UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
DEPARTAMENTO DE ESTATÍSTICA E INFORMÁTICA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ALEX ROGÉRIO DA SILVA CALADO

UM SISTEMA DE APOIO À DECISÃO PARA PRIORIZAÇÃO E ESTRUTURAÇÃO DE HISTÓRIAS DE USUÁRIOS: UM SUPORTE PARA EQUIPES ÁGEIS

TRABALHO DE CONCLUSÃO DE CURSO

Recife
agosto de 2018

ALEX ROGÉRIO DA SILVA CALADO

UM SISTEMA DE APOIO À DECISÃO PARA PRIORIZAÇÃO E ESTRUTURAÇÃO DE HISTÓRIAS DE USUÁRIOS: UM SUPORTE PARA EQUIPES ÁGEIS

Monografia apresentada ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Cícero Garrozi
Coorientadora: Suzana Sampaio

Recife
agosto de 2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Biblioteca Central, Recife-PE, Brasil

C141s Calado, Alex Rogério da Silva.
Um sistema de apoio à decisão para priorização e estruturação de histórias de usuários: um suporte para equipes ágeis / Alex Rogério da Silva Calado. – Recife, 2018.
66 f.: il.

Orientador(a): Cícero Garrozi.
Coorientador(a): Suzana Sampaio.
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal Rural de Pernambuco, Departamento de Estatística e Informática, Recife, BR-PE, 2018.
Inclui referências e apêndice(s).

1. Engenharia de software 2. Aprendizado do computador 3. Sistema de suporte de decisão 4. Priorização 5. Interfaces de usuário (Sistema de computador) 6. Scrum I. Garrozi, Cícero, orient. II. Sampaio, Suzana, coorient. III. Título

CDD 310

À minha mãe e à minha tia Vilma por todo esforço e dedicação na construção da minha educação.

Agradecimentos

A todos os familiares e amigos que incentivaram e contribuíram para minha formação acadêmica.

A todos os meus professores que mostraram paixão e entusiasmo em ensinar-me o conhecimento que decidiram abraçar e perpetuar.

À minha coorientadora pelo seu suporte e por ser uma inspiração.

Resumo

Apesar dos avanços obtidos na Engenharia de Software com os métodos ágeis o mercado ainda apresenta taxas de sucesso em projetos insatisfatórias. Ao longo de um projeto de software, mudanças de requisitos e prioridades são inevitáveis e recorrentes. Esse é um fator importante para empresas de desenvolvimento de software principalmente as de pequeno porte que possuem recursos limitados. O uso de técnicas ágeis como Scrum e Histórias de Usuário (US) beneficiam as empresas e as tornam mais competitivas. Um dos problemas enfrentados com requisitos ágeis é obter uma priorização de US segura de acordo com o valor de negócio dado pelo cliente em consenso com especificações técnicas. Esse trabalho se propõe em apresentar um ensaio que serve de base para construção de uma ferramenta de apoio à decisão na priorização das US. Para tanto foi considerado um projeto de software como uma organização temporária e sugeridas métricas que melhor se adequem às necessidades de pequenas equipes evitando retrabalho, aumento de prazos e custos, sem desconsiderar a satisfação do cliente nas entregas interativas. Baseado no modelo de levantamento de requisitos Volere e na revisão de literatura foi proposto a adoção de cinco métricas a serem consideradas na priorização de Histórias de Usuários: a satisfação do cliente em receber a US e a insatisfação do cliente em não receber a US, substituindo o tradicional valor de negócio; a já usual complexidade; necessidade de aprendizagem da equipe e; riscos de software. Usando formulários online foram captados dados sobre essas métricas e a priorização dada a cada US em projetos de software que usam o método ágil Scrum. Árvore de Decisão foi a sugestão proposta para predição de priorização de US por possuir visualização prática e interpretação mais intuitiva, facilitando a aceitação como método de apoio a decisão de profissionais da área. Apesar do baixo volume da base de dados os resultados obtidos através da ferramenta Weka, como a Curva *ROC* e a precisão, mostraram-se satisfatórios sem tendências de predição e com bons índices de acertos, após ajustes dos algoritmos e da base de dados para evitar *overfitting* e *underfitting*.

Palavras-chave: Engenharia de Software, Aprendizagem de Máquina, Metodologia Ágil, Árvores de Decisão, Priorização, Histórias de Usuário, Scrum.

Abstract

Despite the advances obtained in the Software Engineering with agile methods the market still shows successful rates in unsatisfactory projects. Throughout a software project, changing requirement and priority are unavoidable and recurrent. This is an important factor to the software development businesses mainly small ones with limited resources. The usage of agile techniques such as Scrum and User Stories (US) benefits the businesses and make them more competitive. One of the problems faced with agile requirements is to obtain a prioritization of secure US according to the business value given by the client in agreement with technical specifications. This paper proposes to present an essay that serves as basis to the construction of a decision support tool to the US prioritization decisions. For such was considered a software project as a temporary organization and suggested metrics that best suits the needs of small teams avoiding reworks, the increasing in the deadlines and costs, without disregarding the client satisfaction in interactive deliveries. Based on the Volere requirements template and in the literature review it was proposed the adoption of five metrics to be considered in the prioritization of the User Stories: the client satisfaction in receiving the US and the client dissatisfaction in not receiving it; replacing the traditional value of business; the so usual complexity; the necessity of team learning and; software risks. Using online forms were captured data about these metrics and the prioritization given to each US in software projects that uses the agile Scrum method. Decision Tree was the proposed suggestion for predicting the US prioritization for having a practical visualization and a more intuitive interpretation, facilitating the acceptance as a decision method support for professionals involved in the area. In spite of the low databases volume the results obtained through the Weka tool, like the precision and the ROC curve, were satisfactory without prediction tendencies and with good indexes of correctness, after the algorithms and databases adjustments avoiding overfitting and underfitting.

Key words: Software engineering, Machine Learning, Agile Methodology, Tree of Decision, Prioritization, User Stories, Scrum.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Apresentação

Para tornar os processos mais simples e suscetíveis às mudanças muito se debate sobre métodos ágeis e sobre os desafios que podem ser enfrentados no decorrer do seu processo de implementação. Os métodos ágeis, mais recentes que os métodos tradicionais, valorizam a adaptação rápida às mudanças, interação frequente com o cliente e prefere indivíduos e suas interações à documentação e ferramentas (MANIFESTO ÁGIL, 2001). As metodologias ágeis apresentam soluções práticas em curto intervalos de tempo focando mais no produto do que no processo, como é o caso do método Scrum, um dos métodos de gerenciamento ágil de projetos mais usados. O Scrum é caracterizado por flexibilidade nos resultados e prazos, times pequenos, revisões frequentes e colaboração (SHWABER; SUTHERLAND, 2014).

Apesar das melhorias no que se diz respeito ao desenvolvimento de produtos de software nos últimos anos o mercado ainda apresenta taxas de eficiência insatisfatórias. O documento *Chaos Report* (JOHNSON, 2015) relatou que pequenos projetos possuíam uma taxa de sucesso de 41%, projetos de tamanho moderado de 24,49% e projetos grandes de apenas 8%. Este mesmo estudo aponta que as abordagens ágeis obtiveram, no geral, 39% de sucesso uma taxa 3,5 vezes maior em comparação com os métodos tradicionais.

Ao longo de um projeto de software, mudanças de requisitos e prioridades são inevitáveis e recorrentes (SERRA, 2015). Hoje em dia os ambientes de desenvolvimento de softwares são marcados por sua mutabilidade, adaptação e aperfeiçoamento no decorrer do processo de desenvolvimento (ROBERTO; BARBOSA, 2016). Este é um empecilho a mais no mercado, principalmente para as pequenas empresas que possuem recursos limitados. Os métodos ágeis, como o Scrum, mostram-se ser uma implementação eficiente por ser um processo de desenvolvimento software incremental em ambientes complexos, onde os requisitos não são claros ou mudam com muita frequência. O Scrum é ideal para equipes pequenas e tem por objetivo aperfeiçoar a previsibilidade e controlar os riscos de um projeto (SILVA; SOUZA; CAMARGO, 2013). O uso de métodos ágeis em pequenas empresas traz vários benefícios fundamentais para que elas se tornem competitivas no mercado. Para tanto é necessário que suas equipes estejam familiarizadas com os métodos e suas ferramentas (SANTOS et al., 2016).

Uma dessas ferramentas ágeis é a US (User Stories ou Histórias de Usuário), descrições curtas de funcionalidade de software, geradas a partir da perspectiva de um usuário, amplamente difundidas no meio ágil para levantamento de requisitos. O

usuário de negócio e a equipe de desenvolvimento decidem quais requisitos terão maior prioridade de implementação e quais serão produzidas em seguida (LONGO; SILVA, 2014). Um grande desafio para as equipes ágeis é calibrar a prioridade de entrega do cliente com as necessidades de desenvolvimento da equipe (GAIKWAD; JOEG, 2016).

Nos métodos ágeis os requisitos são desenvolvidos de forma incremental de acordo com a priorização dinâmica definida pelo valor de negócio dado pelo cliente. Um dos principais problemas relacionados com requisitos ágeis é o consenso na negociação ou priorização das histórias de usuário, ainda mais recorrente quando existe mais de um cliente. Mesmo a documentação mínima sendo uma característica dos métodos ágeis, em ambientes de desenvolvimento reais desenvolvedores sentem falta de requisitos documentados de maneira mais completa. O que permite uma visão técnica mais ampla do projeto que leve a uma priorização de requisitos mais segura (JAQUEIRA et al., 2013)

Estudos de caso como o (MELIA; AUWAERTER, 2016) apontam várias dificuldades na adoção prática de métodos ágeis no mercado. Um desses desafios é a negligência na descrição dos requisitos não-funcionais, que podem ser mal definidos ou ignorados quando se usa apenas o valor de negócio para o cliente como referência de prioridade. O cliente muitas vezes está preocupado com funcionalidades essenciais e acaba por não atender requisitos de segurança e eficiência. Além disso, foi evidenciado que a redefinição de prioridades contínua, quando não praticada com cautela, leva à instabilidade do software.

Determinados estudos com a escrita de US como (GAIKWAD; JOEG, 2016)mostram que, na prática, ainda existem muitas dificuldades em se escrever uma US eficaz. Nele destaca-se o fato da maioria dos envolvidos na pesquisa não seguirem boas práticas de escrita de histórias de usuário e reconhecerem que uma ferramenta que auxilie na modelagem de requisitos seria útil. Além disso, erros de priorização e US ambíguas são comuns e representam boa parte dos problemas encontrados.

Esse trabalho se propõe em apresentar uma ferramenta que apoie o time de desenvolvimento sugerindo a prioridade das histórias para um ciclo de trabalho do ponto de vista mais pragmático e não só baseado na prioridade de desejo do cliente. Para tanto serão abordadas e definidas métricas que melhor se adequem às necessidades de pequenas equipes buscando um ciclo de trabalho fluido, evitando ociosidade, retrabalho, aumento de prazos e custos, sem desconsiderar a satisfação do cliente nas entregas interativas.

Essas métricas são o quão satisfeito estará o cliente em receber a estória (ROBERTSON; ROBERTSON, 2012); o quão insatisfeito se não receber (ROBERTSON; ROBERTSON, 2012); complexidade (USMAN et al., 2014); riscos técnicos (SOUZA;

MOURA, 2010)e; necessidade de aprendizado da equipe (SANTOS JUNIOR; BISPO; MOURA, 2007). A ferramenta se baseia no modelo de levantamento de requisitos Volere, conceituada no mercado.

Optou-se pelo *Volere* como modelo para levantamento de requisitos, porque como citado pelos seus autores, James e Suzanne Robertson, é uma técnica que já foi adotada por milhares de organizações ao redor do mundo e ele é baseado no princípio da sumarização de experiências com desenvolvimento de software para criar um modelo simples e completo (SILVA; BONIN; PALUDO, 2005).

1.2 Justificativa

Em 2017 o Brasil abrigava cerca de 5.138 empresas que atuavam no desenvolvimento e produção de software, detnre elas 95,5% podem ser classificadas como micro e pequenas empresas. Neste mesmo ano o mercado de software movimentou 8,183 bilhões de dólaresno país e, mais 174 milhões de doláres em exportação.

Como aponta (BASSI FILHO, 2008) o foco dos métodos ágeis são equipes pequenas, por se basearem-se num processo iterativo e por se caracterizar como suscetíveis a mudanças durante todo o processo. As pequenas empresas representam, em média, 85% do setor de desenvolvimento de software e vários casos de sucesso foram obtidos de pequenas empresas com a implantação do desenvolvimento ágil. Entretanto elas sofrem com recursos limitados, baixos investimentos e mudanças constantes no ciclo de desenvolvimento do produto (SANTOS et al., 2016). Tendo em vista essas informações os métodos ágeis são uma ótima opção no gerenciamento de empresas de desenvolvimento de software.

Scrum e suas ferramentas são apontados como um método ágil estratégico e útil para pequenas empresas de TI com diversos pontos positivos como o crescimento da satisfação de clientes; melhoria na comunicação e aumento da colaboração entre envolvidos nos projetos; aumento da motivação da equipe de desenvolvimento; melhoria na qualidade do produto produzido; dedução dos custos de produção; adição de produtividade da equipe de desenvolvimento; redução no tempo e do risco em projetos de desenvolvimento de novos produtos (CARVALHO; MELLO, 2012).

Empresas desenvolvedoras de software, em geral, buscam a satisfação do usuário conseguida através de um produto compatível, com qualidade e entrega dentro do orçamento e do prazo previstos. Isso tem um custo que engloba tudo o que é necessário na busca pela qualidade bem como os custos causados pela falta de qualidade. Alguns desses custos são as atividades técnicas adicionais para desenvolver modelos completos de requisitos e de projeto e, o retrabalho causado por falhas durante o projeto (PRESSMAN, 2011).

Os custos para se descobrir e reparar um erro aumentam drasticamente à medida que que se avança entre os processos das etapas de planejamento e desenvolvimento. O gasto médio da indústria de software para correção de um erro durante o desenvolvimento inicial do código é de US$ 977 por erro. Para corrigir o mesmo erro na fase de teste (erro interno) o custo é de US$ 7.136 e se identificado pelo usuário (falha externa) é em torno de US$ 14.102 por erro (PERES, 2016).

Gerenciamento proativo de riscos é considerado uma estratégia inteligente no que se refere a qualidade de software. Riscos técnicos que põe em risco a qualidade e a data de entrega dos softwares em produção, como é o caso de ambiguidades de especificações e incerteza técnica, devem ser evitados ou tratados previamente. Esses riscos devem ser identificados o mais cedo possível nas etapas de planejamento e levantamento de requisitos (PRESSMAN, 2011).

As métricas ágeis usadas hoje em dia contam com uma variedade de deficiências e desvantagens. Um problema comum no atual meio ágil, por exemplo, é que eles tendem a misturar métricas de projeto e processo. As abordagens e regras sugeridas são fragmentadas em diferentes autores levando a confusão e não há uma apresentação clara de um conjunto de medidas abrangentes que claramente distingue e define as métricas de projeto, processo e produto (RAM, 2013).

Volere, um dos modelos adotado por milhares de organizações para Especificações de Requisitos, disponibiliza seções para cada um dos tipos de requisitos, adequados aos atuais sistemas aplicativos de computadores. Consiste em um modelo adaptável ao seu processo, como uma ferramenta de coleta de requisitos. Nele existem critérios de ajustes que identificam facilmente requisitos ambíguos e mal compreendidos (ROBERTSON; ROBERTSON, 2012). Esse levantamento de requisitos tende a antecipar o surgimento dos erros de entendimento e inconsistências, aprimorando o processo de desenvolvimento de produtos de software (SILVA; BONIN; PALUDO, 2005).

Empresas têm usado cada vez mais conceitos de Engenharia de Software para gerenciar o processo de desenvolvimento de softwares de modo a garantir qualidade. O processo de desenvolvimento de um software gera uma documentação muito importante que inclui diversas métricas do desenvolvimento do software, como prazos, custos e riscos envolvidos. A mineração de dados, principal atividade do processo de descoberta de conhecimento, consiste em aplicar algoritmos com a finalidade de extrair conhecimento de bases de dados. Esta descoberta de conhecimento pode ser realizada de diversas formas: agrupamentos, hipóteses, regras de associação, árvores de decisão, redes neurais, dentre outras. A mineração de dados pode, portanto, ajudar a encontrar padrões relevantes para o desenvolvimento de projetos futuros (ALMEIDA, 2017).

Em um processo de desenvolvimento de software é preciso medir custo, produtividade e qualidade não só do produto final, mas também de todo o processo. Com a implantação de uma ferramenta de histórias de usuários e de coleta de métricas, os desenvolvedores poderão avaliar melhor a sua produtividade e adaptabilidade ao processo de desenvolvimento e, com isso, estimar melhor o tempo necessário para executar cada tarefa. Além de tornar possível o entendimento do processo, para facilitar a previsão de suas fases e mostrar como controlá-las (GOMES, 2013).

Com base no modelo Volere e na definição de métricas que atentem para as necessidades de pequenas empresas de base tecnológica que adotaram métodos ágeis pretende-se construir um ensaio sobre uma ferramenta para apoiar o time ágil na priorização das histórias na ótica prática, tendo em vista métricas alimentadas durante a construção dessas histórias.

Éproposto que essa problemática é alcançada usando-se de um algoritmo que use múltiplas variáveis e métricas de desenvolvimento de software dentro da metodologia ágil para sugerir dentre as diversas soluções possíveis a melhor priorização de histórias de usuários. Maximizando as chances de sucesso e diminuindo o tempo e recursos gastos em cada ciclo de trabalho de pequenas equipes de desenvolvimento. Para tanto deve ser levado em conta as métricas e valores, com importância real, levantadas durante o decorrer deste trabalho.

1.3 Objetivos

1.3.1 Geral

- Entregar um protótipo de uma ferramenta que possa sugerir uma ordem de prioridade de histórias de usuários baseada em padrões e métricas conceituados de métodos ágeis, para assim potencializar o trabalho ágil de equipes de desenvolvimento de software facilitando a adoção de métodos ágeis;

1.3.2 Específicos

- Identificar métricas e padrões que melhor correspondam às necessidades de pequenas equipes de desenvolvimento para priorização de histórias de casos de uso;
- Estudar e apontar algoritmos que permitam implementar a priorização automática de histórias de usuários;

- Implementar um algoritmo que possa, com base nos parâmetros fornecidos pela equipe, priorizar os requisitos de forma eficiente evitando retrabalho, ociosidade da equipe e aumento de custos;
- Levantar padrões e métricas não usuais no mercado que devem ser consideradas na priorização de histórias de usuário evitando retrabalho e aumento dos custos.

1.4 Organização do Trabalho

Além do capítulo inicial para apresentação do contexto em que o problema a ser atendido se situa e suas características, temos mais cinco capítulos organizados da seguinte forma:

- O capítulo 2 apresenta a pesquisa bibliográfica sobre fundamentações teóricas dos principais conceitos, técnicas e métricas empregadas no estudo. Na primeira parte da fundamentação teórica são abordados os conceitos referentes ao campo da Engenharia de Software, área da computação a ser atendida pelo modelo proposto neste documento. Em seguida, são abordados conceitos fundamentais de Mineração de Dados e Árvores de Decisão, bem como as ferramentas utilizadas na produção da ferramenta proposta.
- O capítulo 3 apresenta os métodos e técnicas utilizados para o desenvolvimento deste trabalho.
- O capítulo 4 apresenta os resultados e discussões.
- O capítulo 5 apresenta as considerações finais.
- O capítulo 6 apresenta sugestões de trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão expostos conceitos-chave sobre Engenharia de Software, Gerência de Riscos, Métodos Ágeis e Mineração de Dados. O objetivo deste é servir de base de conhecimento sobre o assunto que será abordado durante este trabalho, para que o objeto do estudo e os resultados obtidos possam ser compreendidos.

2.1 Modelo Volere

O Volere é um modelo base para especificação de requisitos fornecendo seções para cada tipo de requisitos e que pode ser adaptado ao seu processo. Segundo seus autores, James e Suzanne Robertson, é um roteiro bem estruturado e completo para obtenção de requisitos. Ele pode ser usado com qualquer ferramenta de Engenharia de Requisitos. Este é um método que já foi adotado por milhares de organizações no mundo todo e sua construção vem da sumarização de experiências em desenvolvimento de software para montagem de um modelo simples e completo (ROBERTSON; ROBERTSON, 2012; MAIDEN, 2006). Sendo assim o modelo foi escolhido como base para o processo de levantamento de requisitos para a ferramenta de priorização de histórias de usuários.

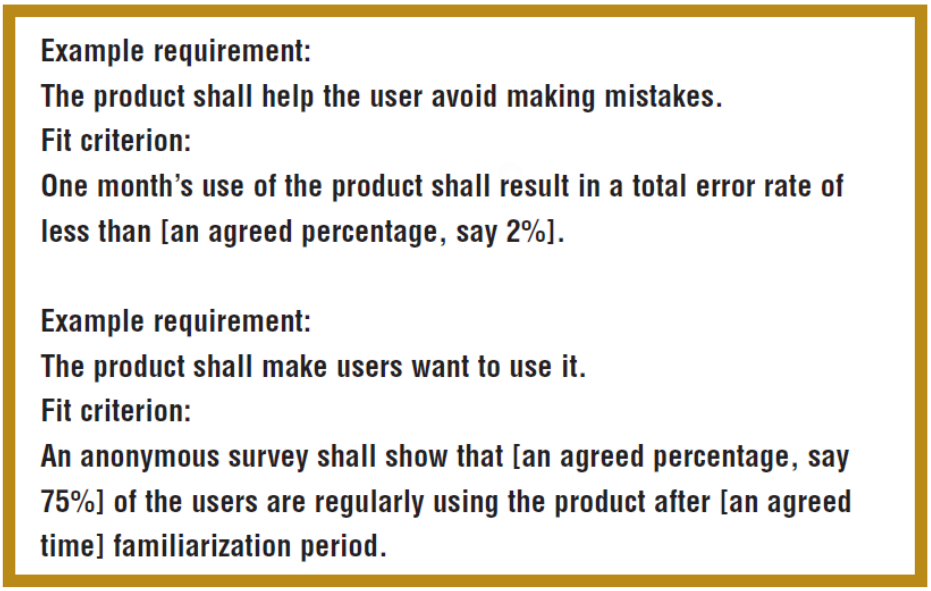
Este modelo reúne sistematicamente várias etapas do ciclo de vida dos requisitos de software: identificação das pessoas e sistemas relacionados, identificação das ações e mensagens trocadas a partir de eventos, formação de requisitos funcionais e não-funcionais, resolução de requisitos conflitantes, critérios de qualidade e protótipos. Os casos de uso devem apresentar, através das fichas do modelo, seus requisitos funcionais e não funcionais e os de restrições. Essas fichas servem para coleta, teste e documentação de cada requisito. Para utilizar o modelo basta escolher entre as opções propostas por Robertson & Robertson as que melhor se adéquam ao projeto. Caso seja necessário novas seções podem ser adicionadas (MANSANI; RIBAS; PALUDO, 2008).

As fichas Volere possuem a descrição, a razão, o critério de aceitação, as dependências, os conflitos, entre outros detalhes sobre cada um dos requisitos e caso de uso. Cada caso de uso pode ter diversos cartões associados a ele (SILVA; BONIN; PALUDO, 2005).

O modelo Volere usa o conceito de critério de ajuste ou critério de aceitação (um dos atributos mais críticos de um requisito) que determina metas quantificadas que o sistema deve atender. Apesar da descrição do requisito ser na linguagem das partes interessadas, o critério de ajuste é escrito de forma precisa e quantificada

para que os analistas possam testar soluções em relação ao requisito. Volere usa a orientação de quantificação orientada por exemplo que baseia-se em taxonomia direta de requisitos, uma ferramenta simples mas eficaz para quantificar seus requisitos inspirando e guiando a quantificação (MAIDEN, 2006).

Figura 1 – Exemplo de requisito e critério de ajuste para Facilidade de Uso.



(MAIDEN, 2006)

2.2 Métodos ágeis

O aumento na demanda por softwares na década de 1970 que veio junto com o avanço do hardware, levou o mercado a passar por uma série de problemas de desenvolvimento de sistemas, período conhecido como a crise do software. A Engenharia de Software percebeu a necessidade da criação de ferramentas que dessem suporte às empresas e equipes de desenvolvimento de software. Essas ferramentas podem ser resumidas em modelos de processos de desenvolvimento de software, metodologias de desenvolvimento ágil e guia de melhores práticas (SANTOS et al., 2016)

O gerenciamento ágil de projetos é uma ferramenta largamente usada para gerenciar projetos dentro de empresas que atuam na área de tecnologia de informação. Esse tipo de gestão viabiliza a flexibilidade que o mercado demanda fornecendo agilidade e eficgiência no desenvolvimento do projeto e do produto (FERREIRA JUNIOR et al., 2016).

Após mais de quatro décadas as empresas de software evoluíram consideravelmente com a ajuda das novas ferramentas de engenharia de software. Em contrapartida

ainda é possível encontrar os mesmos problemas recorrentes durante a crise de software. Na maioria dos casos ocorrem em empresas de pequeno e médio porte que, por contarem com recursos limitados ou pela não adequação das ferramentas auxiliadoras à realidade da empresa, optam por empregar um ciclo de desenvolvimento imaturo, fora de padrões e com tendência a problemas tardios. Metodologias ágeis focam emprego do desenvolvimento de software interativo, focando no código sendo desenvolvido, diminuindo as formalidades e documentação (SANTOS et al., 2016)

2.2.1 Scrum

Segundo o VersionOne (ONE, 2017) o Scrum é a metodologia ágil disparadamente mais utilizada atualmente. Ele fornece um conjunto de práticas que tem como objetivo manter o gerenciamento do projeto visível aos usuários do modelo. Esta metodologia não especifica o que deve ser feito e tão pouco resolve os problemas da empresa, o objetivo do Scrum é dar visibilidade a estes problemas e servir como guia na resolução dos mesmos(FERREIRA JUNIOR et al., 2016). O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos ((SCHWABER; SUTHERLAND, 2013).

O Scrum é uma metodologia apoiada por três pilares: transparência, inspeção e adaptação. A transparência defende que processos específicos devem estar sempre visíveis para seus responsáveis através dos resultados e, que os observadores possuam meios para o entendimento sobre o desenvolvimento do produto. O pilar de inspeção baseia-se na frequência de inspeções dos artefatos Scrum e o progresso em direção, apontando mudanças. O pilar da adaptação, representa a capacidade de ajustes das mudanças, quando essas estão ultrapassando os limites admissíveis (FERREIRA JUNIOR et al., 2016).

Papéis, eventos, artefatos e regras são a base do desenvolvimento do Scrum. Para o sucesso do Scrum cada integrante da equipe deve assumir um propósito. É composto por um time, denominado Scrum Team, constituído de três integrantes principais: Product Owner, Scrum Master e Development Team (Time de Desenvolvimento). E por artefatos e eventos importantes, entre os quais destacam-se: Product Backlog, Planning Sprint, Sprint, Daily Scrum, Sprint Review e Sprint Retrospective (MARTINS, 2016).

O Scrum é um método enxuto de desenvolvimento de produtos, que consiste em um processo eficiente de desenvolvimento de maneira iterativa ou incremental. O Scrum fornece controles práticos que ajudam a fluir o fluxo de desenvolvimento mediante a complexidade do projeto. Usar o Scrum no desenvolvimento de projetos

de software permite entregar o produto em menor tempo sem que ocorra perda da qualidade (SANTOS et al., 2016).

O início do processo se dá com a visão de como o produto será desenvolvido oriunda das definições do cliente, premissas e condições. O *Product Backlog* é preparado contendo a lista de todos os requisitos reconhecidos, em seguida ele é priorizado e dividido em *releases*. Cada *release* contém um subgrupo de requisitos, denominado *Sprint Backlog*, a ser desenvolvido em uma iteração, denominada de *Sprint* (MARÇAL et al., 2007).

Durante a execução da Sprint, diariamente a equipe faz reuniões de 15 minutos (*Daily Scrum Meeting*) para ficar a par do andamento do projeto. Ao final da *Sprint*, é realizada o *Sprint Review Meeting* de modo que o time apresente o resultado alcançado ao *Product Owner* (representante do cliente). Posteriormente, o *Scrum Master* direciona o *Sprint Retrospective Meeting* com o propósito de melhorar a equipe, o processo ou o produto para a próxima *Sprint* (MARÇAL et al., 2007).

2.3 Métricas de Estimativa em Projetos Ágeis

2.3.1 Estimativas de Software

A estimativa do esforço necessário para se desenvolver um projeto de software é fundamental para o sucesso por permitir uma gerência de projetos mais assertiva no que se refere a prazos e custos. Esse processo é realizado através de técnicas de estimativas usando métricas de projeto, que podem ser analogias entre projetos, baseadas na experiência do desenvolvedor ou usando modelos matemáticos por exemplo (RITTER, 2006).

Em métodos ágeis as estimativas são compostas por dois valores $<$v, p$>$, sendo v um palpite sobre um evento e p a probabilidade deste evento ocorrer. Equipes ágeis não lidam com suas estimativas como fatos e admitem que existe uma incerteza no valor a ser estimado e deixam isso bem evidente para os demais envolvidos terem noção do nível de dificuldade de cada tarefa a ser entregue (BASSI FILHO, 2008).

2.4 Complexidade/Tamanho

Estimativas de software usam o princípio que o tamanho de um software é uma métrica para determinar o esforço desprendido em seu desenvolvimento (RITTER, 2006). Esforço é uma variante essencial para a execução, tempo e custo do

desenvolvimento (HAZAN, 2008). Com estas informações bem definidas é possível garantir um melhor gerenciamento do projeto, obtendo-se assim maior exatidão no processo (RITTER, 2006).

A unidade de medida de estimativa pode ser pontos, que fornecem estimativas sobre o volume de trabalho a ser realizado. Quando se usa pontos as medidas são abstratas e definidas pela equipe, baseados numa tarefa pequena que servirá de comparação para as demais funcionalidades .

Através do tamanho de um software é estimado o esforço necessário no desenvolvimento do projeto e através do esforço é possível descobrir o custo do projeto. Uma estimativa de tamanho de projeto requer conhecimento sobre técnicas de estimativas, base histórica e conhecimento sobre o projeto a ser estimado.

Existem várias técnicas de estimativas que se distinguem em suas peculiaridades, mas todas possuem os seguintes atributos em comum (RITTER, 2006):

- Escopo, prioridades e restrições previamente estabelecidas;
- Métricas de software;
- Informações históricas usadas como base de estimativas.

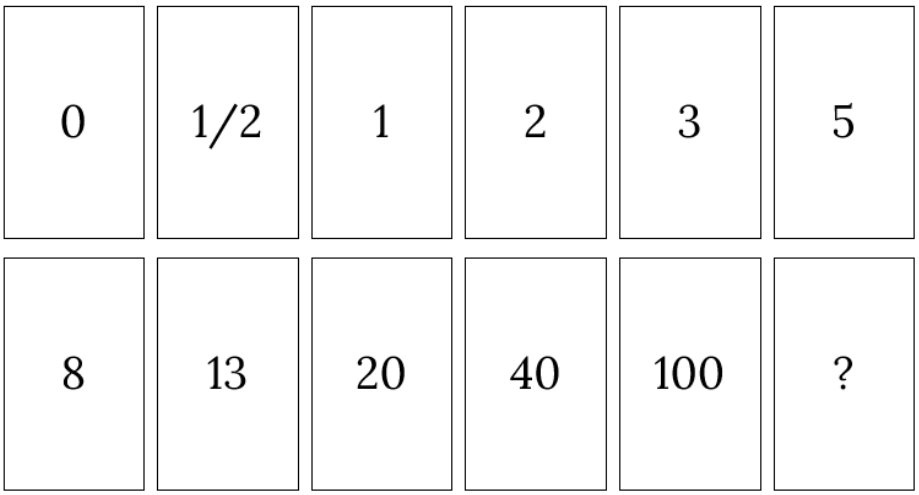
2.4.1 Planning Poker

O Planning Poker se trata de uma técnica de estimativa para metodologias ágeis, inclusive para Scrum no qual James Grenning (GRENNING, 2002)a aponta como a melhor escolha de ferramenta para estimativa de tamanho de projetos com métodos ágeis. Ele derivou da junção de três técnicas menos comuns de métodos ágeis: analogia, opinião de especialistas e desagregação. Isso o torna muito interativo e prático.

A ideia principal dessa técnica é usar um jogo de cartas para que toda a equipe de desenvolvimento participe e coloque sua visão de complexidade considerando o tempo e esforço na pontuação das estórias e em conjunto chegar a um consenso (RITTER, 2006).

No planning Poker os integrantes têm acesso a um baralho de 12 cartas, numeradas numa sequência similar nos números de Fibonacci como mostrado na Figura 2. O recomendável é que no fim do Planning Poker a equipe tenha conseguido com que as estórias da iteração possuam valores entre 12 e 13. Se a estória tem um valor maior que 13 ela pode ser detalhada e quebrada em estórias menores. A equipe também deve evitar estórias muito pequenas para evitar o micro gerenciamento (RITTER, 2006).

Figura 2 – Cartas de Planning Poker



Fonte: Autor, baseado em (Roger, 2006).

Os seguintes procedimentos devem ser tomados ao se estimar uma estória, quando realizado por uma equipe Scrum (COHN, 2005; CTIC-UFPA, 2011)

1) Pontua-se a estória que mais se aproxima do valor ‘3’ (três), que servirá como referência para as demais.

2) O Product Owner descrevefe para a equipe Scrum cada estória e fornece informações sobre seu valor de negócio. A seguir, o coordenador vai perguntar o esforço necessário para desenvolver a história aos membros da equipe.

3) Cada membro da equipe deve pensar a respeito do tempo e esforço necessário para se implementar a estória lida. Os membros da equipe devem estimar considerando todas as tarefas envolvidas pelos responsáveis em desenvolver, testar, criar design, etc. Então, deve escolher uma carta no baralho correspondente ao valor desta estimativa e coloca-a virada para baixo.

4) Quando todos os membros fizerem o procedimento acima, então devem revelar as cartas escolhidas simultaneamente. Isso faz com que cada membro da equipe pense por si próprio na hora de uma decisão de estimativa.

5) Todos avaliam os resultados e verificam se houve convergência entre as cartas mostradas, ou seja, todas as estimativas possuam valores aproximados para a mesma estória.

6) Caso contrário, o Scrum Master solicita aos membros, que mostraram o menor e o maior valor estimado, que expliquem o motivo que os levaram a tal estimativa. Isto faz com que os integrantes reflitam sobre alguns pontos da estória, o que

pode fazer com que mudem os valores propostos para as estimativas. Então, uma nova rodada é realizada até que as estimativas de esforço cheguem a uma convergência.

7) A estimativa final da estória será o valor que tiver maior ocorrência ou a média entre as estimativas informadas. Então começa uma nova rodada com a leitura da próxima estória pelo *Product Owner*. O processo se repete até que tenha sido concluída a estimativa das estórias dos itens do *Product Backlog*.

Estudos de estimativa de esforço (MOLOKKEN-OSTVOLD; HAUGEN, 2007) realizados em um ambiente Scrum mostraram que as estimativas de consenso do grupo eram menos otimistas e mais precisas do que a combinação estatística das estimativas individuais. O planning poker é o método ágil de estimativa mais comumente utilizado, porém altamente dependente de um profissional experiente ou uma base histórica de estimativas para se alcançar uma boa precisão ((RITTER, 2006; MAHNIC; HOVELJA, 2012; BHALERAO; INGLE, 2009).

2.4.2 Story Points

Tratando-se de uma estimativa de esforço é necessário utilizar-se de métricas de tamanho. Estudos de casos (USMAN et al., 2014) mostram que as mais comumente usadas são os Story Points (SP) e os Use Case Points (UCP). (FONSECA, 2008) mostra uma melhora na eficiência das estimativas com pontos por estórias quando utilizada esta técnica para atribuição dos pontos.

2.4.3 Geradores de Custo

Através do estudo em (USMAN et al., 2014)foram apontados vários geradores de custo, dos quais pela filosofia ágil se destacam como cruciais:

- Tamanho da tarefa;
- Competência/habilidades do time;
- Experiência prévia do time ágil.

O trabalho ainda reforça a importância da estimativa de esforço de projetos ágeis ,o que é benéfico para empresas ágeis com pouca base para estimativas e que essa estimativa deveria levar em consideração outros preditores de esforço (ANWER et al., 2014)

Em (BHALERAO; INGLE, 2009; USMAN et al., 2014)são apontados como geradores de custo/esforço outros fatores como:

- Teste de fator de Eficiência e Teste de Fator de Risco
- Área do projeto: Web (Baixo); Sistema de gerenciamento de informações; Comercial, Outsorcing (Médio); Militar e Científico (Alto) (BHALERAO; INGLE, 2009) ;
- Perfomance: Alta, Média, Baixa. O desempenho do projeto pode ser caracterizado pelo tempo de execução, padrões de design e codificação, precisão no resultado, etc., de acordo com os requisitos do cliente (BHALERAO; INGLE, 2009) ;
- Processamento Complexo: As práticas ágeis sugerem refatoração do código e dos dados para simplificar o design e a codificação. Ainda assim, o processamento complexo, como regras comerciais e científicas ou acesso remoto, precisa de uma maior eficiência dos membros da equipe e testes rigorosos para evitar a redundância (BHALERAO; INGLE, 2009) ;
- Nível de Transação de dados: O volume e frequência de transação de dados afeta diretamente o design e implementação do projeto (BHALERAO; INGLE, 2009) ;
- Desenvolvimento distribuído geograficamente: Se a equipe está distribuída isso aumenta a relevância do grau de risco técnico a ser considerado em um projeto. Fator que, em geral está muito associado com a necessidade de mudanças nos códigos e da complexidade desses códigos.

Antinyan (ANTINYAN et al., 2014) identifica 24 riscos técnicos recorrentes no mercado de desenvolvimento de software. Deste montante se destacam alguns riscos técnicos por aparecem em outros estudos (ARNUPHAPTRAIRONG, 2011; ROPPONEN; LYYTINEN, 1997), o que leva a crer que ocorrem com mais frequência. Delas podemos citar:

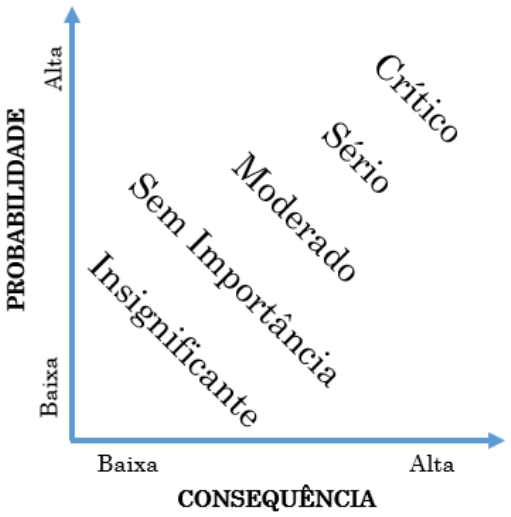
- Representação inadequada de requisitos: o requisito contém fatores que diminuem o entendimento no sentido semântico como por pseudo-código, referências a outros documentos ou a outros requisitos, etc;
- Mudanças de requisitos: alterações frequentes mesmo por causa de objetivos de negócios sólidos podem causar risco de entrega tardia do produto;
- Dependência de componentes externos fornecidos: o arquivo depende diretamente e “fortemente” de componentes fora do nosso controle;
- Dependência de outras tarefas: depende diretamente e “fortemente” da execução de tarefas fora do seu controle;

As técnicas usadas na avaliação de risco estão classificadas em duas categorias básicas: Qualitativas ou Quantitativas. As técnicas qualitativas empregam um método descritivo, onde palavras subjetivas indicam o nível do risco, por exemplo, o risco pode ser alto, médio ou baixo. Por outro lado, as técnicas quantitativas transformam essa subjetividade, de modo a obter um valor monetário ou numérico que indique o nível do risco (SOUZA; MOURA, 2010).

Os métodos quantitativos geralmente seguem os métodos qualitativos de risco. Kleining salienta que os métodos qualitativos podem viver muito bem sem o posterior emprego de métodos quantitativos; porém, os quantitativos precisam dos anteriores para explicar as relações encontradas (KLEINING, 1982). Existem referências diversas para se quantificar o risco como é o caso do modelo proposto por Porthin e do PMBOK a serem descritos a seguir.

De acordo com Porthin (PORTHIN, 2004), a gravidade de um risco pode ser avaliada quantitativamente mapeando o risco numa matriz de acordo com a repercussão do resultado e sua probabilidade ou frequência de ocorrência, conforme apresentado na figura abaixo. Quanto mais perto do canto superior direito se situa o risco, mais crítico é a sua representatividade (SOUZA; MOURA, 2010).

Figura 3 – Gráfico Probabilidade X Consequência



Adaptado de PORTHIN, 2004.

Segundo o PMBOK (PMI, 2013) a análise qualitativa dos riscos pode ser baseada na avaliação de sua probabilidade e impacto. A avaliação da importância de cada risco e a prioridade de atenção pode ser feita com uma matriz de probabilidade e impacto que determina as combinações de probabilidade e impacto gerando uma classificação dos riscos como de prioridade baixa, moderada ou alta. De acordo com a

preferência da organização podem ser usados termos descritivos ou valores numéricos. Cada risco é classificado de acordo com a sua probabilidade de ocorrência e impacto em um objetivo (custo, tempo, qualidade, etc), se ele realmente ocorrer. Os limites de tolerância da organização para riscos baixos, moderados ou altos são mostrados na matriz e determinam se o risco é alto, moderado ou baixo para aquele objetivo.

Figura 4 – Matriz de probabilidade e impacto

Prob.	Ameaças					Oportunidades				
0.90	0.05	0.09	0.18	0.36	0.72	0.72	0.36	0.18	0.09	0.05
0.70	0.04	0.07	0.14	0.28	0.56	0.56	0.28	0.14	0.07	0.04
0.50	0.03	0.05	0.10	0.20	0.40	0.40	0.20	0.10	0.05	0.03
0.30	0.02	0.03	0.06	0.12	0.24	0.24	0.12	0.06	0.03	0.02
0.10	0.01	0.01	0.02	0.04	0.08	0.08	0.04	0.02	0.01	0.01
	0.05	0.10	0.20	0.40	0.80	0.80	0.40	0.20	0.10	0.05

Fonte: https://danielettinger.com/2011/06/14/gerenciando-os-riscos-do-projeto-com-a-matriz-de-probabilidade-e-impacto (2011)

A pontuação dos riscos ajuda a orientar as respostas aos riscos. Por exemplo, os riscos com impacto negativo nos objetivos (ameaças) e que estão na zona de alto risco (cinza escuro) podem exigir uma ação prioritária e estratégias agressivas como resposta. As ameaças que estão na zona de baixo risco (cinza médio) podem não exigir uma ação proativa de gerenciamento. De forma semelhante, os riscos com impacto positivo nos objetivos (oportunidades) na zona de alto risco (cinza escuro) podem ser obtidas mais facilmente e oferecem o maior benefício, devem ser abordadas primeiro. As oportunidades na zona de baixo risco (cinza médio) devem ser monitoradas.

Segundo Sotille (SOTILLE, 2003), a medição qualitativa requer dados precisos e não tendenciosos, implicando em examinar a medida do entendimento do risco, a qualidade, confiabilidade e integridade dos dados disponíveis sobre o mesmo. A análise quantitativa proposta pelo PMBOK foi a escolhida para ser aplicada a este trabalho por ser um modelo amplo e completo, bem descrito e que permite a entrega de valores precisos.

2.5 Satisfação e Insatisfação do Cliente

Existem várias maneiras de medir o valor de negócio das funcionalidades de um software. Estes incluem Valor Presente Líquido (NPV), Taxa Interna de Retorno (IRR)

e Retorno no Investimento (ROI). Cada método tem seus benefícios e desvantagens. É recomendado que o negócio determine seu método preferido para valorizar seu investimento (HARTMANN; DYMOND, 2006).

O Scrum usa o Business Value (Valor de Negócio) como técnica de estimativa. É comum nesta técnica que o cliente ponha vários requisitos, senão todos, com o valor de negócio muito alto e assim dificultando a real noção do que realmente é mais importante. É natural essa tendência a dar alto valor às atividades a serem desempenhadas por um software. Para contornar essa situação é interessante incluir o critério de insatisfação para perceber o que realmente deixa infeliz se não for desenvolvido. O Volere é uma das ferramentas que usam os critérios de satisfação e insatisfação que o cliente agrega a cada requisito (ASFORA; ALVES, 2009).

A técnica do modelo Volere consiste em fazer duas perguntas sobre cada requisito, uma pergunta funcional e outra disfuncional: a satisfação e insatisfação caso o requisito seja desenvolvido. No Volere essas respostas são colocadas em valores de 1 a 5. Para satisfação, em 1 o cliente não está preocupado se o requisito vai ser entregue. E 5 ele ficará muito feliz se esse requisito for implementado. Já para insatisfação, a classificação é a seguinte: 1 para não liga se o requisito não for entregue e 5 para extremamente infeliz se o requisito não for implementado. A comparação do que deixa o cliente muito satisfeito e o do que o deixa infeliz é muito importante para se ter a real noção de necessidade de cada requisito (ASFORA; ALVES, 2009).

2.6 Necessidade de Aprendizagem da Equipe.

Habilidades de equipe, experiência prévia e tamanho da tarefa são tidos como os três principais fatores para estimar o esforço no Desenvolvimento Ágil de Software. As habilidades e a experiência dos desenvolvedores desempenham um papel crucial em todas as tarefas do desenvolvimento, incluindo a estimativa do esforço (USMAN et al., 2014).

As necessidades de aprendizagem da equipe está relacionado com o grau de experiência/habilidade da equipe que é abordada na literatura como um aspecto que pode acrescentar complexidade ao projeto. Para este trabalho em específico por se concentrar em equipes ágeis imaturas o desmembramento e ressalva é coerente.

2.7 Projeto como Organização Temporária

A visão do projeto como uma organização temporária, relativamente recente, é analisado a partir da visão da teoria organizacional o que leva a rever os conceitos de um projeto. O projeto como uma organização temporária pode ser visto como uma função de produção, como uma agência de atribuição de recursos para a gerencia-

mento da mudança dentro da organização funcional e como uma agência para gerir a incerteza. Um projeto é uma combinação de recursos humanos e não-humanos postos em conjunto numa organização temporária para atingir uma finalidade específica (TURNER; MÜLLER, 2003).

A definição de projetos vista em (TURNER; MÜLLER, 2003) enfatizam que os projetos entregam mudanças. Organizações tradicionais adotam projetos como um veículo (ou agência) para a mudança. Eles criam a organização temporária para entregar um conjunto coerente de objetivos de mudança, porque os projetos são mais adequados para o gerenciamento da mudança do que a organização funcional. (TURNER; MÜLLER, 2003)cita alguns benefícios do uso de projeto como organização temporária:

- As organizações funcionais têm elevada inércia para mudar e os projetos podem fornecer um impulso para superar a inércia. O projeto como organização pequena e temporária pode ser configurado de forma isolada da organização funcional permitindo a mudança com pouca ou nenhuma inércia;
- Os projetos como pequenas organizações temporárias são mais flexíveis e com melhor resposta às incertezas no processo de mudança.
- A organização funcional é projetada para o gerenciamento da rotina, e por tanto não é adequada para o gerenciamento da mudança.

Um projeto é uma organização temporária para a qual os recursos são atribuídos para um empreendimento único e transitório, gerenciando a incerteza inerente e a necessidade de integração, de modo a proporcionar objetivos benéficos de mudança (TURNER; MÜLLER, 2003).

Em (ARNUPHAPTRAIRONG, 2011) ao pesquisar e listar os principais riscos de software para se obter uma visão melhor da lista, esses riscos foram mapeados usando as seis dimensões de riscos de software de Wallice, que se aproximam muito da visão de um projeto de software como organização temporária:

- Usuário
- Requisitos
- Complexidade de Projeto: detalhes técnicos do projeto
- Planejamento e Controle: gerenciamento do projeto
- Time/Equipe
- Envolvimento Organizacional

Baseado na abstração de um projeto de software como organização temporária (TURNER; MÜLLER, 2003) e na categorização de riscos de software de (ARNUPHAPTRAIRONG, 2011), este trabalho propõe a adoção de marcadores para os riscos de softwares a fim de facilitar o levantamento dos dados de requisitos durante o uso da ferramenta de apoio à decisão e como forma de organização dos dados voltados para requisitos ágeis. Assim ao se usar a ferramenta e alimentá-la com dados das histórias do projeto evita-se o retrabalho de alimentação de dados. Ao se fornecer informações sobre a história não é necessário realimentar o conjunto de dados da árvore de decisão com dados de camadas acima dela como Organização por exemplo. As camadas propostas são:

1) Organização
2) Projeto
3) Time
4) História

2.8 Planejamento de Iteração Guiado por Alto Risco e Alto Valor de Negócio

O modelo de ciclo de vida incremental e iterativo surgiu como uma resposta aos problemas encontrados no modelo em cascata. Nele o desenvolvimento de um produto de software é dividido em ciclos. Em cada ciclo são implementadas as fases de análise, projeto, implementação e testes. A cada ciclo considera um subconjunto de requisitos para ser desenvolvido e é fabricado um novo incremento do sistema. O produto se desenvolve em versões, numa construção incremental e iterativa. Para a abordagem incremental e iterativa funcionar é necessário um mecanismo para divisão os requisitos do sistema em subconjuntos que serão alocados a um ciclo de desenvolvimento. Essa alocação é dada de acordo com o grau de importância de cada requisito. Para tanto são levados em consideração o valor de negócio (importância do requisito para o cliente) e o risco de cada requisito (BEZERRA, 2015).

Na abordagem incremental, os requisitos mais arriscados são os primeiros a serem considerados. Uma das vantagens dessa abordagem é que os riscos do projeto podem ser melhor geridos. Os riscos de projeto não podem ser eliminados por completo então todo processo de desenvolvimento deve considerar a probabilidade de ocorrência de riscos (BEZERRA, 2015).

Como cada ciclo de desenvolvimento gera uma versão do produto para o cliente, inconsistências entre requisitos e suas implementações são evidenciadas de maneira precoce. Se as inconsistências não forem tão graves são removidas e uma nova versão do sistema é entregue ao usuário. Casos as inconsistências descobertas sejam de grave impacto, a sua identificação torna possível uma reação antecipada sem tantas consequências graves para o projeto. Assim se torna menor a probabilidade de ocorrerem prejuízos devido a esses requisitos com alto risco (BEZERRA, 2015). Segundo (CRAIG, 1989), “Se você não atacar os riscos [do projeto] ativamente, então eles irão ativamente atacar você”

Dessa forma, cada caso de uso se encaixa em uma das categorias a seguir:

1) **Risco alto e prioridade alta:** são os casos de uso mais críticos, devem ser considerados quanto antes.
2) **Risco alto e prioridade baixa:** embora os casos de uso nesta categoria tenham risco alto, é preciso, antes de considerá-los, negociar com o cliente a sua verdadeira necessidade.
3) **Risco baixo e prioridade alta:** embora os casos de uso tenham prioridade alta, é preciso ter em mente que os casos de uso de mais alto risco precisam ser considerados primeiro.
4) **Risco baixo e prioridade baixa:** são os de menor relevância, em situações de desenvolvimento atrasado, estes casos de uso são os primeiros a serem “cortados”.

Segundo (LARMAN, 2007) a maioria dos novos métodos incentiva a combinação de planejamento iterativo guiado por risco e por cliente. Isso quer dizer que as finalidades das iterações iniciais devem ser selecionadas para:

1) Identificar e gerir os maiores riscos.
2) Desenvolver características visíveis com as quais o cliente se preocupa.

2.9 Mineração de Dados

O processo que utiliza técnicas de Inteligência Artificial para proporcionar reconhecimento/extração de padrões e posteriormente aplica manipulação de dados e de análises pode ser entendido como Mineração de Dados (MD) (GOMES, 2015).

> "Extração de Conhecimento de Base de Dados e o processo de identificação de padões válidos, novos, potencialmente úteis e compreensíıveis embutidos nos dados".
>
> Fayyad, Piatetsky-Shapiro, & Smyth (1996a)

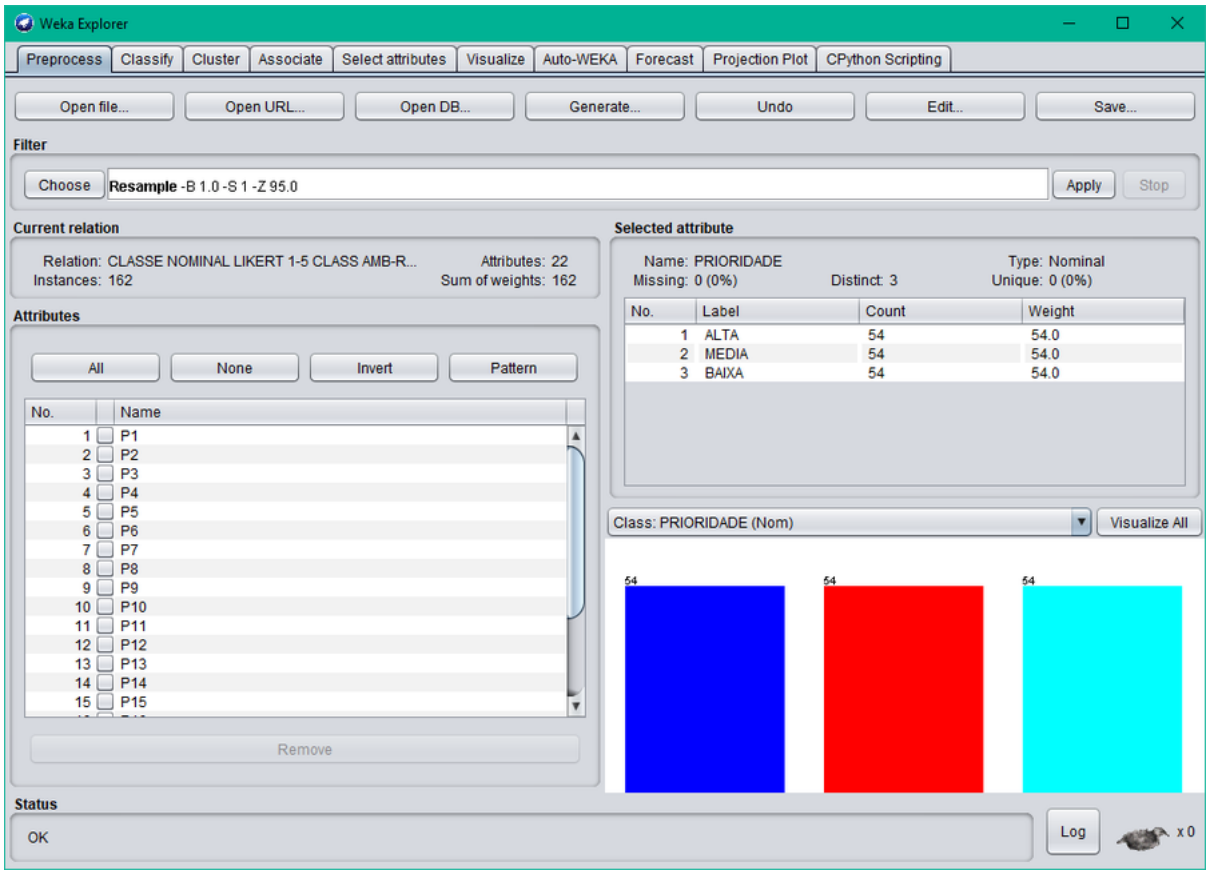
A descoberta de padrões é um processo que começa pela seleção dos dados que contribuem de alguma forma as questões que os especialistas desejam responder. Os dados devem ser integrados e pré-processados passando por processos de limpeza, seleção e padronização. Na etapa seguinte , a de mineração, aplica-se alguma técnica inteligente que permita o encontro de soluções auxiliadoras na descoberta de respostas. Os resultados devem ser pós-processados para se obter análises qualitativas ou quantitativas e, quando possível, apresentados de maneira que possa ser interpretada de modo a auxiliar na tomada de decisão (RODRIGUES et al., 2014).

2.9.1 Weka

Estão disponíveis variadas ferramentas com a finalidade de auxiliar o processo de mineração de dados, como é o caso da Weka, um software de código aberto emitido sob a Licença Pública Geral GNU. O Weka reuni uma coleção de algoritmos de aprendizagem de máquina que permitem a realização de tarefas de mineração de dados. Estes algoritmos podem ser aplicados diretamente ao conjunto de dados ou podem serem chamados através de código Java. O software possui ferramentas para pré-processamento de dados, classificação, regressão, agrupamento, regras de associação e visualização. Weka é um software de código aberto sob a Licença Pública Geral GNU sendo adequado ainda para o desenvolvimento de novos esquemas de aprendizagem de máquina (WAIKATO,).

A suíte Weka foi usada neste trabalho tanto para pré processamento de dados como para implementação e ajustes de algoritmos de aprendizagem de máquina, mais detalhes podem ser observados no Capítulo 3. Na Figura 5 podemos observar uma das telas da suíte Weka com os dados deste trabalho carregados.

Figura 5 – Ferramenta Weka Explorer para Windows.



Fonte: Autor

2.10 Aprendizagem de Máquina

O Aprendizado de Máquina (AM) pode ser concebido como um subcampo da Inteligência Artificial que explora métodos computacionais quanto a aquisição automática de conhecimentos, novas habilidades e novas maneiras de organizar o conhecimento já existente. O aprendizado de máquina pode, ainda, ser usado contra um dos maiores problemas enfrentados pelos Sistemas de Inteligência Artificial: o gargalo da aquisição de conhecimento (LEE, 2000). Um sistema de AM é, portanto, um software que toma decisões com base em experiências acumuladas derivadas de casos bem sucedidos (WEISS; KULIKOWSKI, 1990).

Apesar do aprendizado de máquina ser uma ferramenta poderosa para a aquisição automática de conhecimento é importante ressaltar que não há um único algoritmo que possua o melhor desempenho na resolução de todos os problemas. Assim, é necessário entender as limitações dos variados algoritmos de AM usando alguma metodologia que permita avaliar os conceitos induzidos por estes algoritmos em determinados problemas (GOMES, 2015).

Dentro do ambiente de AM, a Classificação pode ser definida como a predição

de novos dados em classes usando um classificador modelado por um algoritmo de AM. É considerada a técnica mais usada para resolver problemas e mineração de dados. No caso da Regressão ela é utlizada quando se tem como saída um valor numérico (CARVALHO, 2014).

2.11 Árvores de Decisão

O modelo de classificação ou regressão com uma estrutura que pode ser resumida em um conjunto de determinados nós e arcos (ou ramos) é conhecido como Árvore de Decisão (AD) (FÜRNKRANZ; GAMBERGER; LAVRAČ, 2012). Este modelo possui estrutura no formato de uma árvore, onde cada nó interno da árvore expressa um determinado teste em uma característica de uma instância, e os arcos por sua vez representam o resultado do teste realizado. Os nós externos da árvore, também chamados de nós-folha ou nós terminais, representam as classes de classificação. Para uma instância ser classificada é preciso percorrer a Árvore de Decisão de cima para baixo, seguindo pelos arcos dos nós onde as características das instâncias satisfazem os testes realizados em cada nó até chegar em um nó-folha, que contém a classificação atribuída aquela instância (CARVALHO, 2014).

Árvores de Decisão é um dos algoritmos de aprendizado de máquina mais usados em Mineração de Dados. Existem vários motivos para isso, entre eles: é um modelo que lida com variados tipos de características, sua representação gráfica de conhecimento adquirido é útil e de fácil entendimento, e o seu processo de aprendizado e treinamento é relativamente rápido, comparado a outros algoritmos (CARVALHO, 2014).

A compreensibilidade é algo importante ao se escolher um método de AM. Compreensibilidade pode ser delimitada como a capacidade de compreender a resposta de um determinado preditor. Considerando a aplicação é fundamental o tomador de decisão entender o motivo pelo qual determinado modelo chega a uma determinada conclusão. Árvores de decisão, por possuírem visualização prática e interpretação mais intuitiva, são recomendadas como métodos de apoio a decisão de profissionais ao invés de processos *black-boc* ou não-lineares, como redes neurais e SVM (MORAES, 2016). Sendo assim árvore de decisão foi o método escolhido para representar os testes e resultados deste trabalho, mais especificamente o algoritmo REPTree ou Árvore de Decisão de Regressão.

2.11.1 J48

O classificador J48 é uma árvore de decisão C4.5 direta para classificação, que cria uma árvore binária. É a abordagem de árvore de decisão mais útil para

problemas de classificação. Essa técnica constrói uma árvore para modelar o processo de classificação. Depois que a árvore é construída, o algoritmo é aplicado a cada tupla no banco de dados e resulta na classificação para essa tupla (MORAES, 2016).

2.11.2 Random Forest

Acredita-se que o classificador Random Forest (Florestas Aleatórias) seja a melhor opção “off-the-shelf” (prontos para uso) para dados de alta dimensão. Random Forest são um conjunto de árvores preditoras onde cada árvore depende de valores de um vetor aleatório amostrado de forma autônoma e com uma distribuição igual para todas as árvores na floresta (DEVASENA, 2014).

O erro de generalização em florestas depende da força das árvores individuais na floresta e da associação entre elas e converge para um limite quando o número de árvores na floresta se torna grande. Cada árvore é treinada com um subconjunto distinto dos dados de treinamento, com substituição. Os dados de treinamento remanescentes são utilizados para estimar erro e importância de variável. A atribuição de classe é realizada pelo número de votos de todas as árvores e, no caso de regressão, é utilizada a média dos resultados. É semelhante às árvores de decisão ensacadas, com algumas diferenças importantes, como a não poda. No Random Forest as árvores podem ser cultivadas até que cada nó contenha apenas algumas poucas observações. Tem como vantagem possuir uma melhor previsão e não necessitar de quase nenhum ajuste de parâmetros (DEVASENA, 2014).

O ponto fraco desta metodologia é a perca compreensibilidade dos resultados, se posicionando entre uma estrutura simples de árvore de decisão (mais compreensíveis) e os modelos *black-box* (menos compreensíveis) (MORAES, 2016).

2.11.3 REPTree

O Classificador REPTree é uma árvore de decisão de aprendizagem rápida e baseia-se no princípio de computar o ganho de informação com a entropia e minimizar o erro resultante da variação. Ele constrói uma árvore decisão ou regressão usando ganho de informação/variância e, realiza sua poda com redução de erros resultantes da variância. REPTree aplica lógica de árvore de regressão além de montar várias árvores em iterações alteradas. Em seguida escolhe a melhor de todas as árvores geradas. Além disso, este algoritmo realiza poda da árvore usando a remoção de erro reduzido com o método de ajuste de retorno. No início da preparação do modelo, ele classifica os valores dos atributos numéricos uma única vez e, assim como no Algoritmo C4.5, esse algoritmo também lida com os valores ausentes dividindo as instâncias correspondentes em partes(DEVASENA, 2014).

O REPTree foi escolhido como foco principal para este trabalho por ter uma boa representatividade dos resultados gerados pelo algoritmo para os dados utilizados para desenvolvimeneto deste trabalho com representação gráfica na forma de uma árvore não tão ramificada e extensa quanto a gerada por outras árvores de decisão. Além disso o classificador REPTree não se mostrou custoso computacionalmente em relação e teve um resultado relevante quanto a predição das instâncias trabalhadas.

2.12 Validação Cruzada com *Leave-one-out*

A facilidade com que as árvores de decisão podem se especializar demasiadamente representa uma de suas desvantagens em potencial. A recursividade com a qual as AD são construídas facilitam o *overfitting* já que é possível aprofundar-se tanto quanto se queira. Árvores com muitos nós, provavelmente possuem erros dentro das subamostras de dados. No entanto, isto não significa que a generalização para amostras fora do conjunto de treinamento seja ótima. É possível que a árvore esteja memorizando o resultado das observações ao invés de entender os padrões. Assim como também é possível que erros dentro da amostra podem levar a baixa generalização por não aprender de forma correta os padrões presentes nos dados (problema de underfitting) (MORAES, 2016).

Dividir o conjunto de dados em treino e teste, de forma que as observações realizadas em teste não são usadas no treinamento é uma maneira de evitar o *overfitting.* Estas instâncias são consideradas fora da amostra e são usadas para verificar o erro de generalização do classificador. No entanto, em conjuntos com poucas observações, essa separa¸ção pode levar a poucas amostras em cada conjunto e, consequentemente, a baixa generaliza¸ção (MORAES, 2016).

A técnica de Validação Cruzada é amplamente utilizada para calcular se o modelo está sendo executado com precisão satisfatória, evitando os problemas de overfitting e underfitting por escolha ruim dos dados de treinamento. A melhor abordagem para a Validação Cruzada é a utilização do método de k-partições. Neste método, o conjunto de dados é dividido em k partições de forma randômica. O conjunto de dados de treinamento é criado com $k - 1$ partições, e apenas uma partição é usada para testes. Em seguida são feitas k iterações, onde cada partição é utilizada uma vez para testes enquanto as outras são usadas para treinamento. A margem de erro de cada iteração é somada e a média das k iterações se torna a margem de erro do modelo.

O método *leave-one-out* é um caso específico de validação cruzada onde k é igual ao número total de dados N. Nesta abordagem são realizados N cálculos de erro, um para cada dado (MORAES, 2016). O *leave-one-out* é normalmente usada

apenas para aplicações onde a quantidade de dados de treinamento disponíveis é severamente limitada, de forma que até mesmo uma pequena perturbação dos dados de treinamento provavelmente resultarão em uma mudança substancial do modelo. Neste caso, faz sentido utilizar validação cruzada *leave-one-out*, uma vez que essa estratégia minimiza a perturbação dos dados em cada tentativa. Esta técnica é raramente adotada em aplicações de grande escala porque é computacionalmente custoso $O(l^4)$, onde l é o número de padrões de treinamento, ou seja, é igual número total de dados (CAWLEY; TALBOT, 2003).

Pelas características dos dados coletados para este trabalho, de volume reduzido, foi utilizado a técnica de validação cruzada com *leave-one-out* visando a construção de um algoritmo bem estruturado com a maior confiabilidade possível.

2.13 Grid Search

A técnica Grid Search (procura em grade) consiste na busca pela melhor combinação em pares de parâmetros para o tipo selecionado de estimativa escolhida (precisão, por exemplo). O melhor ponto da grade é então obtido e uma validação cruzada de 10 vezes é realizada com os pares de parâmetros adjacentes. Se um par melhor for encontrado, este atuará como novo centro e outra validação cruzada de 10 vezes será realizado (tipo de subida). Esse processo é repetido até que nenhum par melhor seja encontrado ou o melhor par esteja na borda da grade. O Grid Search foi executado através da Weka para otimização de parâmetros com árvores de decisão.

2.14 Classes desbalanceadas

Uma das muitas questões levantadas em AM é a questão das classes desbalanceadas, que é quando domínios de dados são formados por classes que possuem um número bem maior de exemplos que outra classe. Muitos sistemas de aprendizado tomam como base que as classes estão balanceadas acabam gerando classificadores com alta precisão para classes majoritárias, mas, falham em predizer. com precisão aceitável, classes minoritárias (CARVALHO, 2014).

2.15 Matriz Confusão e Curva ROC

Para avaliação do modelo existem outras técnicas além da validação cruzada. A Matriz de Confusão e a Curva ROC[1], ajudam a incrementar a avaliação do modelo, pois apresentam resultados mais detalhados. Através da matriz de confusão, é pos-

[1] ROC é uma sigla para *Receiver Operating Characteristic*, um termo utlizado em detecção de sinais para caracterizar a relação de perda e ganho entre taxa de acerto e taxa de falso alarme em um canal com ruído (BATISTA et al., 2003).

sível verificar quantas instâncias foram corretamente classificadas, e quantas foram incorretamente classificadas. A tabela 1 mostra como que é constituída uma matriz de confusão para um conjunto de classificação com duas classes-alvo:

Tabela 1 – Exemplo de matriz de confusão.

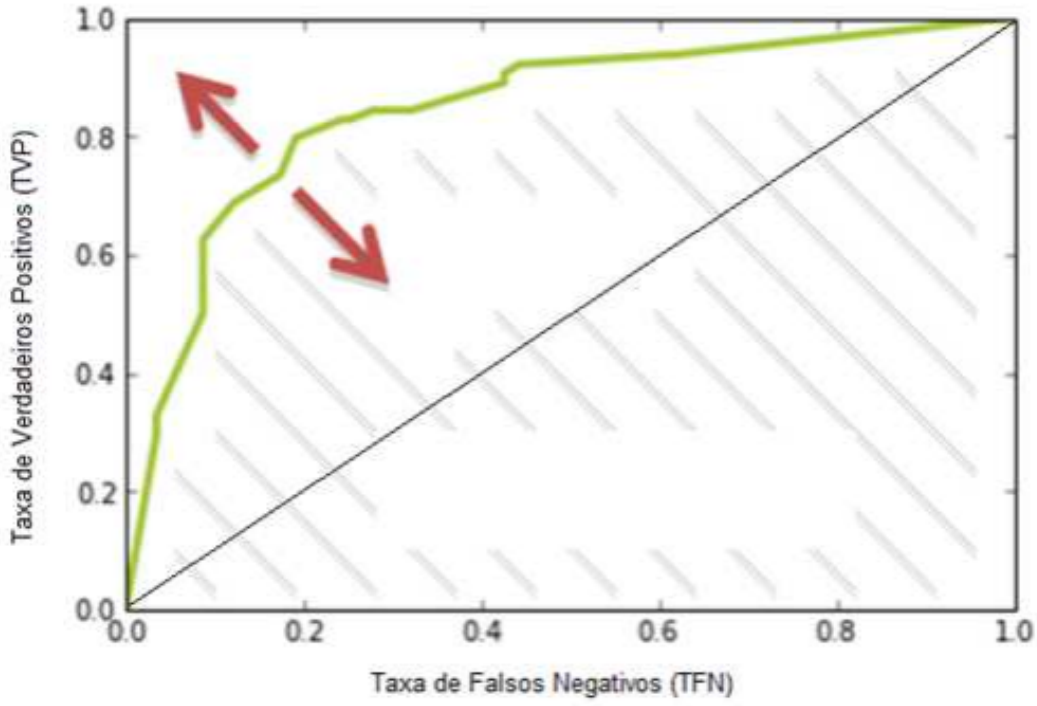
Verdadeiros Positivos (VP)	Falsos Positivos (FP)
Verdadeiros Negativos (VN)	Falsos Negativos (FN)

Adaptado de Carvalho, Hialo Muniz (2015)

Com base na matriz de confusão gerada na execução do modelo escolhido, diversas métricas podem ser obtidas para avaliar o desempenho do modelo. Duas mais importantes para a análise são a sensibilidade e a especificidade. A sensibilidade diz respeito a capacidade do modelo de predizer de forma correta a classe da instância testada e é conhecida como Taxa de Verdadeiro Positivo (TVP). Já a especificidade é a capacidade do modelo em predizer de forma correta que determinada instância NÃO pertence à determinada classe, conhecida como Taxa de Verdadeiro Negativo (TVN) (CARVALHO, 2014).

A análise dos resultados também pode ser realizada por meio de um gráfico conhecido como curva ROC. Ele consiste na plotagem de duas métricas calculadas com base na matriz de confusão: TVP e a Taxa de Falsos Positivos (TFP). Cada modelo se torna um ponto no gráfico, e o seu desempenho pode ser avaliado pela proximidade com o ponto ideal: canto superior esquerdo do gráfico (TVP = 1.0 e TFP = 0.0) onde o modelo classifica todas as suas instâncias corretamente, sem FPs ou FNs (CARVALHO, 2014).

Figura 6 – Exemplo de uma curva ROC



(BRINK; RICHARDS, 2014)

A varição de todos os pontos que podem ser produzidos através de variação dos parâmetros do classificador produzem o desenho de um *curva ROC*. Essa curva é, portanto um conjunto discreto de pontos que são conectados por segmentos de retas. A partir do gráfico *ROC* é possível calcular a área sob a curva (*AUC*[2]), que é fração da área que situa sob a *curva ROC*. Essa é uma medida equivalente as diversas outras medidas estatísticas para a avaliação de modelos de classificação e fatora de forma efetiva o desempenho do classificador perante os custos e distribuições (BATISTA et al., 2003).

2.16 Trabalhos Relacionados

Na revisão da literatura sobre trabalhos relacionados às metodologias propostas neste trabalho e a problemática que ele tenta resolver foram encontrados trabalhos próximos, que se assemelham em métodos como (BRUEGGE et al., 2009; GRACIOLI NETO; VITERBO; KALINOWSKI, 2016) ou em problemática como (DINIZ et al., 2012; ASFORA; ALVES, 2009), servindo como referência e fornecendo uma contribuição relevante para a construção deste trabalho. Entretanto, não foram encontrados trabalhos que se propusessem a usar aprendizagem de máquina para priorização de requisitos de software ou casos de uso, sendo mais específico no âmbito ágil.

[2] Area under the *ROC curve*

Uma das principais limitações da priorização descrito pelas metodologias ágeis é a dificuldade de comparar a importância real dada aos requisitos e a falta de análise caso determinados requisitos não sejam selecionados. Mesmo em métodos ágeis as orientações fornecidas são bastante básicas sobre como conduzir essa priorização. Scrum usa a técnica de estimativa utilizando Valor de Negócio. Nesta técnica é muito comum que o cliente coloque vários requisitos senão todos com o valor de negócio máximo fazendo com que se perca a real noção do que realmente é mais importante (ASFORA; ALVES, 2009).

Uma das ferramentas utilizadas pela literatura para atender o problema da problematização da prioridade de requisitos é a técnica Kano. Essa técnica possibilita aos desenvolvedores de produtos transformar informações levantadas através de pesquisas em melhorias reais para o produto buscando a satisfação do cliente. No entanto, também é produzida muita informação e sua representação textual expressa em tabelas e relatórios, faz com que os responsáveis despendam muito tempo processando grandes volumes de dados. Além disso, a técnica é indicada para projetos com vários clientes ou usuários com realidades diferentes, sendo recomendado um volume entre 20 a 30 usuários para responder as perguntas funcionais e disfuncionais para obter a importância dado pelo usuário sobre os requisitos (DINIZ et al., 2012; ASFORA; ALVES, 2009)

Outro problema observado na técnica Kano é que quanto a visualização dos resultados ela é bastante insatisfatória, pois, é muito abstrato a escolha de quais dos requisitos vão ser priorizados. A técnica resulta em requisitos classificados em categoria. Cada categoria tem uma preferência para priorização e requisitos da mesma categoria são de difícil priorização (ASFORA; ALVES, 2009). Para contornar esse problema (ASFORA; ALVES, 2009) utiliza os percentuais que foram mais votados pelos usuários com gráficos para cada requisito. O que não é automático e gera mais volume de informações.

Uma alternativa para visualização de dados insatisfatória da técnica Kano foi o uso de TreeMaps, uma técnica de Visualização de Dados baseada em Árvore que explora conceitos básicos de ergonomia (DINIZ et al., 2012)

Quanto a metodologia usada destacam-se dois trabalhos relacionados (GRACIOLI NETO; VITERBO; KALINOWSKI, 2016; BRUEGGE et al., 2009)que pesquisaram sobre Aprendizagem de Máquina em problemáticas de engenharia de software, porém não especificamente para a questão de prioridade de requisitos. (GRACIOLI NETO; VITERBO; KALINOWSKI, 2016) comparou e analisou vários métodos de aprendizagem de máquina no uso de estimativas de esforço de software, enquanto (BRUEGGE et al., 2009) empregou aprendizagem de máquina para classificação de tarefas de software. Em ambos os trabalhos observa-se que o emprego de aprendizagem de má-

quina conseguiu resultados satisfatórios e (GRACIOLI NETO; VITERBO; KALINOWSKI, 2016) compara os resultados e mostra que as técnicas em geral tem boa acurácia nas bases utilizadas, além de mostrar bons resultados na revisão de trabalhos com aprendizagem de máquina e apontar para bases de dados empregadas como é o caso do PROMISEnota .

Ambos os trabalhos que propuseram soluções para priorização de requisitos usando a técnica Kano se mostraram insatisfatórias se considerados alguns aspectos como a visualização de informação para tomada de decisões e não levarem em consideração outros pontos importantes para os requisitos como riscos técnicos, experiência da equipe, complexidade e correlação entre requisitos. Em comparação com o modelo Volere, o Kano se mostra muito semelhante quanto a proposta de uma pergunta funcional e uma disfuncional para analisar o valor dado aos requisitos pelo cliente.

Os trabalhos que usaram aprendizagem de máquina demonstram como essa técnica pode ser eficiente e trazer bons resultados na área de Engenharia de Software. Eles indicaram resultados satisfatórios com completude em relação às bases de dados utilizadas e com consistência quanto a sua classificação de informação. Através deles é possível concluir com dados estatísticos e discussões sobre as técnicas utilizadas que Aprendizagem de Máquina é uma ferramenta poderosa e adaptável que pode ser explorada na área de Engenharia de Software em outros pontos, inclusive para priorização de requisitos.

Os trabalhos analisados e as bases de dados de Engenharia de Software recorrentes nos trabalhos estudados se caracterizaram como limitadas. Não abordaram fatores importantes que devem ser levados em consideração na priorização de requisitos, como já foi visto nos capítulos anteriores deste trabalho. As bases de dados observadas como as presentes no PROMISE são inadequadas por não trabalharem especificamente com dados em padrões ágeis e por não terem atributos essenciais para o modelo que este trabalho propõe.

3 METODOLOGIA

Esta seção tem como objetivo descrever como se deu o processo metodológico usado no desenvolvimento deste trabalho abordando a seleção e tratamento dos dados, as tecnologias empregadas (os algoritmos e as ferramentas), bem com a descrição, passo a passo, de todas as fases do processo de descoberta de conhecimento.

3.1 Coleta de Dados

Os dados analisados foram oriundos da aplicação de dois formulários *online* através da ferramenta Google Forms aplicados em equipes de desenvolvimento ágil que adotaram o método Scrum. Neste formulário foram alocadas perguntas referentes ao Projeto, Organização, Equipe (primeiro formulário) e Histórias de Usuários (segundo formulário) de acordo com a revisão da literatura apresentada. As perguntas foram construídas com a finalidade de capturar valores de riscos de software, complexidade, satisfação do cliente em receber, insatisfação do cliente em não receber e necessidade de aprendizagem da equipe. Seu objetivo em sua totalidade era o de capturar os valores mais relevantes para priorização de histórias de usuários servindo de base para a aprendizagem de máquina, alimentando os algoritmos a serem empregados. Os formulários, juntos, contêm 22 perguntas técnicas e se encontram no Apêndice I.

Foi pedido às equipes que após o termino dos projetos o time ágil fizesse uma revisão do desenvolvimento do projeto com as métricas de software fornecidas durante o preenchimento dos formulários e refletissem sobre a priorização inicial das histórias de usuários. Caso a equipe percebesse que poderiam ter realizado uma priorização mais adequada, depois de conhecerem melhor o projeto e terem contornados as dificuldades e retrabalhos não previstos, é capturada essa nova priorização para ser adotada nos experimentos e a antiga é ignorada.

No total, foram capturados pelos formulários 13 projetos e 172 US ao longo do período de um ano. Um destes projetos é oriundo do Centro de Estudos e Sistemas Avançados do Recife (CESAR - Recife). Os outros doze projetos são de equipes de desenvolvimento de softwares acadêmicos formadas por alunos da Universidade Federal Rural de Pernambuco (UFRPE) mas que apresentam o mesmo procedimento padrão de desenvolvimento ágil e tem aplicação útil para resolução de problemas reais.

3.2 Pré-Processamento dos Dados

O pré-processamento abrange as etapas de seleção, limpeza e transformação dos dados, sendo fundamental para obtenção de padrões e conhecimentos relevantes, gerados a partir da mineração de dados. Começa logo após a coleta de dados e é importante para que a etapa seguinte, a de extração de conhecimento, seja a mais efetiva possível. Estas fases do projeto tem como objetivo identificar e remover o ruído, que nada mais são do que inconsistências nas características das instâncias dos dados utilizados, ou até mesmo instâncias inteiras que destoam do restante da base, e removê-las para que a acurácia do modelo de AM não seja prejudicada durante a etapa de classificação. São tarefas que necessitam determinado conhecimento sobre o domínio no qual a base de dados atende (MORAES, 2016). A revisão da literatura do Capítulo 2 e a pesquisa para construção dos formulários serviu como estudo do domínio dos dados.

Após o levantamento de dados foi feita a unificação das respostas obtidas em uma única planilha e feita a conversão da mesma para o formato *Comma-separated Values (*CSV), arquivos de texto com um formato de terminador de linha, separando valores com vírgulas. O CSV é um dos formatos de arquivos de coleção de dados de entrada compatíveis com a ferramenta WEKA, usada na implementação e testes dos algoritmos de mineração de dados.

As atividades que foram realizadas nesta etapa são:

- Identificação de possíveis discrepâncias nos dados;
- Verificar possíveis inconsistências em valores numéricos e no preenchimento de dados;
- Transformação do conjunto de dados para padrões de entrada aceito pela Weka;
- Alteração dos nomes das colunas, perguntas dos formulários para um padrão minimalista (P1, P2, P3, etc. . .) a fim de evitar erros de leitura da Weka;
- Denominação da coluna “PRIORIDADE” para a coluna que representa a classe a ser usada pelos algoritmos;
- Normalização e discretização dos valores de priorização de prioridade por projeto.

As respostas oriundas do preenchimento dos formulários pelas equipes de desenvolvimento foram automaticamente alocadas em uma planilha online pelo *Google Forms* onde o título de cada coluna é representado por uma pergunta do formulário. Em um primeiro momento os dados foram analisados à procura de inconsistências, respostas duplicadas e diferenças na forma de escrita para mesmos valores como por exemplo “Sprint 01”, “Sprint 1”, “1”. Em seguida foram feitas as correções necessárias.

Para uso dos dados no Weka foi realizada a seleção dos atributos (colunas) de teor técnico, desconsiderando os atributos derivados dos formulários *online* que possuíam valor apenas de identificação e organização dos dados, como, por exemplo, nome do projeto, número da *Sprint* de cada história de usuário, hora de preenchimento e nome do projeto.

Após o primeiro tratamento de dados foram realizadas adaptações nos nomes das colunas, que representam a categorização dos atributos (perguntas de 1 ate 21) e da classe para predição (pergunta 22, a ordem de prioridade atribuída a cada US de cada projeto) para evitar erros de entrada e facilitar o uso dos dados na ferramenta Weka. Assim foi adotado o seguinte padrão para cada atributo: P+"número da pergunta", por exemplo: P1. A última coluna representa a classe, alvo da classificação e foi renomeada como "PRIORIDADE".

O formulário usa a escala *Likert* de escala de 5 pontos, com exceção da pergunta correspondente à complexidade das US, que usa a escala adotada pelo *Planning Poker* [2, 3, 5, 8, 13, 20] que foi mantida no pré-processamento, e a ordem de prioridade dada às US. Sendo assim as respostas que usaram a Escala *Likert* foram quantificadas para uso dos algoritmos de AM com a correspondência utilizadas na Tabelas 2 :

Tabela 2 – Quantificação da escala Likert de 5 pontos

Satisfação	**Insatisfação**	**Riscos/ Necessidade de Aprendizagem**	**Quantificação**
Extremamente Satisfeito	Extremamente Insatisfeito	Totalmente de acordo	5
Satisfeito	Insatisfeito	De acordo	4
Indiferente	Indiferente	Indiferente	3
Parcialmente Desinteressado	Parcialmente sem interesse	Em desacordo	2
Totalmente Desinteressado	Não se importa	Totalmente em desacordo	1

3.2.1 Normalização e Discretização da classe PRIORIDADE

Para que fosse realizado a hominização das escalas, os valores da classe PRIORIDADE passaram por normalização na escala de 0 até 1 por cada projeto, sendo 0 o valor da US que foi colocada no fim do *product backlog* e 1 o valor da US colocada no topo da lista que é aquela que deveria ser feito primeiro, ou seja, a de maior importância.

Após a normalização dos valores de prioridade foi feita uma categorização e discretização destes valores para as classes nominais (ALTA, MÉDIA e BAIXA). A discretização para valores nominais se faz necessária para a execução de classificadores que precisam trabalhar com classes nominais.

Tabela 3 – Discretização das faixas de valores normalizados do atributo classificador PRIORIDADE.

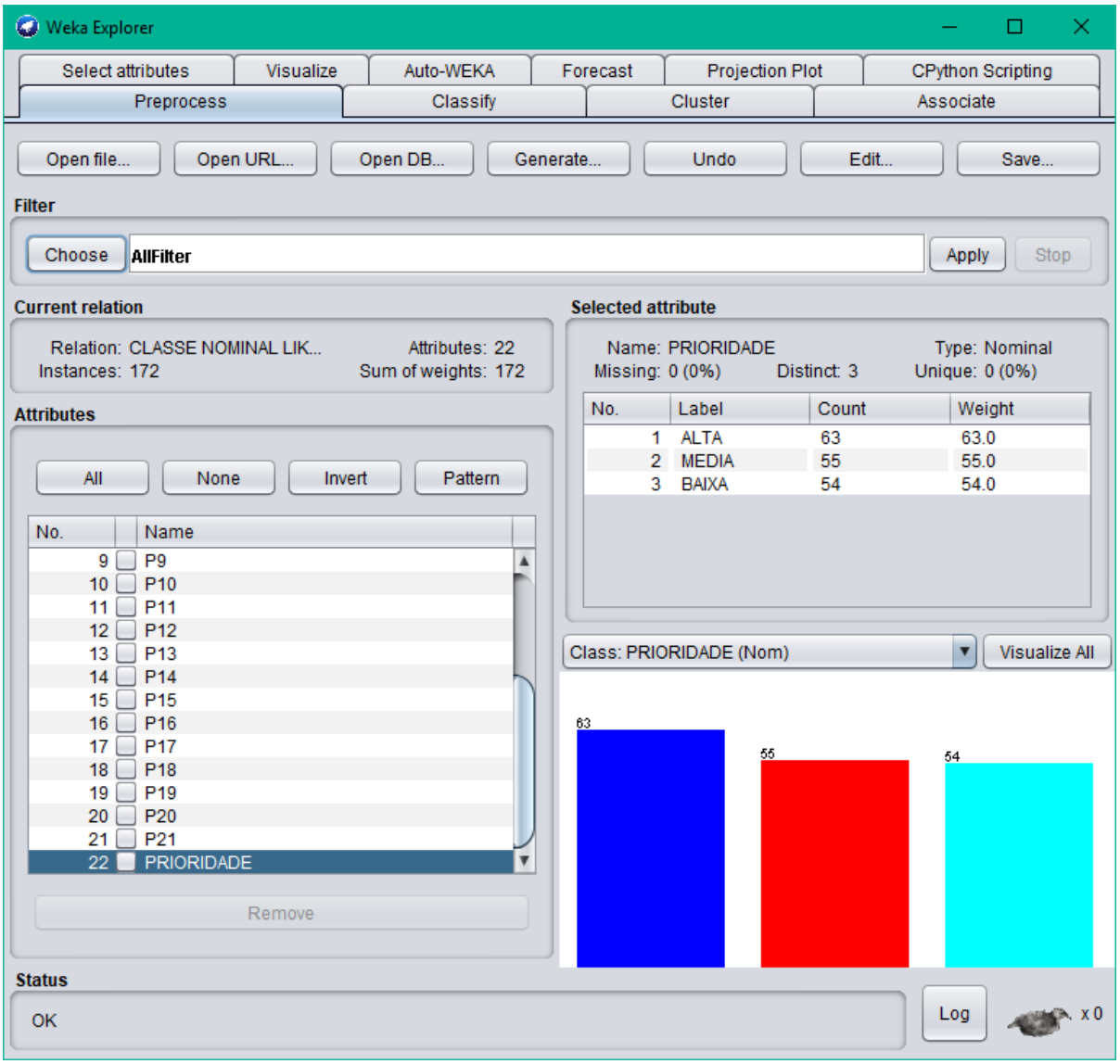
Faixa de valores normalizados do atributo classificador PRIORIDADE	Classificador (PIORIDADE)
0 até 0.39	BAIXA
0,4 até 0,69	MÉDIA
0,7 até 1	ALTA

Fonte: Autor

3.3 Balanceamento das classes

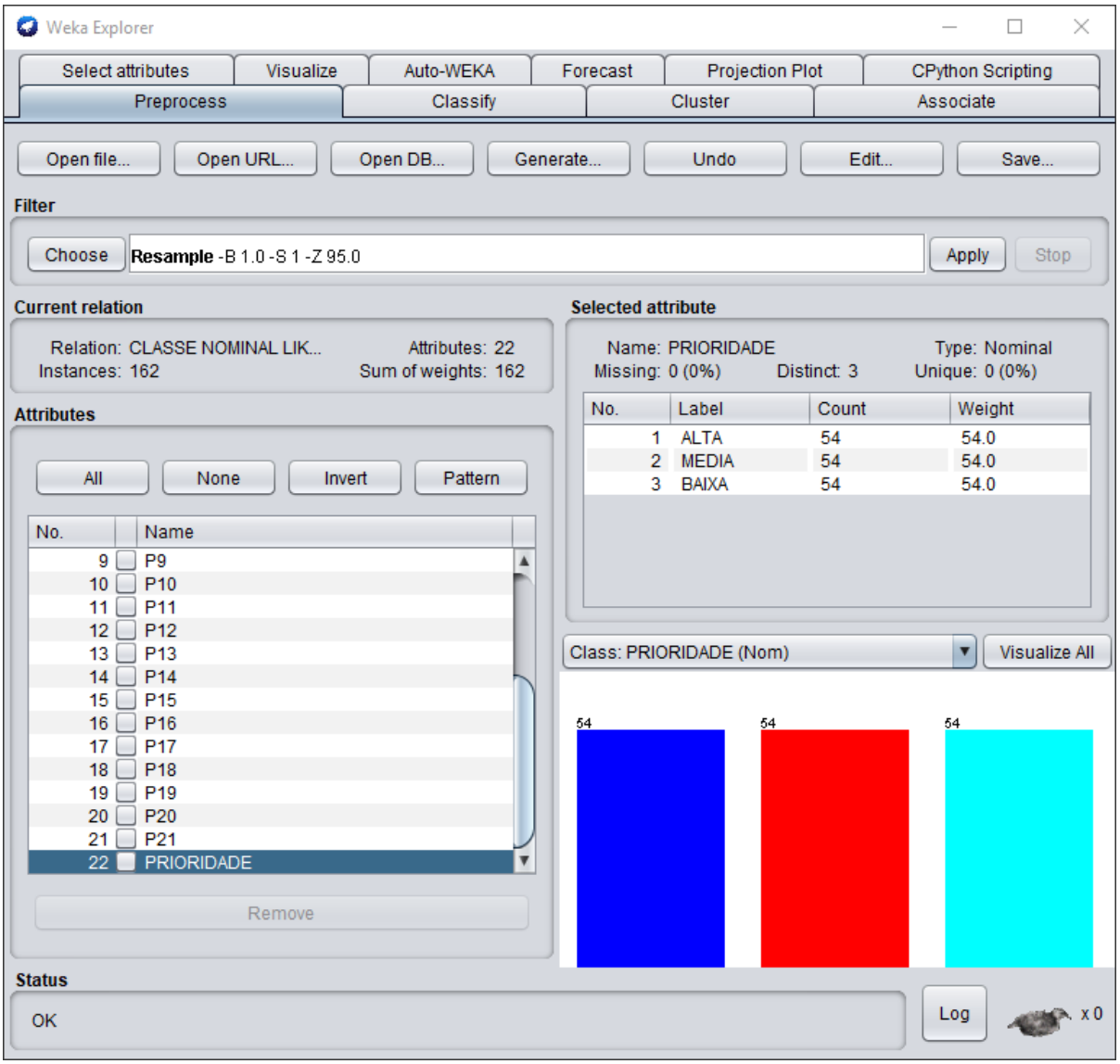
Para melhorar o desempenho dos algoritmos de aprendizagem de máquina foi utilizadq a técnica de subamostragem visando um aprendizado coerente dos algoritmos sem que quantidades diferentes de instâncias com classificações diferentes gerassem predições tendenciosas. Para realizar a subamostragem foi usado o filtro *Resample* do Weka na aba de pré-processamento, que pode criar subamostragem automática dos dados de forma uniforme. Antes de ser executado o filtro *Resample* a coleção de dados possuía 172 Histórias de Usuário (instâncias), das quais 63 eram de prioridade ALTA, 55 de prioridade MÉDIA e 54 de prioridade BAIXA. Um resumo dos dados pode ser observado na figura 7, cuja imagem foi retirada da suíte Weka antes do balanceamento da classe PRIORIDADE.

Figura 7 – Dados antes da subamostragem.



Após o balanceamento dos dados obteve-se uma distribuição uniforme do volume de dados para cada classificação, que passaram a ter 54 instâncias cada, totalizando 162 para toda a base. Na figura 8 pode-se observar a nova situação dos dados.

Figura 8 – Dados após aplicação do Resample para subamostragem.



3.4 Otimização dos algoritmos

Na busca por melhores resultados dos algoritmos foram realizadas alterações de alguns parâmetros quando possível e o uso do *Grid Search* quando permitido pelo algoritmo em questão. Em todos os testes, foi utilizado validação cruzada com *leave-one-out* para evitar *overfitting,* configuração plausível visto o tamanho reduzido do volume de dados. Assim, para as coleções de dados balanceadas, *k-folds* foi igual a 162 e para as coleções desbalanceadas, originais, *k-folds* foi igual a 172.

Não foi realizada nenhuma otimização com relação à seleção de atributos, pois isso implicaria em vários testes com uma análise delicada dada a fragilidade da coleção de dados disponível. Por possuir um volume de dados reduzido a construção

do embasamento necessário para realizar a exclusão de atributos, com pouco ou nenhum ganho de informação aos algoritmos, é prejudicada. Assim, em todos os testes foram utilizados 21 atributos para aprendizagem de máquina e 1 atributo classificador (PRIORIDADE).

4 Resultados e Discussões

Neste Capítulo serão mostrados os resultados dos testes realizados na suíte Weka com alguns algoritmos de aprendizado de máquina, bem como seus resultados e comparações. O foco destes testes é garantir a precisão na predição do modelo escolhido. Vale ressaltar que não foi feita uma gama extensiva de testes para verificar qual classificador teria a melhor predição. Considerando a compreensibilidade e aceitação pelos usuários finais, árvore de decisão foi o método escolhido para atender o problema. Neste capítulo pode-se averiguar melhor o seu desempenho.

4.1 Comparação com o método *Naive Bayes*

Realizar uma comparação com outra abordagem de aprendizado de máquina, mesmo que usar árvore de decisão seja a finalidade primordial, é interessante para aferir as relações de desempenho do classificador escolhido. Assim foi escolhido o algoritmo *Naive Bayes*, que é uma aproximação do Teorema de Bayes aplicado à AM. Para ele não há necessidades de otimização de parâmetros e qualquer tentativa de alteração dos mesmo não resultou em nenhum ganho significativo na predição das classes. Seu desempenho com e sem o balanceamento das classes está registrado na tabela 4:

Tabela 4 – Desempenho*Naive Bayes*

Teste	**Execução**	**Acertos (%)**	**Erros (%)**
I	*Naive Bayes* padrão	44,19	55,81
II	*Naive Bayes* com balanceamento	57,41	42,59

Fonte: autor

De forma resumida o *Naive Bayes* apresentou um desempenho insatisfatório como pode-se observar na tabela 4. O desempenho das árvores de decisão testadas foi superior.

4.2 Desempenho J48

Para o classificador J48 as configurações testadas que apresentaram o melhor desempenho foram: desativar a poda da árvore (P) e diminuir o número mínimo de instâncias em cada folha (M) para 0. Na tabela 5 podemos comparar o desempenho

antes e depois das alterações nos parâmetros da árvore de decisão e também a melhoria oferecida pelo balanceamento dos dados nos testes executados no Weka.

Tabela 5 – Desempenho J48

Teste	Execução	Acertos (%)	Erros (%)	Tamanho da Árvore
I	J48 padrão	50	50	59
II	J48 com balanceamento	76.54	23.46	51
III	J48 com otimização de parâmetros	50.58	49.42	129
IV	J48 com balanceamento e otimização de parâmetros	80.25	19.75	83

Fonte: autor

Apesar do bom desempenho do teste IV observa-se que a árvore gerada é relativamente grande (83) além de possuir 42 níveis. Sua ramificação e tamanho são compreensíveis visto que a árvore não foi construída com poda. É notável o ganho de desempenho médio dos testes no conjunto de treino com o balanceamento dos dados.

4.3 Desempenho *Random Forest*

Como já foi abordado, o Random Forest dispensa a maioria das configurações de parâmetros. Os testes foram realizados no Weka comparando o desempenho do *Random Forest* padrão usando a base com e sem balanceamento de dados (Tabela 6).

Tabela 6 – Desempenho Random Forest

Teste	Execução	Acertos (%)	Erros (%)
I	Sem balanceamento	55, 23	44,77
II	Com balanceamento	81,48	18,52

Fonte: autor

Novamente o balanceamento apresentou um ganho considerável na predição do algoritmo em questão. Entretanto o *Random Forest* não gera uma única árvore e sim um conjunto, uma floresta, portanto não podemos avaliar diretamente as ramificações até a tomada de decisão e classificação de cada instância como o algoritmo J48.

4.4 Desempenho *REPTree*

Com o emprego do *Grid Search* foram testadas várias configurações de parâmetros e algumas delas melhoraram o desempenho do classificador. Estes parâmetros foram:

- *maxDepth*: profundidade máxima alterada para 15;
- *minVarianceProp:* variância mínima para divisão alterada para 0,03;
- *noPruning*: não-poda alterado para falso.

A tabela a seguir mostra o desempenho do *REPTree* em alguns testes realizados no Weka:

Tabela 7 – Desempenho REPTree

Teste	**Execução**	**Acertos (%)**	**Erros (%)**	**Tamanho da Árvore**
I	*REPTree* padrão	37,21	62,79	31
II	*REPTree* com balanceamento	62,96	37,04	19
III	*REPTree* com otimização de parâmetros	47,67	52,33	71
IV	*REPTree* com balanceamento e otimização de parâmetros	77,78	22,22	57

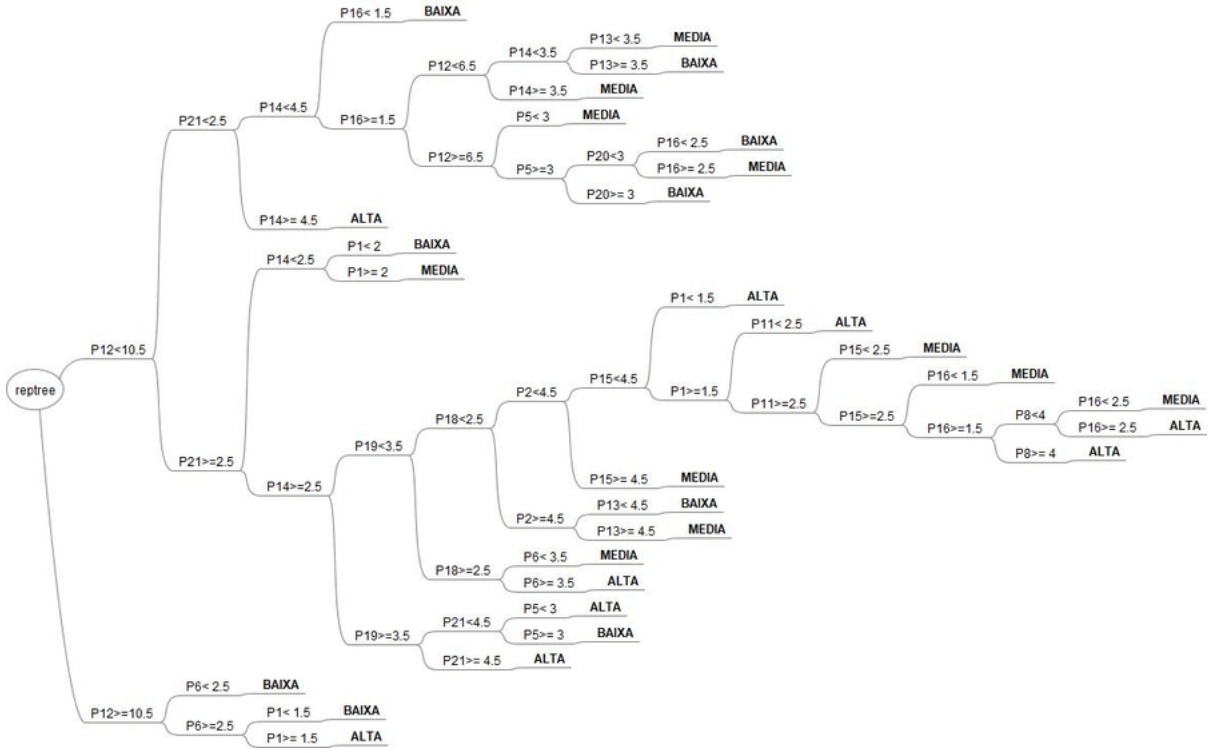
Fonte: autor

O melhor resultado com o REPTree é o teste IV que além de ter a melhor predição não possui um tamanho tão extenso quanto o Teste III e outras árvores montadas por outros algoritmos, como já foi representado. Pode-se dizer que é um caso de desempenho e compreensibilidade oferecida razoáveis, em relação aos demais testes. Por esse motivo o teste IV foi escolhido para representar a solução ao problema apresentado neste trabalho, a seguir mais detalhes sobre ele serão apresentados.

4.4.1 Árvore de Decisão obtida com REPTree

A representação gráfica da árvore de decisão gerada pelo modelo escolhido através do classificador REPTree pode ser observado na Figura 9. A árvore em modo textual e os demais dados e métricas gerados pelo Weka estão disponíveis no Apêndice II.

Figura 9 – Representação gráfica da árvore de decisão gerada pelo REPTree.



4.4.2 Análise da Matriz de Confusão e da Curva ROC

A execução do modelo no Weka fornece várias métricas em seu resultado incluindo dados da matriz confusão de métricas que ajudam a perceber a eficiência do classificador escolhido, como pode-se ver Figura 10.

Figura 10 – Detalhes sobre métricas de desempenho do classificador REPTree no Weka.

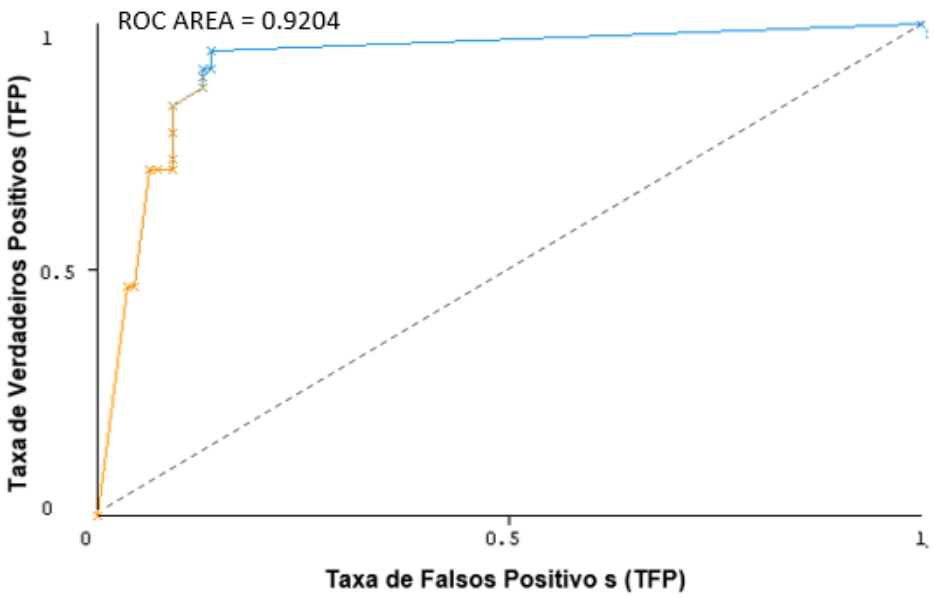
```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,870    0,130    0,770      0,870    0,817      0,721    0,920     0,812     ALTA
                 0,741    0,148    0,714      0,741    0,727      0,587    0,828     0,709     MEDIA
                 0,722    0,056    0,867      0,722    0,788      0,702    0,882     0,761     BAIXA
Weighted Avg.    0,778    0,111    0,784      0,778    0,778      0,670    0,877     0,761

=== Confusion Matrix ===

  a  b  c   <-- classified as
 47  6  1 |  a = ALTA
  9 40  5 |  b = MEDIA
  5 10 39 |  c = BAIXA
```

Fonte: Autor

Entre as métricas disponibilizadas na saída de informações da execução do Weka, temos: *TP Rate* (Taxa de Verdadeiros Positivos), *FP Rate* (Taxa de Falsos

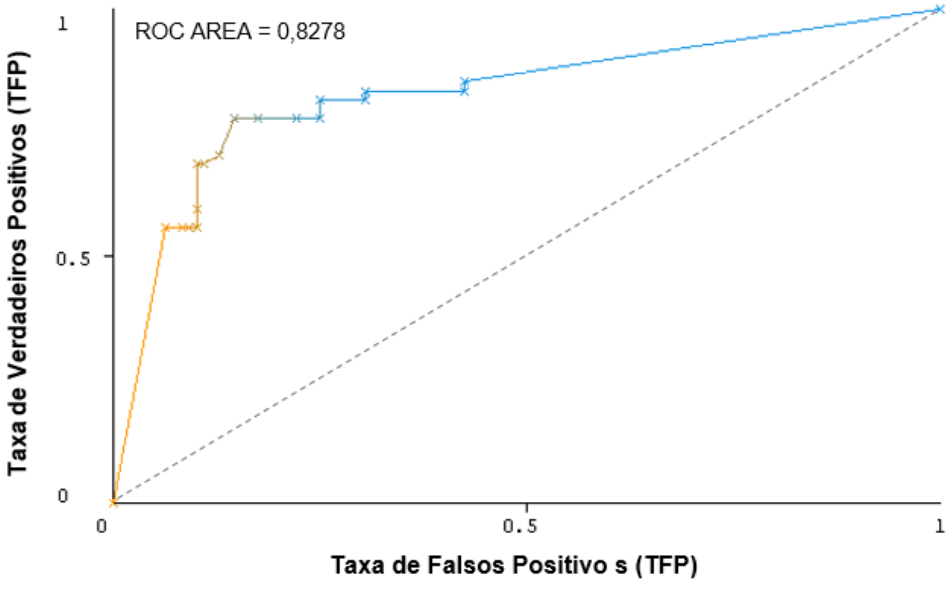
Positivos) e *ROC Area*. São métricas que fornecem impressões sobre a confiança e sobre a capacidade de predição do modelo criado como foi visto no Capítulo 2. Pode-se dizer que o modelo em questão possui taxas de predição satisfatórias. Com a finalidade de averiguar o desempenho da predição do modelo com uma boa visualização, também foram gerados gráficos de curvas *ROC* para cada atributo classificador, conforme figuras de 11 a 13.

Figura 11 – Curva ROC - Classe ALTA



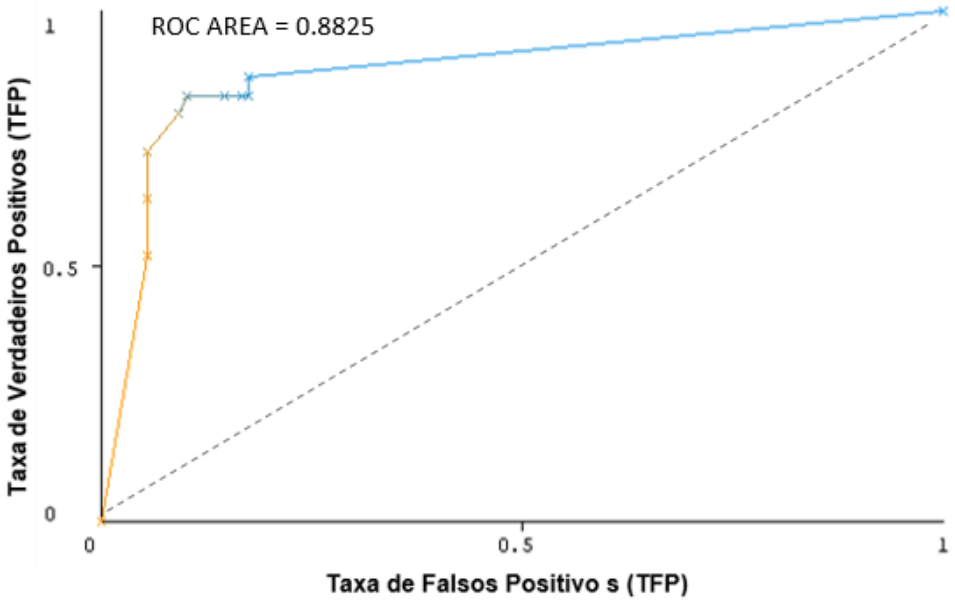
Fonte: Autor

Figura 12 – Curva ROC - Classe MÉDIA



Fonte: Autor

Figura 13 – Curva ROC - Classe BAIXA



Fonte: Autor

Como já foi explanado, o desempenho do modelo pode ser analisado em relação a sua proximidade com o canto superior esquerdo (situação ideal de classificação). A linha diagonal que corta o gráfico, representa a probabilidade de uma classificação aleatória. Modelos que são classificados acima da linha são tidos como bons, e abaixo da linha são descartados (CARVALHO, 2014). Desta forma além da validação cruzada e das taxas de acertos, a matriz confusão e os gráficos de curva ROC contribuem para a aceitação do modelo proposto. Pelas métricas de saída e pelos gráficos de *curvas ROC* pode-se observar que o classificador mantém um padrão muito próximo para as três classes indicando, provavelmente, que o classificador não possui tendências classificatórias maiores para uma classe e menores para outras.

5 CONSIDERAÇÕES FINAIS

Em linhas gerias, acredita-se que o trabalho alcançou o seu objetivo: apresentar uma proposta de priorização inteligente de histórias de usuários que não considere apenas complexidade e priorização sentimental do cliente como dados de entrada para escolha de cada prioridade. Além de também entregar um ensaio sobre uma ferramenta que poderia auxiliar equipes de desenvolvimento ágil, principalmente as imaturas, na realização de uma priorização mais abrangente que tem como objetivos a economia de tempo e recursos à medida que ganha em entrega de valor de negócio.

Na primeira parte da construção da ferramenta de priorização foi notado que a dificuldade no levantamento de dados e, por consequência, a falta de confiança nos ajustes dos algoritmos levaria a uma ferramenta incompleta. A distribuição dos dados disponíveis é insuficiente para que o modelo escolhido possa identificar um número razoável de comportamentos e, assim, não deve lidar de forma eficiente com novas situações reais de predição. Entretanto é notório que os ajustes de parâmetros, balanceamentos de classes e, o uso de validação cruzada com *leave-one-out* foram técnicas ideais para o volume e tipo de dados usados, surtindo efeito na otimização do modelo além de evitar *underfitting* e *overfitting*.

Considerando todos os resultados obtidos, é possível observar que o algoritmo de árvore de decisão funciona de forma satisfatória, sendo um classificador poderoso capaz de apresentar os resultados de forma simples e clara, diferentemente de outros algoritmos conhecidos de AM, visto que elas possuem boas condições de compreensibilidade. É esperado que o uso de árvore de decisão seja um facilitador da adoção de ferramentas de apoio a decisão quanto à priorização de histórias de usuários, pois mesmo não sendo o algoritmo mais eficiente ele atende as necessidades da ferramenta proposta. Para trabalhos futuros, sugere-se um estudo mais profundo do levantamento de dados, de modo a encontrar as características necessárias para uma Mineração de Dados mais adequada do que a utilizada neste trabalho.

6 SUGESTÕES PARA TRABALHOS FUTUROS

Dado a falta de bases de dados específicas de grandes dimensões disponíveis em repositórios públicos e as dificuldades em se implementar formulários extensos em equipes de trabalho, seria útil realizar o levantamento desses dados através da criação de um histórico de desenvolvimento ágil em uma ou mais empresas de desenvolvimento de software. Nelas poderia-se analisar a eficiência das priorizações apresentadas por projeto, dificuldades não planejapdas e fatores que levaram a algum tipo de retrabalho. Assim seria possível criar algoritmos em condições melhores de aprendizagem de máquina, já que neste trabalho a limitação da base de dados foi crucial no desempenho apresentado.

É importante ressaltar que a falta de pesquisas similares levou a um levantamento de atributos baseado nos principais fatores de risco de desenvolvimento de *software* apresentados na revisão da literatura deste trabalho. Uma maneira de calibrar melhor os algoritmos para obter melhores resultados é realizando otimização de atributos, o que devido aos poucos dados e tempo disponíveis não foi possível neste trabalho.

Tendo disponível uma base de tamanho relevante e dados reais consolidados gerados a partir de um histórico organizado de desenvolvimento de *software*, outra opção de pesquisa futura seria a criação de um algoritmo ou *framework* para priorização inteligente de histórias de usuários. Essa ferramenta após ser alimentada com informações sobre complexidade, riscos de *software* e valor de negócio (no modelo Volere, de satisfação e insatisfação) indicaria uma sugestão de priorização de histórias de usuários baseado na aprendizagem de máquina e nas predefinições que induziriam a equipe a levar em consideração as métricas de *software* mais abrangentes e menos usuais no mercado, expostas neste trabalho.

Referências

ALMEIDA, B. A. E. de. *Mineração de Dados Aplicada em Engenharia de Software*. 2017. Disponível em: <https://pt.slideshare.net/BrunoAlisson/minerao-de-dados-aplicada-em-engenharia-de-software>. Acesso em: 05/05/2018.

ANTINYAN, V. et al. Defining technical risks in software development. In: IEEE, 2014. *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*. [S.l.], 2014. p. 66 – 71.

ANWER, M. U. et al. Natural variation reveals that intracellular distribution of ELF3 protein is associated with function in the circadian clock. *eLife*, eLife Sciences Publications, Ltd, v. 3, p. e02206 –, 2014. ISSN 2050-084X. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4071560/>.

ARNUPHAPTRAIRONG, T. Top ten lists of software project risks: Evidence from the literature survey. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. [S.l.: s.n.], 2011. v. 1, p. 1 – 6.

ASFORA, D. M.; ALVES, C. F. *Uma abordagem para priorização de requisitos em ambientes ágeis*. 2009. Dissertação (Mestrado) — Universidade Federal de Pernambuco. Disponível em: <http://repositorio.ufpe.br/handle/123456789/2264>.

BASSI FILHO, D. L. *Experiências com desenvolvimento ágil*. 2008. Dissertação (Mestrado) — Universidade de São Paulo. Disponível em: <https://www.ime.usp.br/~colli/Arquivos_linkados/Dissertacao_Dairton_banca.pdf>. Acesso em: 19 nov. de 2017.

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

BEZERRA, E. *Princípios de Análise e Projeto de Sistemas com UML* . 3. ed. [S.l.]: Campus, 2015.

BHALERAO, S.; INGLE, M. Incorporating Vital Factors in Agile Estimation through Algorithmic Method. *IJCSA*, v. 6, n. 1, p. 85 – 97, 2009. Disponível em: <http://www.tmrfindia.org/ijcsa/v6i17.pdf>.

BRUEGGE, B. et al. Classification of tasks using machine learning. In: ACM, 2009. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. [S.l.], 2009.

CARVALHO, B. V. de; MELLO, C. H. P. Implementation of scrum agile methodology in software product project in a small technology-based company. *Gestão & Produção*, SciELO Brasil, v. 19, n. 3, p. 557 – 573, 2012.

CARVALHO, H. M. *Aprendizado de máquina voltado para mineração de dados: árvores de decisão*. 2014. 100 p. Monografia (Bacharelado em Engenharia de Software) — Universidade de Brasília - UnB.

CAWLEY, G. C.; TALBOT, N. L. Efficient leave-one-out cross-validation of kernel fisher discriminant classifiers. *Pattern Recognition*, Elsevier, v. 36, n. 11, p. 2585 – 2592, 2003.

COHN, M. *Agile estimating and planning*. [S.l.]: Pearson Education, 2005.

CRAIG, I. D. Principles of Software Engineering Management by Tom Gilb with Susannah Pinzi, Addison-Wesley, Wokingham, UK, 442 pages (incl. index) (£17.95). *Robotica*, v. 7, n. 2, p. 175 –, 1989. Disponível em: <http://dx.doi.org/10.1017/S0263574700005671>.

CTIC-UFPA. *"Procedimentos do Planning Poker", versão 1.3.* [S.l.: s.n.], 2011. Centro de Tecnologia da Informação e Comunicação da Universidade Federal do Pará.

DEVASENA, C. L. Comparative Analysis of Random Forest, REP Tree and J48 Classifiers for Credit Risk Prediction. *International Conference on Communication, Computing and Information Technology (ICCCMIT)*, 2014.

DINIZ, F. A. et al. Aplicação de uma técnica de visualização de dados baseado em árvores para auxiliar a priorização de requisitos em projetos ágeis. In: *II Workshop Brasileiro de Visualização de Software*. Mossoró-RN: [s.n.], 2012. p. 1 – 8.

FERREIRA JUNIOR, A. et al. Análise das práticas Scrum desempenhadas em uma empresa de grande porte do setor de tecnologia da informação. *Iberoamerican Journal of Project Management*, v. 7, n. 1, p. 37 – 46, 2016.

FONSECA da S. W. D. Ferramenta para apoio à estimativa baseada em Planning Poker utilizando a metodologia Scrum. UFPE – Recife. Junho 2008. Disponível em: <http://www.cin.ufpe.br/~tg/2012-1/dwas.pdf>.

FÜRNKRANZ, J.; GAMBERGER, D.; LAVRAČ, N. *Foundations of rule learning*. [S.l.]: Springer Science & Business Media, 2012.

GAIKWAD, V.; JOEG, P. An empirical study of writing effective user stories. *International Journal of Software Engineering and Its Applications*, v. 10, n. 11, p. 387 – 404, 2016.

GOMES, A. E. *Métricas e Estimativas de Software: O início de um rally de regularidade*. 2013. Disponível em: <http://www.apinfo.com/artigo44.htm>.

GOMES, T. C. S. *Descoberta de Conhecimento Utilizando Mineração de Dados Educacionais Abertos*. 2015. 67 p. Monografia (Bacharelado em Sistemas de Informação) — Universidade Federal Rural de Pernambuco.

GRACIOLI NETO, C.; VITERBO, J.; KALINOWSKI, M. Um Levantamento do Uso de Técnicas de Aprendizado de Máquina na Estimativa de Esforço de Software. 2016.

GRENNING, J. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, v. 3, 2002.

HARTMANN, D.; DYMOND, R. Appropriate agile measurement: Using metrics and diagnostics to deliver business value. *Proceedings - AGILE Conference, 2006*, v. 2006, p. 126 – 131, 2006.

HAZAN, C. Análise de Pontos de Função: Uma aplicação nas estimativas de tamanho de Projetos de Software. *Engenharia de Software Magazine, Edição*, v. 2, p. 25 – 31, 2008.

JAQUEIRA, A. de O. P. et al. *Uso de modelos i* para enriquecer requisitos em métodos ágeis*. 2013. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Norte. Disponível em: <http://repositorio.ufrn.br:8080/jspui/handle/123456789/18077>.

JOHNSON, J. The Chaos Report. West Yarmouth: The Standish Group International. 2015.

KLEINING, G. Umriss zu einer Methodologie qualitativer Sozialforschung. *Kölner Zeitschrift für Soziologie und Sozialpsychologie*, Deutschland, v. 34, n. 2, p. 224 – 253, 1982.

LARMAN, C. *Utilizando UML e padrões*: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. Terceira. Porto Alegre: Bookman, 2007.

LEE, H. D. *Seleção e construção de features relevantes para o aprendizado de máquina.* 2000. Tese (Doutorado) — Universidade de São Paulo.

LONGO, H. E. R.; SILVA, M. P. A Utilização de Histórias de Usuários no Levantamento de Requisitos Ágeis para o Desenvolvimento de Software. *International Journal of Knowledge Engineering and Management (IJKEM)*, v. 3, n. 6, p. 1 – 30, 2014.

MAHNIC, V.; HOVELJA, T. On using planning poker for estimating user stories. *Journal of Systems and Software*, v. 85, n. 9, p. 2086 – 2095, 2012. Disponível em: <http://dx.doi.org/10.1016/j.jss.2012.04.005>.

MAIDEN, N. Improve your requirements: Quantify them. *IEEE Software*, IEEE, v. 23, n. 6, 2006.

MANIFESTO ÁGIL. *O Manifesto para Desenvolvimento Ágil de Software*. 2001. Disponível em: <http://www.agilealliance.org/pt/o-manifesto/>.

MANSANI, A. S. F.; RIBAS, M. S.; PALUDO, M. A. Processo de requisitos: uma abordagem para empresas de pequeno e médio porte. 2008.

MARÇAL, A. S. C. et al. Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI. *CLEI Electronic Journal*, 2007.

MARTINS, J. Metodologia ágil – Framework Scrum em gerenciamento de projetos de software. *COGNITIO/PÓS-GRADUAÇÃO UNILINS*, v. 1, n. 7, 2016.

MELIA, M. T.; AUWAERTER, P. G. Time for a Different Approach to Lyme Disease and Long-Term Symptoms. *The new england journal o f medicine*, p. 1277 – 1278, Março 2016.

MOLOKKEN-OSTVOLD, K.; HAUGEN, N. C. Combining estimates with planning poker–an empirical study. In: IEEE, 2007. *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*. [S.l.], 2007. p. 349 – 358.

MORAES, L. de O. F. *Classificação de linfomas utilizando uma abordagem baseada em árvores de decisão*. 2016. Tese (Doutorado) — Universidade Federal do Rio de Janeiro.

ONE, V. *The 11th annual State of Agile™ Report*. [S.l.], 2017. Disponível em: <https://www.versionone.com/pdf/VersionOne-11th-Annual-State-of-Agile-Report.pdf>.

PERES, H. *Automatizando testes de software com Selenium*. [S.l.]: Simplissimo Livros Ltda, 2016.

PMI. *Um guia do conhecimento em gerenciamento de projetos (Guia PMBOK®)*. 5. ed. [S.l.]: PMI, 2013. Acesso em: 23/05/2017.

PORTHIN, M. Advanced Case Studies in Risk Management. *System*, v. 39, p. 123 – 127, 2004.

PRESSMAN, R. S. *Engenharia de Software - Uma abordagem profissional*. [S.l.]: AMGH Editora LTDA, 2011.

RAM, P. *Agile Metrics: A seminal approach for calculating Metrics in Agile Projects*. 2013. Disponível em: <http://www.articlesbase.com/project-management-articles/agile-metrics-a-seminal-approach-for-calculatingmetrics-in-agile-projects-3130252.html>.> Acesso em: 20/03/2018.

RITTER, R. Técnicas de Abordagem de Estimativa Ágil: Planning Poker e Ideal Day. 2006. Disponível em: <http://www.rogerritter.com.br/planning-poker-e-ideal-day-tecnicas-de-abordagem-de-estimativa-agil/>.

ROBERTO, A. A.; BARBOSA, A. L. Metodologias ágeis: auxiliando o processo de desenvolvimento de software de pequenas e médias empresas. *Revista Tecnológica da Fatec Americana*, v. 3, n. 1, 2016.

ROBERTSON, J.; ROBERTSON, S. *Volere Requirements Specification Template*. 16. ed. [S.l.]: Tech. Rep., Atlantic Systems Guild, 2012.

RODRIGUES, R. L. et al. A literatura brasileira sobre mineração de dados educacionais. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*. [S.l.: s.n.], 2014. v. 3, n. 1.

ROPPONEN, J.; LYYTINEN, K. Can software risk management improve system development: an exploratory study. *European Journal of Information Systems*, Palgrave Macmillan, v. 6, n. 1, p. 41 – 50, 1997.

SANTOS JUNIOR, A. B. dos; BISPO, F. C. da S.; MOURA, L. S. A Gestão da Aprendizagem nas Organizações. In: *IV Simpósio de Excelência em Gestão e Tecnologia*. Resende: [s.n.], 2007.

SANTOS, M. A. G. et al. Benefícios da aplicação do método scrum no desenvolvimento de software em uma pequena empresa de base tecnológica. In: *XXXVI Encontro Nacional de Engenharia de Producão*. João Pessoa/PB: [s.n.], 2016.

SCHWABER, K.; SUTHERLAND, J. Guia do Scrum–Um guia definitivo para o Scrum: As regras do jogo. *http://www. scrumguides. org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR. pdf>. Citado*, v. 3, 2013.

SERRA, R. Gestão ágil de projetos de software versus PMBOK. *Revista de Ciências Exatas e Tecnologia*, v. 7, n. 7, p. 141 – 157, 2015.

SHWABER, K.; SUTHERLAND, J. Guia do Scrum–Um guia definitivo para Scrum: As regras do jogo, out. 2011. *Acessado em*, v. 23, 2014.

SILVA, D. E. dos S.; SOUZA, I. T. de; CAMARGO, T. Metodologias Ágeis Para O Desenvolvimento De Software: Aplicação E O Uso Da Metodologia Scrum Em Contraste Ao Modelo Tradicional De Gerenciamento De Projetos. *Revista Computação Aplicada-UNG*, v. 2, n. 1, p. 39 – 46, 2013.

SILVA, S. da; BONIN, M. R.; PALUDO, M. A. Levantamento de requisitos segundo o método volere. *http://publica. fesppr. br/index. php/rnti/article/viewFile/v1n1ART2/86*>, v. 1, n. 07, 2005.

SOTILLE, M. PMBOK & CMM + CMMI. *Porto Alegre*, 2003. Disponível em: <http://www.pmtech.com.br>. Acesso em: 20/11/2017.

SOUZA, M. M. de; MOURA, H. P. de. *Uma proposta para aplicar análise quantitativa de riscos em projetos de software ágeis*. 2010. Dissertação (Mestrado) — Universidade Federal de Pernambuco. Disponível em: <http://repositorio.ufpe.br/handle/123456789/2317>.

TURNER, J. R.; MÜLLER, R. On the nature of the project as a temporary organization. *International journal of project management*, Elsevier, v. 21, n. 1, p. 1 – 8, 2003.

USMAN, M. et al. Effort estimation in agile software development: a systematic literature review. In: ACM, 2014. *Proceedings of the 10th International Conference on Predictive Models in Software Engineering*. [S.l.], 2014. p. 82 – 91.

WAIKATO, M. L. G. at the University of. *Weka 3: Data Mining Software in Java*. Disponível em: <https://www.cs.waikato.ac.nz/ml/weka/>. Acesso em: 26/07/2018.

WEISS, S. M.; KULIKOWSKI, C. A. *Computer Systems That Learn*: Classification and prediction methods from statistics, neural nets, machine learning and expert systems (machine learning series). 1. ed. [S.l.]: Kaufmann Publishers, Inc, 1990.

Apêndices

APÊNDICE I

Perguntas dos Formulários Aplicados em Equipes de Desenvolvimento Ágil de Software.

Organização

- Existem Políticas Organizacionais com repercussão negativa sobre o projeto. (Organização pode-se entender como a entidade a qual a equipe atende e faz parte: empresa, startup, Universidade, etc)?
- Ambiente Organizacional instável ou em fase de reestruturação/mudanças?

Projeto

- O projeto faz uso de tecnologias imaturas (relativamente novas/não consolidadas)?
- O nível de complexidade técnica do projeto levando em consideração o processamento de dados e performance é ALTO?
- O nível de complexidade técnica do projeto levando em consideração necessidades de segurança é ALTO?
- O projeto faz uso de tecnologias nunca usadas em projetos anteriores?
- O projeto conta com muitas ramificações e integrações de código ao ramo de desenvolvimento principal?

Equipe

- Existem membros inexperientes na equipe?
- Membros da equipe não possuem habilidades específicas exigidas pelo Projeto?
- A equipe conta com membros distribuídos geograficamente/em teletrabalho?
- O gerente/líder de equipe é experiente?

Histórias de Usuário (US)

- O quão complexo a equipe considera essa história (Usando a escala comum de *Planning Poker*: 2, 3, 5, 8, 13 ou 20 pontos)?
- Qual o grau de Satisfação do cliente em receber o requisito implementado?
- Qual o grau de Insatisfação do cliente se o requisito não for implementado?
- O desenvolvimento da história irá contar com o comprometimento do cliente para solucionar qualquer dúvida ou problema que possa surgir?
- Este requisito contém pseudo-código, referências a outros documentos ou requisitos, múltiplos desejos, etc., o que pode diminuir sua compreensão do ponto de vista semântico e pode tornar a história mais difícil de desenvolver e testar?
- Esta história possui um amplo acoplamento com outros requisitos que podem torná-lo vulnerável a fatores externos (as mudanças tardias desses requisitos são prováveis)?
- Requisito pouco claro ou não testável. Há a probabilidade de que este requisito seja difícil de entender e implementar devido à descrição sintática obscura?
- Este requisito será alterado frequentemente por causa de objetivos de negócios fortes, o que pode causar risco de entrega tardia do produto?
- Essa história possui dependências ocultas/incertas indesejadas ou de componentes externos que podem criar problemas de manutenção e desempenho em tempo real?
- Existem histórias que esperam ou dependem da conclusão desta história?
- Qual a ordem de prioridade dada a essa história no P*roduct Backlog*? Exemplo: 1 = a que deve ser feita primeiro, a de maior prioridade) e assim sucessivamente?

APÊNDICE II

Melhor Árvore de Decisão gerada pelo algoritmo REPTree

Scheme: weka.classifiers.trees.REPTree -M 2 -V 0.03 -N 3 -S 1 -L 15 -P -I 0.0 -num-decimal-places 100

Relation: CLASSE NOMINAL LIKERT 1-5 CLASS AMB-weka.filters.supervised.instance.Resample-B1.0-S1-Z95.0

Instances: 162

Attributes: 22

P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
P16
P17
P18
P19
P20
P21
PRIORIDADE

Test mode: 162-fold cross-validation

=== Classifier model (full training set) ===

REPTree

==========

P12 $<$ 10.5

```
|  P21 < 2.5
|  |  P14 < 4.5
|  |  |  P16 < 1.5 : BAIXA (5/0) [0/0]
|  |  |  P16 >= 1.5
|  |  |  |  P12 < 6.5
|  |  |  |  |  P14 < 3.5
|  |  |  |  |  |  P13 < 3.5 : MEDIA (5/0) [0/0]
|  |  |  |  |  |  P13 >= 3.5 : BAIXA (6/1) [0/0]
|  |  |  |  |  P14 >= 3.5 : MEDIA (15/0) [0/0]
|  |  |  |  P12 >= 6.5
|  |  |  |  |  P5 < 3 : MEDIA (2/0) [0/0]
|  |  |  |  |  P5 >= 3
|  |  |  |  |  |  P20 < 3
|  |  |  |  |  |  |  P16 < 2.5 : BAIXA (2/0) [0/0]
|  |  |  |  |  |  |  P16 >= 2.5 : MEDIA (3/1) [0/0]
|  |  |  |  |  |  P20 >= 3 : BAIXA (5/0) [0/0]
|  |  P14 >= 4.5 : ALTA (3/0) [0/0]
|  P21 >= 2.5
|  |  P14 < 2.5
|  |  |  P1 < 2 : BAIXA (7/0) [0/0]
|  |  |  P1 >= 2 : MEDIA (3/0) [0/0]
|  |  P14 >= 2.5
|  |  |  P19 < 3.5
|  |  |  |  P18 < 2.5
|  |  |  |  |  P2 < 4.5
|  |  |  |  |  |  P15 < 4.5
|  |  |  |  |  |  |  P1 < 1.5 : ALTA (21/1) [0/0]
|  |  |  |  |  |  |  P1 >= 1.5
|  |  |  |  |  |  |  |  P11 < 2.5 : ALTA (7/0) [0/0]
|  |  |  |  |  |  |  |  P11 >= 2.5
|  |  |  |  |  |  |  |  |  P15 < 2.5 : MEDIA (3/0) [0/0]
|  |  |  |  |  |  |  |  |  P15 >= 2.5
|  |  |  |  |  |  |  |  |  |  P16 < 1.5 : MEDIA (2/0) [0/0]
|  |  |  |  |  |  |  |  |  |  P16 >= 1.5
|  |  |  |  |  |  |  |  |  |  |  P8 < 4
|  |  |  |  |  |  |  |  |  |  |  |  P16 < 2.5 : MEDIA (3/0) [0/0]
|  |  |  |  |  |  |  |  |  |  |  |  P16 >= 2.5 : ALTA (3/0) [0/0]
|  |  |  |  |  |  |  |  |  |  |  P8 >= 4 : ALTA (9/0) [0/0]
|  |  |  |  |  |  P15 >= 4.5 : MEDIA (2/0) [0/0]
```

| | | | | P2 >= 4.5
| | | | | | P13 < 4.5 : BAIXA (2/0) [0/0]
| | | | | | P13 >= 4.5 : MEDIA (3/0) [0/0]
| | | | P18 >= 2.5
| | | | | P6 < 3.5 : MEDIA (12/2) [0/0]
| | | | | P6 >= 3.5 : ALTA (2/0) [0/0]
| | | P19 >= 3.5
| | | | P21 < 4.5
| | | | | P5 < 3 : ALTA (2/0) [0/0]
| | | | | P5 >= 3 : BAIXA (10/1) [0/0]
| | | | P21 >= 4.5 : ALTA (6/1) [0/0]
P12 >= 10.5
| P6 < 2.5 : BAIXA (14/0) [0/0]
| P6 >= 2.5
| | P1 < 1.5 : BAIXA (3/1) [0/0]
| | P1 >= 1.5 : ALTA (2/0) [0/0]

Size of the tree : 57

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===

Sumary

Correctly Classified Instances	126	77.7778 %
Incorrectly Classified Instances	36	22.2222 %
Kappa statistic	0.6667	
Mean absolute error	0.1609	
Root mean squared error	0.3547	
Relative absolute error	35.9822 %	
Root relative squared error	74.7827 %	
Total Number of Instances	162	