PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Proposta de um Modelo para Comparação e Escolha de *Frameworks* para o Desenvolvimento de *Software* Baseado em Requisitos não Funcionais*

Proposal of a Model for Comparison and Choosing of Frameworks for Software Development Based on Non-Functional Requirements

José Mauro da Silva Sandy[1]
Flávio Luiz Schiavoni[2]

Resumo

A escolha dos *frameworks* utilizados ao longo do desenvolvimento de *software* causa um impacto direto na qualidade do produto desenvolvido, mas, muitas vezes, é realizada de forma subjetiva. Nesse contexto, este artigo visa apresentar um modelo para realizar a escolha de *frameworks* de maneira racional por meio de um modelo iterativo, baseado nos requisitos não funcionais dos *frameworks* que serão avaliados. O modelo define três etapas que podem ser repetidas, caso exista alguma dúvida após o término de cada iteração. Na primeira etapa, a configuração do questionário é realizada com a definição das questões que serão utilizadas durante a avaliação e seus respectivos pesos. A próxima etapa consiste na aplicação do questionário configurado na primeira etapa, para posterior apuração dos resultados. Na terceira e última etapa, as respostas dadas são quantificadas para determinar qual o *framework* é mais indicado para o contexto avaliado. Ao terminar, caso ainda não seja possível definir qual o mais adequado, uma nova iteração pode ser realizada. O modelo proposto tem por objetivo decidir de forma racional qual *framework* deve ser adotado. Por fim, apresentamos um estudo de caso desse modelo na escolha de um *framework* com bibliotecas gráficas para o projeto *Mosaicode*.

Palavras-chave: *Framework*. Modelo de Comparação. Requisitos Não Funcionais. Mosaicode.

*Submetido em 01/12/2017 - Aceito em 17/08/2018

[1]Mestrando em Ciência da Computação pela Universidade Federal de São João del-ReiUFSJ Programa de Pós-graduação em Ciência da Computação da UFSJ, Brasil– jmsandy@gmail.com

[2]Doutor em Ciência da Computação pela Universidade de São Paulo, professor do Programa de Pós-Graduação em Ciência da Computação e do Programa Interdepartamental de Pós-Graduação Interdisciplinar em Artes, Urbanidades e Sustentabilidade (PIPAUS) da Universidade Federal de São João del-Rei– UFSJ, Brasil– fls@ufsj.edu.br

Abstract

The choice of the frameworks that are used throughout software development causes a direct impact over the developed product but is often done subjectively. In this context, this article aims to present a model for making the frameworks choice in a rational way through an iterative model based on the nonfunctional requirements of the frameworks that will be evaluated. The model defines three steps that can be repeated if there is any doubt at the end of each iteration. During the first stage, the configuration of the questionnaire is carried out with the definition of the questions that will be used during the evaluation and their respective weights. The next step is to apply the questionnaire set up in the previous step for further calculation of the results. In the third and last step the given answers are quantified to determine which framework is most suitable for the evaluated context. At the end of the third step, if it is still not possible to define which is the most appropriate, a new iteration can be performed. The proposed model aims to rationally decide which artifact should be adopted. At the end, it is presented a study case of this model which chooses framework of graphic libraries for the Mosaicode project.

Keywords: Frameworks. Comparison Model. Non-Functional Requirements. *Mosaicode*.

1 INTRODUÇÃO

Para o desenvolvimento de aplicações, a utilização de *frameworks* é recomendada devido à abstração que o mesmo possibilita. Booch et al. (2006) definem *framework* como um padrão de arquitetura que fornece um *template* extensível para aplicações de um domínio. Uma das formas de classificar um *framework* é quanto à sua utilização em determinado contexto, podendo ser: 1) *frameworks* de aplicação ou horizontal, que independem do domínio e podem ser utilizados em várias aplicações; 2) *frameworks* de domínio ou vertical, atendem a um domínio específico e, normalmente, resolvem boa parte do domínio ao qual pertencem. Domínio é uma área de conhecimento ou de atividade, caracterizada por um conjunto de conceitos e terminologias compreendidas pelos seus participantes (BOOCH et al., 2006).

A diversidade de *frameworks* que realizam papéis similares pode dificultar a tomada de decisão sobre qual deve ser escolhido em certo contexto, haja vista que há diferentes abstrações e soluções para o domínio do problema ao qual se refere ou pretende solucionar. Com essa diversidade de *frameworks* que atendem ao mesmo propósito, surge a dificuldade de decidir qual *framework* é o melhor a ser adotado em determinado projeto de software. Essa escolha depende de uma tomada de decisão que irá influenciar todo o projeto, do desenvolvimento à manutenção, e pode implicar em riscos ou sucessos, além de influenciar a qualidade do projeto final e o foco a ser dado nas partes do projeto que não farão uso de *frameworks*. Por isto, ao realizar a escolha adequada de um *framework* para um projeto, as chances de sucesso do produto final podem aumentar de maneira considerável.

Segundo Pressman (2006), a qualidade do produto do *software* pode ser medida por meio de várias diretrizes adotadas pelos interessados no projeto. Tais diretrizes devem estabelecer critérios técnicos que serão avaliados para determinar a qualidade do projeto. Por poder afetar a qualidade do projeto final, a escolha de um *framework* deve ser realizada de forma racional. De acordo com Klein (2017), a capacidade de tomada de decisão nos seres humanos parte de duas perspectivas: a natural e a racional. Na primeira, os decisores estão, normalmente, envolvidos com problemas ou objetivos mal definidos, e as decisões são baseadas na experiência, pela intuição, simulações mentais, dentre outras. Já na decisão racional, existe um processo formal de tomada de decisão, ou linha de raciocínio a ser seguida onde, passo a passo, o decisor é levado a atingir o objetivo proposto pelo processo (PRESSMAN, 2006; KLEIN, 2017).

As escolhas dos *frameworks* utilizados ao longo do desenvolvimento de *software* são, muitas vezes, realizadas de forma subjetiva, em que a experiência prévia de algum membro da equipe direciona a escolha para um ou outro *framework*. Devido a isso, apresenta-se, neste artigo, um modelo decisório que possibilita a tomada de decisão de forma racional levando em conta os requisitos não funcionais dos *frameworks* comparados.

Os requisitos não funcionais são aqueles que não dizem respeito diretamente às funcionalidades fornecidas pelo sistema (PÁDUA, 2008). Eventualmente, podem dizer respeito ao sistema como um todo. Isso significa que, na maioria das vezes, eles são mais importantes que os requisitos funcionais individuais pois se uma falha em cumprir um requisito funcional puder

comprometer parte do sistema, pode torná-lo inútil (SOMMERVILLE, 2003). Requisitos não funcionais estão vinculados às características de qualidade do *software* e não às suas funcionalidades, sendo que, muitas vezes, são de difícil medição, o que torna complicado o processo de comparação com outro *software*, quando não se tem medidas claras e objetivas para fazê-lo. Segundo Pressman (2006), no sentido mais geral, a qualidade de *software* pode ser definida como uma gestão de qualidade efetiva aplicada, de modo a criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam (PRESSMAN, 2006).

Diversas áreas da computação apresentam metodologias para análise de viabilidade e controle de riscos com base em pontos que são medidos e avaliados com o objetivo de satisfazer uma solução da melhor forma possível. Podemos citar, como exemplos, o método *The Architecture Tradeoff Analysis Method (ATAM)* e o *Framework for Improving Critical Infrastructure Cybersecurity*, responsáveis pela análise arquitetural (KAZMAN et al., 1998) e análise e gerência de riscos de ciberataques (SEDGEWICK, 2014), respectivamente. Essas metodologias tentam auxiliar o desenvolvedor a analisar, avaliar, gerenciar e decidir quais processos devem ser realizados no cenário monitorado/avaliado.

No entanto, apesar de existirem metodologias específicas para auxiliar a análise e a gerência arquitetural e de ataques, como as supracitadas, há pouca ou nenhuma bibliografia para auxiliar a escolha racional de um *framework* a ser utilizado para atender um requisito não funcional específico, no contexto do desenvolvimento de sistemas de software. Baseado nos conceitos de medição e avaliação, este artigo apresenta uma metodologia para analisar e decidir qual o melhor *framework* a ser adotado para um contexto de *software* levando em consideração os requisitos não funcionais da aplicação.

Este artigo encontra-se dividido em quatro seções. Na Seção 2, são apresentados alguns dos conceitos relacionados a este trabalho. A Seção 3 apresenta a fundamentação teórica para a metodologia proposta, bem como o método proposto. A Seção 4 traz um estudo de caso aplicado sobre a metodologia por meio de um passo a passo, mostrando como fazer a seleção do melhor *framework* no contexto analisado. Já a Seção 5 traz os resultados alcançados pelo método e os trabalhos futuros que poderão ser realizados.

2 CONCEITOS RELACIONADOS

Uma boa solução de *software* procura alcançar um baixo grau de acoplamento entre seus componentes e alguns atributos de qualidade devem ser atingidos como forma de mensurar a qualidade desta solução. Segundo Pressman (2006), o acoplamento é uma medida qualitativa do grau com que os componentes estão ligados entre si e, conforme sua interdependência, o acoplamento pode aumentar ou diminuir. Um objetivo importante para o projeto é manter o acoplamento o mais baixo possível, pois, dessa maneira, torna-se possível a reutilização destes componentes em outros projetos, cujos domínios sejam semelhantes aos inicialmente projetados (PRESSMAN, 2006).

Analisar a qualidade de um *software* não é tarefa simples. Apesar de o nível de acoplamento entre seus componentes e o reuso serem bons indícios para essa análise, os mesmos não são os únicos fatores importantes para avaliar a qualidade do *software*. A qualidade de *software* depende da perspectiva de avaliação. Usuários finais podem medir a qualidade levando em consideração critérios de usabilidade. Desenvolvedores podem adotar critérios como facilidade de manutenção, dentre outras perspectivas. De maneira ampla, a qualidade pode ser caracterizada como sendo a criação de um *software* útil que agregue valor tanto ao fornecedor quanto ao usuário final.

Com o objetivo de classificar a qualidade de um *software*, Grady e Caswell (1987) propuseram analisar um conjunto de atributos mínimos ao qual foi atribuído o acrônimo *FURPS - functionality* (funcionalidade), *usability* (usabilidade), *reliability* (confiabilidade), *performance* (desempenho) e *suportability* (suportabilidade ou facilidade de suporte). A identificação desses atributos que permitem avaliar a qualidade do *software* ainda não nos possibilita avaliar o *software* como um todo, ou como avaliar a qualidade de um *software* levando em consideração todos esses atributos em conjunto. No entanto, há processos de decisão que, se combinados, aos atributos de qualidade, podem nos auxiliar a utilizá-los de forma mais abrangente, permitindo assim uma avaliação global do sistema (GRADY; CASWELL, 1987).

O processo de decisão em função dos atributos de qualidade será adotado, no modelo de comparação proposto, devido a sua utilização pelo método de análise dos prós e contras de uma arquitetura *ATAM*, que consiste em um conjunto de etapas que são realizadas de maneira iterativa, dentre as quais, se destacam (KAZMAN et al., 1998):

1. avaliar os atributos de qualidade, considerando cada atributo isoladamente;

2. identificar a sensibilidade dos atributos de qualidade, determinando quais atributos são afetados significativamente por uma variação da arquitetura.

A importância deste modelo iterativo é focar em cada atributo isoladamente. Com isso, pode-se avaliar, por exemplo, o baixo acoplamento dos componentes arquiteturais levando em consideração o atributo *facilidade de suporte*. Essas metodologias podem nos ajudar a avaliar os atributos individualmente, mas não nos ajudam ainda a decidir quais componentes utilizar para atender um projeto, principalmente no caso de haver mais de uma solução disponível que atenda aos requisitos estabelecidos. As escolhas entre possíveis soluções para um problema trazem para o projeto riscos que, se não avaliados durante a tomada de decisão, podem afetar diretamente a qualidade final do projeto.

O risco é definido por PMBOK (2013) como sendo um evento ou uma condição incerta que, se ocorrer, provocará um efeito positivo ou negativo, em um ou mais objetivos do projeto tais como: escopo, cronograma, custo e **qualidade**. A análise e a gestão de riscos é de suma importância para qualquer processo decisório e a metodologia proposta considera uma categoria específica de risco: *riscos técnicos*. Segundo Pressman (2006), *riscos técnicos* ameaçam a qualidade do *software* a ser produzido. Se um risco técnico se torna realidade, a implementação pode se tornar difícil ou impossível (PMBOK, 2013; PRESSMAN, 2006).

Com o intuito de diminuir os riscos inerentes a todo e qualquer processo de decisão, técnicas podem ser utilizadas para aumentar a assertividade, mitigar os riscos negativos e maximizar os riscos positivos implícitos em qualquer solução. Sete grupos de ferramentas que podem ser utilizados para resolver problemas e auxiliar na tomada de decisão são definidos por Warner (2000):

1. Serviço ao cliente;
2. Melhoria operacional da empresa;
3. Aperfeiçoamento de criatividade;
4. Solução de problemas;
5. Tomada de decisão;
6. Desenvolvimento organizacional;
7. Melhoria de qualidade.

Assim, notamos que tanto o aumento de qualidade quanto a diminuição de riscos são características importantes de um projeto e estão diretamente associadas a tomadas de decisão. Por isto, acreditamos que formalizar a tomada de decisão de maneira a fazê-la de forma racional, e não de maneira natural, pode auxiliar equipes de desenvolvimento a efetuar decisões de maneira mais adequada, o que possibilita maiores chances de sucesso no projeto final (WARNER, 2000).

Os atributos de qualidade *FURPS* representam uma meta para todo projeto de *software* (PRESSMAN, 2006) e estes serão a base do modelo de decisão ao longo deste artigo, assim como os grupos de ferramentas que podem auxiliar na tomada de decisão.

3 METODOLOGIA

Há diversas metodologias que podem ser utilizadas para auxiliar na tomada de decisão[3]. Conhecer tais metodologias auxiliam o processo decisório dos componentes de *software* que serão utilizados ao longo do desenvolvimento, aumentando a assertividade das escolhas efetuadas, pois podem definir de forma racional qual a melhor decisão a ser adotada para um cenário avaliado.

Os modelos apresentados por Warner (2000) podem nos auxiliar na tomada de decisão, mas nos levam para outro problema, que é decidir qual a metodologia mais adequada para se aplicar ao cenário que será avaliado. Selecionar uma técnica de decisão dentre as várias existentes pode vir a ser um objetivo difícil, e exigir um alto grau de conhecimento do problema a ser resolvido. Há modelos de decisão que não serão indicados para um determinado tipo

[3]Widman em seu livro, Problem-Solving & Decision-Making Toolbox, traz modelos para tomada de decisão.

de problema, cuja adoção poderá trazer riscos desnecessários ao item avaliado. Ao definir um método de decisão adequado, esse deve ser documentado para que esclarecimentos futuros possam ser realizados (WARNER, 2000).

No caso deste trabalho, propõe-se uma metodologia para auxiliar a tomada de decisão do projeto quando se decide por utilizar um *framework* para solucionar um determinado problema e há mais de uma solução que pode ser adotada. Um modelo iterativo é proposto com o intuito de mitigar os riscos na escolha de uma determinada solução, melhorar a qualidade do *software* final e diminuir os riscos do projeto. O modelo consiste em seis etapas que devem ser realizadas de forma iterativa para definir o melhor *framework* dentre os avaliados. As etapas consistem em:

1. Delecionar os *frameworks* candidatos;
2. Definir pesos para os requisitos não funcionais;
3. Montar o questionário de avaliação;
4. Aplicar o questionário de avaliação;
5. Apurar os resultados;
6. Avaliar os resultados.

As etapas do processo, apresentadas na Figura 1, podem ser agrupadas em três grupos que são responsáveis por: 1) configurar o modelo de avaliação; 2) aplicar o questionário a ser avaliado, e 3) apurar e avaliar os resultados obtidos na iteração. Nas subseções que seguem, cada grupo do modelo será apresentado com maiores detalhes.

Figura 1 – Apresentação das etapas de configuração, aplicação e avaliação dos resultados do modelo iterativo proposto ao longo deste artigo

Fonte: Elaborada pelos autores.

3.1 Configurar o Modelo de Avaliação

O primeiro grupo do modelo é responsável pelas três primeiras etapas do processo e tem como principal característica a configuração do modelo para avaliação.

Para uma correta configuração do modelo, reuniões em grupos com os interessados no projeto devem ser realizadas com o intuito de elucidar as dúvidas presentes, e levantar as mais diversas opiniões necessárias para configuração do mesmo. Nessas reuniões, devem ser discutidos: possíveis *frameworks*, questões para avaliação dos *frameworks* e os pesos a serem utilizados para cada requisito envolvido na escolha do *framework*. Técnicas facilitadoras como *brainstorm*, análise de decisão envolvendo critérios múltiplos, diagrama de afinidade, dentre outras, podem ser utilizadas. Warner (2000) apresenta diversas técnicas de decisão e quando essas podem ser empregadas para se solucionar um impasse. Essas técnicas são um rol exemplificativo e têm por objetivo apoiar a etapa de configuração do modelo de avaliação da metodologia aqui proposta, cabendo ao grupo responsável definir qual a melhor forma para a resolução dos impasses que, porventura, surgirem com base nas características de seus participantes (WARNER, 2000).

Nessa etapa, os possíveis *frameworks* a serem utilizados para resolução da tarefa desejada devem ser definidos. Nesse primeiro momento, não é necessário ter certeza de que a indicação feita é a mais adequada ao contexto, mas sim que, de algum modo, o resultado poderá ser obtido por ela. Para definir os possíveis candidatos a resolver o problema, reuniões realizadas com os membros da equipe do projeto podem ser realizadas e, por meio da técnica de *brainstorm*, as possibilidades podem ser apuradas. Além da utilização de técnicas de decisão, como a técnica *brainstorm* mencionada, uma base de conhecimento existente com projetos similares pode ser utilizada para definição dos *frameworks* a serem avaliados.

Uma vez definidos os *frameworks* a serem avaliados, é preciso mensurar a relevância de cada requisito não funcional definido pelo *FURPS* para o *framework* analisado. A importância dessa etapa é estabelecer os pesos que cada requisito indicado pelo *FURPS* tem na solução desejada, pois, por se tratar de requisitos não funcionais, o mesmo *framework* pode vir a ter impacto diferente em outros projetos, tornando sua relevância maior ou menor em função do contexto de utilização. Para auxiliar essa tarefa, sugere-se a utilização da Tabela 1.

Por fim, um questionário baseado na Escala de *Likert* deve ser definido, contendo as questões relevantes definidas nas reuniões em grupo realizadas, distribuindo-as entre os requisitos que serão analisados. Tais questões devem ser respondidas para cada *framework* que será avaliado. A Escala de *Likert* é definida por Brocke e Rosemann (2013) como sendo um conjunto de afirmações para as quais os participantes expressam suas opiniões, escolhendo um dos pontos da escala. Cada ponto possui uma numeração e a soma representa a pontuação das afirmações analisadas. Para esta metodologia, a escala será aplicada com os valores: 0 - Não Atende **(NA)**, 1 - Atende Parcialmente **(AP)** e 2 - Atende Completamente **(AC)**. Dessa maneira, cada resposta atribui um ponto à relação requisito/*framework* a fim de mensurá-la para determinar qual o melhor dentro do contexto analisado (BROCKE; ROSEMANN, 2013).

A Tabela 1 apresenta a estrutura padrão para configuração do modelo de avaliação para os *frameworks* (artefatos) avaliados.

Tabela 1 – Tabela para análise dos *frameworks* levando em consideração o conjunto de atributos FURPS

Requisitos	Pesos	Questões	Frameworks						
			Framework 1			...	Framework 2		
			NA	AP	AC	...	NA	AP	AC
Funcionalidade	(P)eso 1	Questão 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Questão N	○	○	○	...	○	○	○
Usabilidade	(P)eso 2	Questão 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Questão N	○	○	○	...	○	○	○
Confiabilidade	(P)eso 3	Questão 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Questão N	○	○	○	...	○	○	○
Desempenho	(P)eso 4	Questão 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Questão N	○	○	○	...	○	○	○
Suportabilidade	(P)eso 5	Questão 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Questão N	○	○	○	...	○	○	○
Resultados			(R)esultado 1			...	(R)esultado N		

Fonte: Elaborada pelos autores.

3.2 Aplicar Questionário

Uma vez que, na etapa de **Configuração do Modelo de Avaliação**, foram definidos os *frameworks*, os pesos dos requisitos não funcionais e as questões a serem aplicadas, pode-se passar para a etapa de **Aplicação do Questionário a ser Avaliado**. Nessa etapa, o questionário deve ser aplicado para avaliação e as diretrizes definidas serão aplicadas para condução do questionário. O questionário pode ser criado e disponibilizado da forma que for mais conveniente para a equipe, podendo ser desde plataformas *online* a impressos.

Segundo PMBOK (2013), os interessados podem ser pessoas, grupos ou organizações que podem ter impacto ou serem impactados por uma decisão, atividade ou resultado do projeto. Para comparação de *frameworks*, podem existir questões a serem destinadas aos interessados que não possuem envolvimento direto em todas as questões a serem aplicadas, ou que detenham envolvimento específico em alguns pontos avaliados. Por exemplo, equipes de arquitetura podem estar interessadas exclusivamente na plataforma em que se dará a execução do *framework*, ao passo que gerentes de projeto podem estar interessados no custo que a escolha de um *framework* trará ao produto final (PMBOK, 2013).

Como a quantidade de pessoas envolvidas na aplicação do questionário dependerá muito do tamanho da equipe participante do projeto, existirão questões que não serão destinadas a todos os interessados na escolha dos *frameworks* avaliados. Portanto, poderão existir questões destinadas a interessados específicos e estes podem responder apenas as que sejam de seu interesse e/ou conhecimento.

Algumas questões podem requerer que pequenas aplicações sejam desenvolvidas para avaliar as respostas. Esse requerimento fica a critério da equipe de desenvolvimento que deve definir quando algum protótipo será criado para validar os *frameworks* avaliados. Como regra, se faz necessário que o mesmo número de questões sejam respondidas para todos os *frameworks*, para que não ocorra imprecisão no modelo de apuração, ou seja, todas as questões respondidas para um determinado *framework* devem ser respondidas para os outros *frameworks* avaliados por um mesmo interessado. Caso o interessado não tenha envolvimento com as questões, as mesmas devem ser respondidas com o valor: 0 - Não Atende (NA).

3.3 Apurar e Avaliar os Resultados

As duas últimas etapas a serem realizadas consistem em apurar e avaliar os resultados dos questionários aplicados. Um questionário pode possuir várias questões (Q) atribuídas a cada requisito não funcional (*FURPS*) e esse, por sua vez, possui um peso (P) referente à sua relevância no contexto apurado. Dessa maneira, cada interessado (R) que receber o questionário deve responder todas as suas questões com base na escala anteriormente definida. Caso haja questões que não sejam da responsabilidade do interessado que estiver respondendo o questionário, essas devem ser marcadas na escala com o valor 0, para não interferir nos somatórios realizados. Sendo assim, a apuração por interessado se dá pela fórmula da Equação 1.

$$R_n = \frac{P_F \sum_{q=1}^{N} + P_U \sum_{q=1}^{N} + P_R \sum_{q=1}^{N} + P_P \sum_{q=1}^{N} + P_S \sum_{q=1}^{N}}{\sum_P} \quad (1)$$

Após realizar o somatório de cada interessado, os mesmos devem ser agrupados para definição da nota de cada *framework* (SF), conforme apresentado na Equação 2.

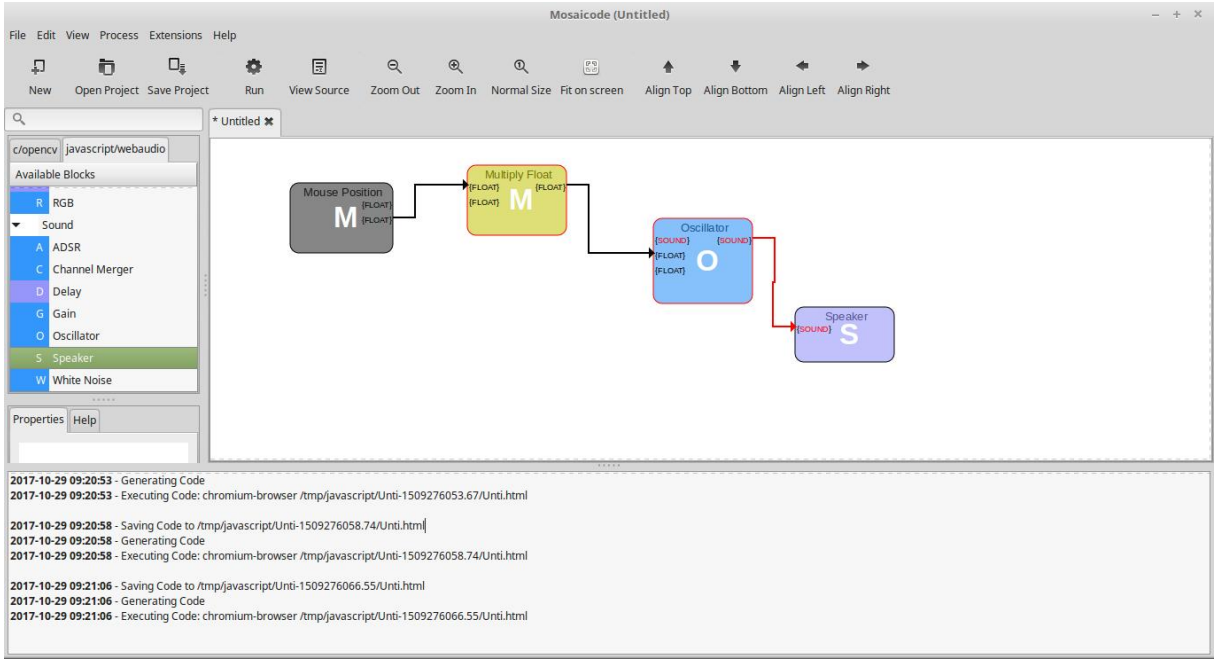
$$SF_n = \sum_{1}^{R} \quad (2)$$

A próxima e última etapa é a avaliação dos resultados obtidos pelo somatório das notas recebidas. Para definir o melhor *framework* após a avaliação, deve-se ordenar de forma decrescente os somatórios obtidos. Caso haja empate entre dois ou mais *frameworks*, a nota de maior peso deve ser considerada como critério de desempate. Em casos onde o empate persista, uma nova iteração deve ser realizada com novas questões e pesos a fim de definir o melhor *framework*. Além disso, ao término do questionário, caso exista alguma dúvida sobre a confiabilidade do resultado obtido pela avaliação, qualquer membro participante pode solicitar a aplicação de um novo questionário para elucidar as dúvidas da equipe.

4 ESTUDO DE CASO

Para validação do modelo proposto, um estudo de caso foi realizado para escolher uma biblioteca gráfica a ser adotada pelo projeto *Mosaicode*. Schiavoni e Goncalves (2017) definem o projeto *Mosaicode* como um ambiente de programação visual que utiliza o paradigma de encapsulamento de código em blocos onde cada bloco realiza uma tarefa específica. A visão geral do funcionamento do projeto *Mosaicode* é ilustrada na Figura 2, onde, ao executar os blocos, um aviso sonoro é emitido no navegador (SCHIAVONI; GONÇALVES, 2017).

Figura 2 – Ambiente de Programação Visual do *Mosaicode* contendo os blocos que são executados pelo sistema



Fonte: Elaborada pelos autores.

O projeto *Mosaicode* se caracteriza pela alta dependência de bibliotecas gráficas, haja vista que se trata de um ambiente de programação visual e que a maior parte de seu código se concentra em implementações de interfaces gráficas. Esse ambiente foi criado com base em um projeto denominado *Harpia* que não estava funcional devido à descontinuação de uma de suas bibliotecas para a persistência *XML*. O projeto *Harpia* utilizava a biblioteca gráfica **LibGlade** em sua interface e, ao iniciar o projeto *Mosaicode*, a equipe decidiu analisar qual era a melhor opção para desenvolver as interfaces gráficas para o projeto, ou seja, continuar com a utilização da **LibGlade** ou adotar uma nova biblioteca. A sugestão de alteração ocorreu devido ao fato de o projeto *Harpia* ser baseado em uma versão antiga da **LibGlade**, cujas ferramentas para manutenção das *GUI's* existentes não se encontram mais à disposição.

Para definir as questões pertinentes ao contexto do projeto, reuniões foram realizadas com os membros participantes do mesmo a fim de levantar as possíveis bibliotecas gráficas que, porventura, atenderiam a solução de maneira satisfatória. Tal levantamento foi feito com base na experiência dos membros da equipe em soluções anteriores. Ao longo das reuniões, embora

tenham surgido outras possibilidades, foram escolhidas duas bibliotecas: **LibGlade** e **GTK**. As mesmas foram escolhidas devido ao conhecimento da equipe na biblioteca **GTK+** e pelo projeto *Harpia* já fazer uso da biblioteca **LibGlade**. As bibliotecas foram avaliadas em sua versão na época em que a análise foi realizada. A biblioteca **LibGlade** estava na versão 2.6.4 e a biblioteca **GTK+** se encontrava na versão 3.2.

O próximo passo após a definição das bibliotecas a serem avaliadas foi a determinação dos pesos relacionados aos requisitos não funcionais que incidem sobre a parte do escopo do projeto que está sendo avaliado, e quais questões devem ser aplicadas para definir qual é a mais adequada de forma quantitativa. Como apresentado na Seção 3, os pesos possuem a função de determinar a relevância dos requisitos não funcionais e, assim, criar um resultado mais preciso para o cenário avaliado. Para o estudo de caso em questão, a equipe decidiu pela importância dos requisitos não funcionais, na seguinte ordem: confiabilidade, usabilidade, suportabilidade, funcionalidade e desempenho. Logo, seus pesos foram 5, 4, 3, 2 e 1, respectivamente. Portanto o requisito não funcional de confiabilidade foi considerado o mais importante pela equipe, recebendo o peso 5, o requisito de usabilidade recebendo o peso 4 e, assim, sucessivamente, até o requisito de desempenho, que recebeu o peso 1, por ser o menos relevante no cenário avaliado.

A Tabela 2 apresenta as questões e pesos mencionados que foram definidos após reuniões com os membros da equipe do projeto.

Tabela 2 – Questionário aplicado para comparação dos *frameworks LibGlade* e *Gtk+* no projeto *Mosaicode*

Requisitos	**Pesos**	**Questões**	**Frameworks**					
			LibGlade			**GTK+**		
			NA	**AP**	**AC**	**NA**	**AP**	**AC**
Func.	2	É madura para o problema existente?	○	○	○	○	○	○
		Atende as estratégias e demandas do projeto?	○	○	○	○	○	○
Usab.	4	Portável para várias plataformas (Linux, Mac, Windows)?	○	○	○	○	○	○
		Permite um fraco acoplamento ao projeto?	○	○	○	○	○	○
		Necessita dependências externas para funcionamento?	○	○	○	○	○	○
Conf.	5	O responsável pelo código-fonte é confiável e estável?	○	○	○	○	○	○
		O código-fonte está disponível e manutenção e auditoria é possível?	○	○	○	○	○	○
Desem.	1	Apresenta um desempenho satisfatório de resposta ao usuário?	○	○	○	○	○	○
Supor.	3	É documentada e exemplificada?	○	○	○	○	○	○
Resultados								

Fonte: Elaborada pelos autores.

Devido a uma experiência anterior com o projeto *Harpia*, em que uma biblioteca se tornou obsoleta e afetou todo o projeto, o grupo optou por considerar como mais importante a

confiabilidade e usabilidade do que o desempenho ou a funcionalidade. Além disto, quesitos de desempenho não importam para este projeto, pois o mesmo não tem preocupações de tempo real e visa apenas ser confiável e disponível para o usuário em diferentes plataformas. A suportabilidade possui um valor intermediário, pois está diretamente relacionada à documentação que a biblioteca fornece, o que pode facilitar o desenvolvimento da aplicação.

Após a definição do questionário a ser aplicado, o mesmo foi submetido a cada membro do projeto para avaliação. Os membros da equipe responderam às questões com base nos critérios definidos na Subseção 3.3, onde as opções: Não Atende, Atende Parcialmente e Atente Completamente, valiam 0, 1 e 2, respectivamente. A Tabela 3 apresenta a relação das respostas apresentadas pelos 6 membros da equipe que participaram do questionário. Ao observar as bibliotecas **LibGlade** e **GTK+**, é possível notar que há seis colunas abaixo de cada biblioteca e cada coluna refere-se as respostas de cada membro da equipe para cada biblioteca avaliada.

Tabela 3 – Respostas obtidas pelo questionário com seus respectivos pesos para a avaliação de *frameworks* de *GUI* no projeto *Mosaicode*

Questões	**Peso**	**LibGlade**						**GTK+**					
É madura para o problema existente?	2	1	1	2	2	1	2	2	2	2	1	1	2
Atende as estratégias e demandas do projeto?		0	1	0	2	1	1	2	2	2	1	1	2
Portável para várias plataformas (Linux, Mac, Windows)?	4	2	2	1	2	1	0	2	2	2	2	2	0
Permite um fraco acoplamento ao projeto?		2	2	2	1	1	0	2	2	1	2	1	0
Necessita dependências externas para funcionamento?		1	1	2	0	1	2	1	1	2	1	1	2
O responsável pelo código-fonte é confiável e estável?	5	1	1	0	1	0	1	2	2	2	1	1	2
O código-fonte está disponível e sua manutenção e auditoria é possível?		1	2	2	0	1	2	2	2	2	0	1	2
Apresenta um desempenho satisfatório de resposta ao usuário?	1	2	2	0	0	0	2	2	2	2	1	1	2
É documentada e exemplificada?	3	2	1	1	1	0	2	2	2	1	2	1	2

Fonte: Elaborada pelos autores.

Uma vez definidas as respostas dos membros da equipe, as mesmas foram sumarizadas com o objetivo de determinar qual biblioteca é melhor dentro do contexto avaliado para o projeto. As respostas foram somadas com base nos pesos definidos para os requisitos não funcionais analisados. Cada artefato avaliado contém um somatório total que será determinante para escolha da biblioteca a ser adotada. A Tabela 4 apresenta o somatório das respostas do questionário aplicado.

Tabela 4 – Resumo da avaliação das respostas com as equações aplicadas

Questões	Peso	LibGlade						GTK+					
É madura para o problema existente?	2	1	1	2	2	1	2	2	2	2	1	1	2
Atende as estratégias e demandas do projeto?		0	1	0	2	1	1	2	2	2	1	1	2
Portável para várias plataformas (Linux, Mac, Windows)?	4	2	2	1	2	1	0	2	2	2	2	2	0
Permite um fraco acoplamento ao projeto?		2	2	2	1	1	0	2	2	1	2	1	0
Necessita dependências externas para funcionamento?		1	1	2	0	1	2	1	1	2	1	1	2
O responsável pelo código-fonte é confiável e estável?	5	1	1	0	1	0	1	2	2	2	1	1	2
O código-fonte está disponível e sua manutenção e auditoria é possível?		1	2	2	0	1	2	2	2	2	0	1	2
Apresenta um desempenho satisfatório de resposta ao usuário?	1	2	2	0	0	0	2	2	2	2	1	1	2
É documentada e exemplificada?	3	2	1	1	1	0	2	2	2	1	2	1	2

Fonte: Elaborada pelos autores.

Ao analisar o resultado apresentado na Tabela 4, é possível determinar que ambas apresentam resultados bastante similares para algumas questões. A biblioteca GTK+ totalizou 279 pontos na pesquisa, enquanto a biblioteca **LibGlade** totalizou 207 pontos. Isso nos informa que, com base nos requisitos avaliados pela equipe, a biblioteca GTK+ deve ser adotada para utilização no projeto *Mosaicode*. A principal diferença entre os dois *frameworks* se dá quanto a confiabilidade do código fonte. Os membros da equipe julgaram que a biblioteca GTK+ tende a ter um futuro mais longevo. Os resultados obtidos foram calculados com base nas equações 1 e 2 apresentadas na Seção 3.

5 CONCLUSÃO E TRABALHOS FUTUROS

O processo de definição de qualquer artefato de *software* é de suma importância para o bom andamento de um projeto durante todo o seu ciclo de vida. O processo de escolha, na maioria das vezes, ocasiona bastante dificuldade pois não é possível analisar de forma objetiva os requisitos importantes para o projeto. Devido a isso, a realização de passos exequíveis para a escolha adequada deve ser adotada já nos primeiros momentos do projeto.

O estudo de caso apresentado, com ênfase no projeto *Mosaicode*, mostrou que o modelo proposto para escolha de artefatos é uma maneira eficiente e racional para se determinar qual é o melhor artefato dentro do contexto avaliado pela equipe. Além disso, o modelo criou um ambiente com sinergia entre os membros da equipe, pois todos puderam participar de forma direta durante o processo de escolha do *framework* a ser adotado pelo projeto e fez com que equipe pensasse em alternativas que poderiam ser importantes para o projeto. Além do mais,

propiciou um ambiente de troca de experiências e aprendizado entre os membros da equipe tornando-os mais capacitados para futuras escolhas.

O modelo aqui apresentado pode ser aplicado em diferentes contextos e equipes, mas para que o mesmo se torne eficiente é necessário que os membros da equipe tenham um conhecimento sobre técnicas de decisão, ainda que minimamente, ou que haja entre os membros da equipe um mediador que possa facilitar e definir o fluxo das decisões tomadas durante as reuniões. Devido a isto, a principal dificuldade em aplicar o modelo proposto é o conhecimento prévio da equipe participante em técnicas de decisão para guiar o processo de formulação do questionário. À medida que o modelo for sendo aplicado, uma base de conhecimento pode ser formada para decisões futuras e questões podem ser agrupadas para um determinado contexto, o que pode tornar o modelo mais ágil para ser aplicado.

De maneira geral, o modelo se mostrou eficaz para o contexto analisado e, para que o mesmo se torne maduro, é necessário que seja executado em outros contextos e por equipes distintas. Com isso, o modelo não pode ser considerado definitivo e poderá sofrer evoluções quando for mais utilizado por outras equipes e projetos.

REFERÊNCIAS

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário**. [S.l.]: Elsevier Brasil, 2006.

BROCKE, Jan Vom; ROSEMANN, Michael. Metodologia de pesquisa. **Porto Alegre: AMGH Editora**, 2013.

GRADY, Robert B; CASWELL, Deborah L. Software metrics: establishing a company-wide program. Prentice Hall, 1987.

KAZMAN, Rick et al. The architecture tradeoff analysis method. In: IEEE. **Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on**. [S.l.], 1998. p. 68–78.

KLEIN, Gary A. **Sources of power: How people make decisions**. [S.l.]: MIT press, 2017.

PÁDUA, Wilson Paula Filho de. **Engenharia de software**: Fundamentos, métodos e padrões. [S.l.]: LTC, 2008.

PMBOK, GUIA. Um guia do conhecimento em gerenciamento de projetos. **Quarta Edição**, v. 123, 2013.

PRESSMAN, Roger S. **Engenharia de Software. Tradução: Rosângela Delloso Penteado**. [S.l.]: São Paulo: McGraw-Hill, 2006.

SCHIAVONI, Flávio Luiz; GONÇALVES, Luan Luiz. Programação musical para a web com o mosaicode. In: **XXVII Congresso da Anppom-Campinas/SP**. [S.l.: s.n.], 2017.

SEDGEWICK, Adam. Framework for improving critical infrastructure cybersecurity, version 1.0. **NIST**, 2014.

SOMMERVILLE, Ian et al. **Sources of power: How people make decisions**. [S.l.]: Addison Wesley São Paulo, 2003. v. 6.

WARNER, Jon. **Problem-Solving & Decision-Making Toolbox**. [S.l.]: Human Resource Development, 2000.