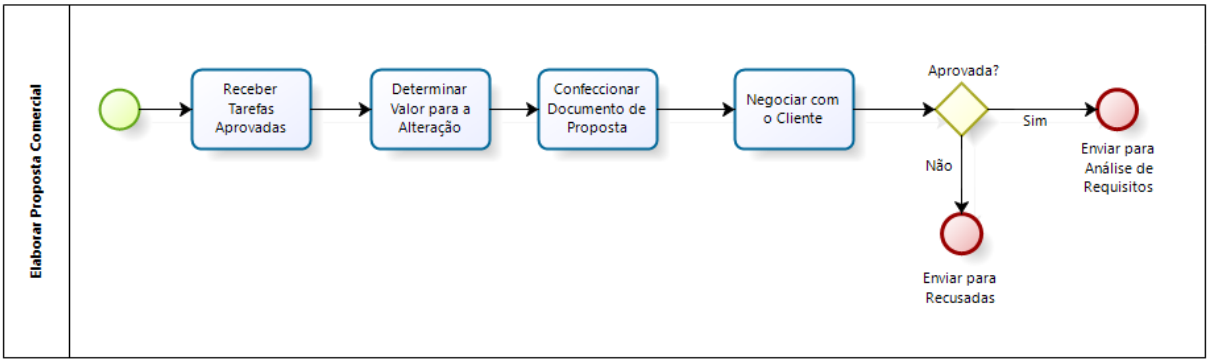

Figura 7: Processo de Elaborar Proposta Comercial.
Fonte: Autoria Própria.

- **Análise de Requisitos**: Nesta etapa o analista de requisitos realiza uma entrevista com o solicitante do recurso, levantando todos os requisitos necessários para solucionar a necessidade do cliente. Após, é realizado uma documentação textual, acompanhada de protótipos elaborados pela ferramenta Balsamiq [1]. A Figura 8 ilustra detalhes do processo de análise de requisitos.

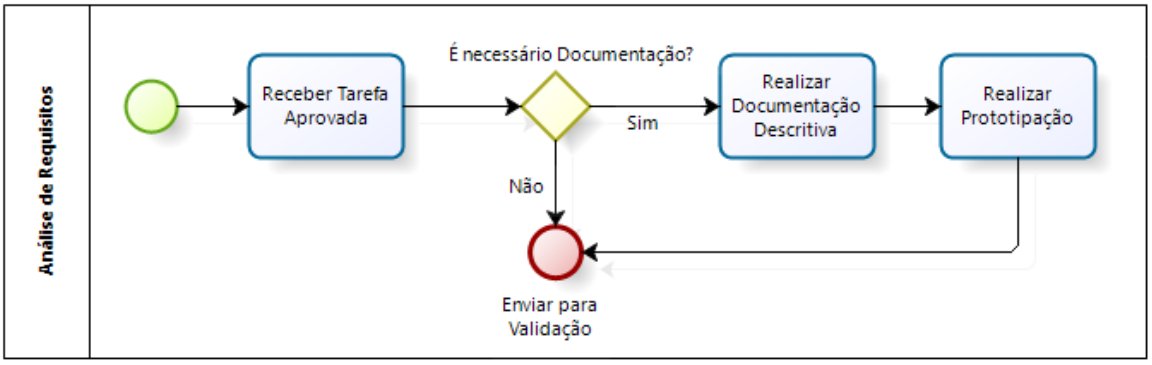

Figura 8: Processo de Análise de Requisitos.
Fonte: Autoria Própria.

- **Validação de Requisitos**: Esta etapa contempla duas atividades, a primeira é realizada por um programador, que verifica se é tecnicamente viável desenvolver o recurso proposto pelo analisa de requisitos, levando em consideração apenas aspectos técnicos. A segunda atividade é apresentar e validar os protótipos com o solicitante do recurso. A Figura 9 ilustra detalhes do processo de validação de requisitos.

[1] https://balsamiq.com/

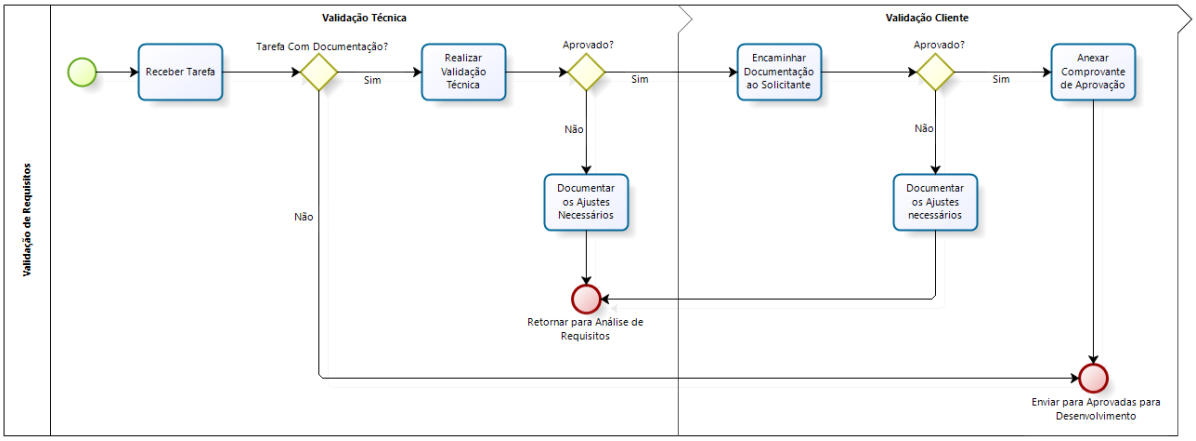

Figura 9: Processo de Validação de Requisitos.
Fonte: Autoria Própria.

- **Planejar Desenvolvimento**: Nesta etapa o coordenador de desenvolvimento avalia se existem inconsistências na documentação das tarefas. Caso não haja inconsistências, as atividades são encaminhadas para o *backlog* de desenvolvimento. A Figura 10 ilustra detalhes do processo de planejar desenvolvimento.

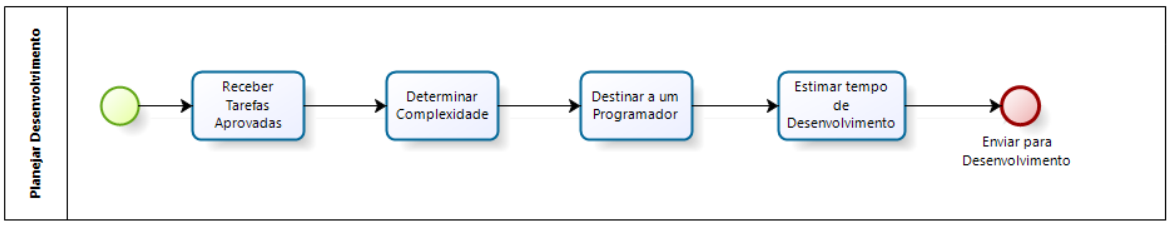

Figura 10: Processo de Planejar Desenvolvimento.
Fonte: Autoria Própria.

- **Desenvolver os Requisitos**: Nesta etapa o programador seleciona uma das tarefas disponíveis para serem desenvolvidas, avalia a documentação de requisitos e realiza a implementação no software correspondente. A Figura 11 ilustra detalhes do processo de desenvolver os requisitos.

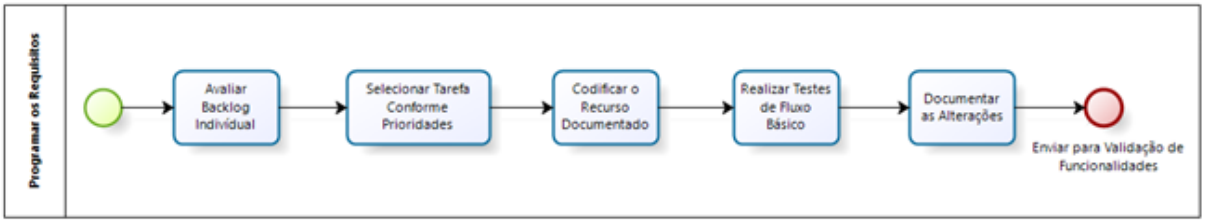

Figura 11: Processo de Desenvolver os Requisitos.
Fonte: Autoria Própria.

- **Validar e Verificar as Funcionalidades**: Nesta etapa os testadores recebem a versão com as alterações de determinado ciclo e realizam todas as atividades de teste, realizando as devidas documentações e notificações. A Figura 12 ilustra detalhes do processo de verificar as funcionalidades.

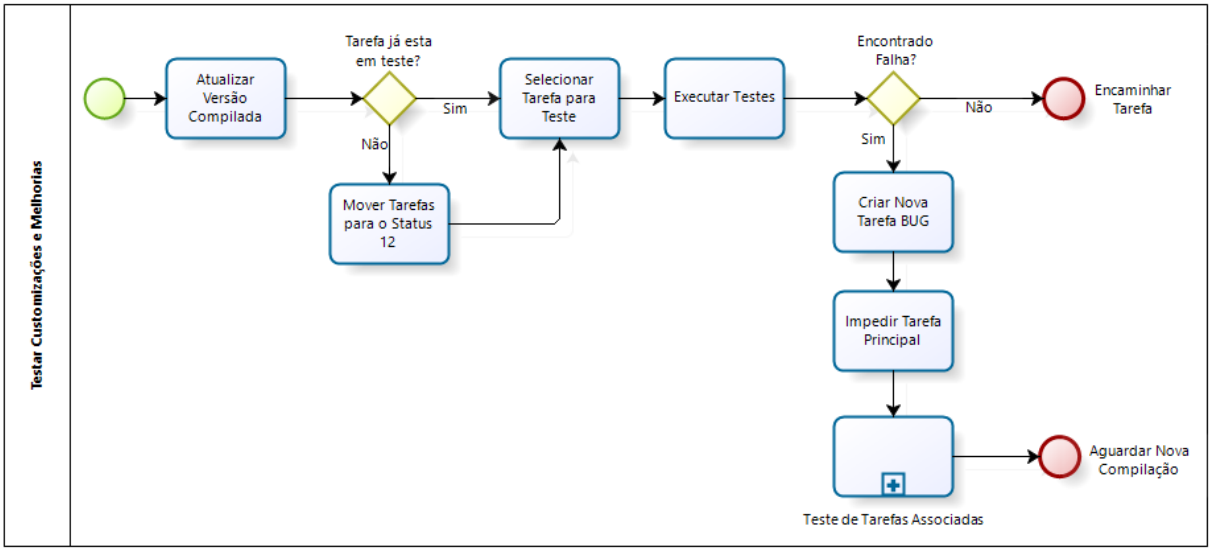

Figura 12: Processo de Verificar e Validar as Funcionalidades.
Fonte: Autoria Própria.

- **Verificar Defeitos Registrados**: Nesta etapa os testadores recebem os relatos de falhas que ocorreram nos clientes, realizam a montagem do cenário, documentam os passos para ocasionar tal falha e encaminham a tarefa para desenvolvimento. A Figura 13 ilustra detalhes do processo de verificar defeitos registrados.

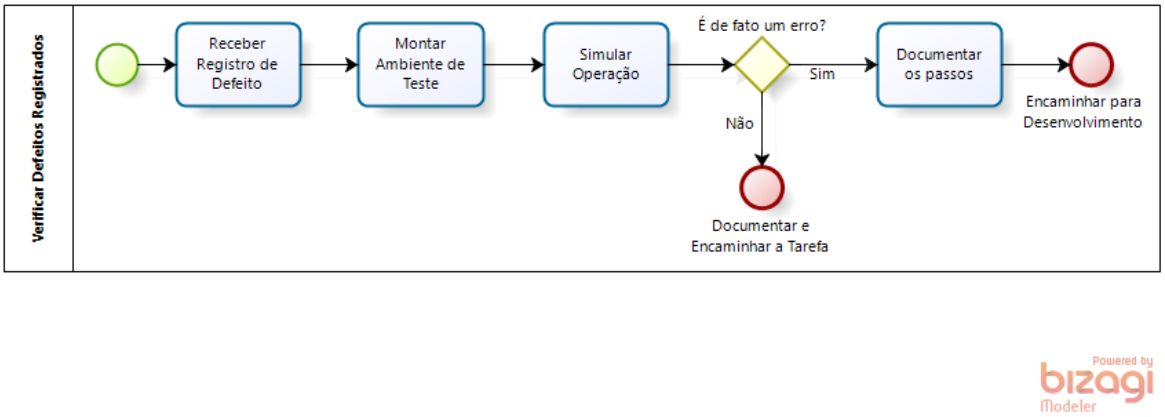

Figura 13: Processo de Verificar Defeitos Registrados.
Fonte: Autoria Própria.

•**Corrigir Defeitos**: Nesta etapa os programadores selecionam as tarefas do tipo *BUG* que ficam listadas juntamente com as customizações e melhorias, selecionam a tarefa do tipo *BUG* conforme a prioridade e realiza a correção no software correspondente. A Figura 14 ilustra detalhes do processo de corrigir defeitos.

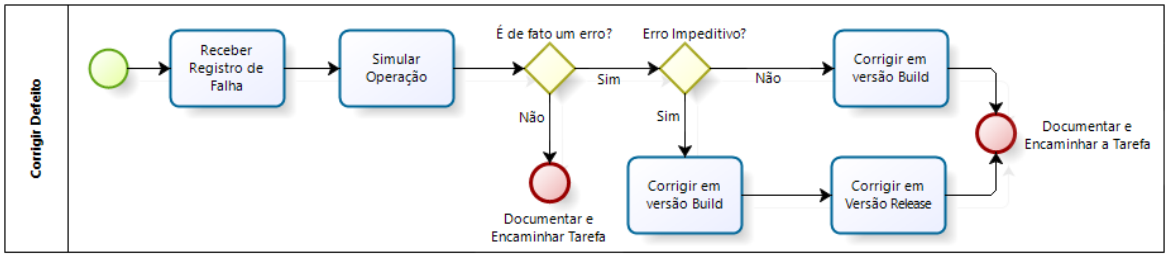

Figura 14: Processo de Corrigir Defeitos.
Fonte: Autoria Própria.

•**Liberar Versão**: Nesta etapa o coordenador de desenvolvimento avalia o andamento do ciclo de desenvolvimento e quando necessário lança as versões para ambiente de produção. A Figura 15 ilustra detalhes do processo de liberar versão.

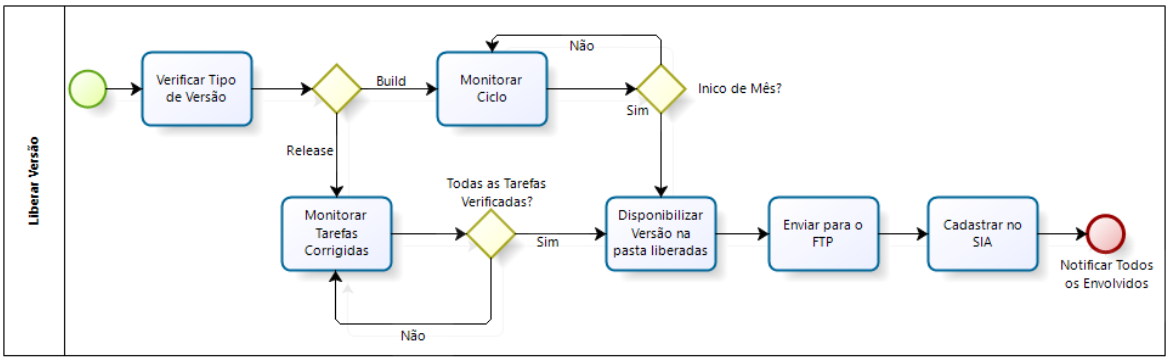

Figura 15: Processo de Liberar Versão.

Fonte: Autoria Própria.