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

Metodologia para Comparação de Frameworks*

Methodology for Comparing Frameworks

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

Resumo

Este artigo apresenta uma metodologia para realizar a comparação entre *frameworks* através de um modelo iterativo baseado em requisitos não funcionais dos *frameworks* que serão avaliados. O modelo define três etapas que podem vir a 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 perguntas que serão utilizadas durante a avaliação e seus respectivos pesos. A próxima etapa consiste na aplicação do questionário configurado na etapa anterior 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 término da terceira etapa, 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. Para validação do modelo, um estudo de caso realizado sob o projeto *Mosaicode* com o intuito de escolher qual o *framework* com bibliotecas gráficas é o mais indicado para utilização no projeto.

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

---

*Artigo apresentado à Revista Abakos

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

[2]Doutor em Ciência da Computação pela Universidade Federal de São Paulo - USP , E-mail:fls@ufsj.edu.br Programa de Pós-graduação em Ciência da Computação da UFSJ, Brasil .

Abstract

This article presents a methodology to perform the comparison between frameworks through an iterative model based on non-functional requirements of the frameworks that will be evaluated. The model is defined in three steps that can be repeated if there is any doubt after the end of each iteration. In 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. For the validation of the model, a study case was carried out under the Mosaicode project with the intention of choosing which framework of graphic libraries is the most suitable to use in the project.

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

1 INTRODUÇÃO

Para o desenvolvimento de aplicações a utilização de *frameworks* é recomendada devido a abstração que o mesmo possibilita. (BOOCH et al., 2006) define *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 a sua utilização em um 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 pertence. 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 determinado contexto, haja vista que há diferentes abstrações e soluções para o domínio do problema ao qual o mesmo se refere ou pretende solucionar. Com esta 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*. Esta 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 na qualidade do projeto final e no 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 através 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 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. Consideramos que a tomada de decisão quanto a escolha de um *framework* deve ser uma decisão racional e apresentamos neste artigo um modelo decisório que possibilita a tomada de decisão 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 et al., 2003). Requisitos não funcionais estão vinculados às características de qualidade do *software* e não às suas

funcionalidades, sendo que em 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.

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 *ATAM - The Architecture Tradeoff Analysis Method* e o *framework Cybersecurity - 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. Estas 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 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 através de um passo a passo mostrando como fazer a seleção do melhor *framework* no contexto analisado. Já a Seção 5 trará os resultados alcançados pelo método e 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 desta forma torna-se possível a reutilização destes componentes em outros projetos que possuam domínios semelhantes aos quais foram inicialmente projetados.

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 esta 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; CASWELL, 1987) propôs analisar um conjunto de atributos mínimos ao qual foi atribuído o acrônimo FURPS - *functionality* (funcionalidade), *usability* (usabilidade), *reliability* (confiabilidade), *performance* (desempenho) e *suportability* (facilidade de suporte). A identificação destes 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 estes atributos em conjunto. No entanto, há processos de decisão que combinados aos atributos de qualidade, podem nos auxiliar a utilizar estes de forma mais abrangente, permitindo assim uma avaliação global do sistema.

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 - The Architecture Tradeoff Analysis Method)* que consiste em conjunto de etapas que são realizadas de maneira iterativa, dentre as quais, se destaca (KAZMAN et al., 1998):

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

2. identificar a sensibilidade dos atributos de qualidade.

A importância deste modelo iterativo é focar em cada atributo isoladamente. Com isto, pode-se avaliar, por exemplo, o baixo acoplamento dos componentes arquiteturais levando em consideração o atributo *facilidade de suporte*. Estas 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 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 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.

Com o intuito de diminuir os riscos inerentes a todo e qualquer processo de decisão, técnicas podem ser utilizadas para aumentar a assertividade e assim 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 que 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.

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 no 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.

No entanto, apesar de estas metodologias auxiliarem na tomada de decisão, elas 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 às 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 de problema e, sua adoção, poderá trazer riscos desnecessários ao item avaliado. Ao definir um método de decisão adequado, este deve ser documentado para que esclarecimentos futuros possam ser realizados.

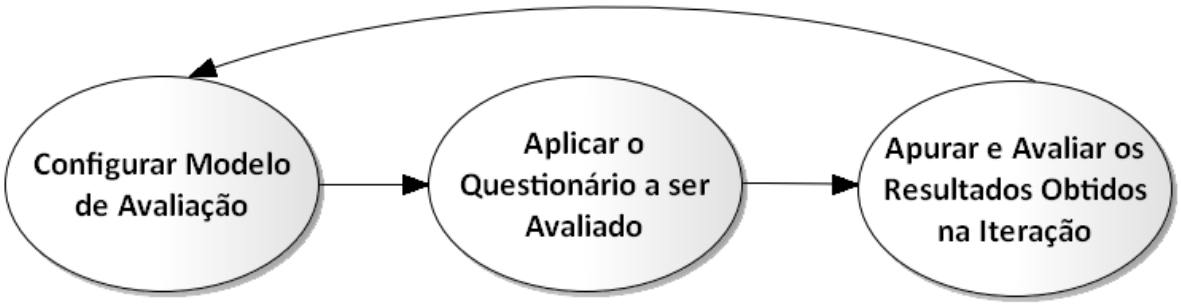
[3]Widman em seu livro, Problem-Solving & Decision-Making Toolbox, traz modelos para tomada de decisão.

No caso deste trabalho, estamos propondo 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. selecionar 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.



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. Técnicas facilitadoras como *brainstorm*, análise de decisão envolvendo critérios múltiplos, diagrama de afinidade, dentre outras, podem ser utilizadas. Para maiores informações sobre estas técnicas, consulte (WARNER, 2000), que apresenta de forma clara diversas técnicas de decisão e quando estas podem ser empregadas para se solucionar um impasse.

Nesta etapa, os possíveis *frameworks* a serem utilizados para resolução da tarefa desejada devem ser definidos. Neste 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 através 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 nesta tarefa, sugerimos a utilização da Tabela 1.

Por fim, um questionário baseado na Escala de *Likert* deve ser definido e suas questões distribuídas entre os requisitos. Tais questões devem ser respondidas para cada *framework* que será avaliado. A Escala de *Likert* é definida por (BROCKE; 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 onde 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 forma, cada resposta atribui um ponto a relação requisito/*framework* afim de mensurá-la para determinar qual o melhor dentro do contexto analisado.

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**	**Perguntas**	**Frameworks**						
			Framework 1			...	**Framework N**		
			NA	**AP**	**AC**	**...**	**NA**	**AP**	**AC**
Funcionalidade	(P)eso 1	Pergunta 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Pergunta N	○	○	○	...	○	○	○
Usabilidade	(P)eso 2	Pergunta 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Pergunta N	○	○	○	...	○	○	○
Confiabilidade	(P)eso 3	Pergunta 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Pergunta N	○	○	○	...	○	○	○
Desempenho	(P)eso 4	Pergunta 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Pergunta N	○	○	○	...	○	○	○
Suportabilidade	(P)eso 5	Pergunta 1	○	○	○	...	○	○	○
		...	...	...	...	...	...	...	...
		Pergunta N	○	○	○	...	○	○	○
Resultados			(R)esultado 1			...	(R)esultado N		

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, podemos passar para a etapa de **Aplicação do Questionário a ser Avaliado**. Nesta etapa, o questionário deve ser aplicado aos interessados no projeto para avaliação e as diretrizes definidas, em cada projeto, 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 ou até mesmo em arquivos impressos. Não é necessário que várias pessoas participem do questionário, até mesmo quando houver um único desenvolvedor e este estiver em dúvida entre qual *framework* utilizar, o modelo pode ser adotado para diminuir as incertezas e indicar o melhor caminho.

Segundo (PMBOK, 2013) uma parte interessada pode ser pessoas, grupos ou organizações que podem ter impacto ou serem impactados por uma decisão, atividade ou resultado do projeto. Mesmo para comparação de *frameworks* podem existir perguntas a serem destinadas a interessados não técnicos e estas podem ser avaliadas isoladamente no modelo ao apurar os resultados.

A quantidade de pessoas envolvidas na aplicação do questionário dependerá muito do tamanho da equipe envolvida no projeto. Nem todos os membros da equipe podem estar aptos a avaliar todos os requisitos não funcionais ou até mesmo nenhum. Devido a isso as perguntas podem ser direcionadas a membros que possuam os conhecimentos necessários para avaliar os *frameworks* selecionados.

Algumas questões podem requerer que pequenas aplicações sejam desenvolvidas para avaliar as respostas, este 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 os mesmos números 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* deve ser respondido para os outros *frameworks* avaliados.

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 este por sua vez possui um peso (P) referente a sua relevância no contexto apurado. Desta forma cada interessado (R) que receber o questionário deve responder todas as suas questões com base na escala anteriormente definida. Caso haja perguntas que não sejam da responsabilidade do interessado que estiver respondendo o questionário, estas devem ser marcadas na escala com o valor: 0 - (NA) Não Atende, para não interferir nos somatórios realizados. Sendo assim, a apuração por interessado se dá pela fórmula da Equação 1.

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

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.

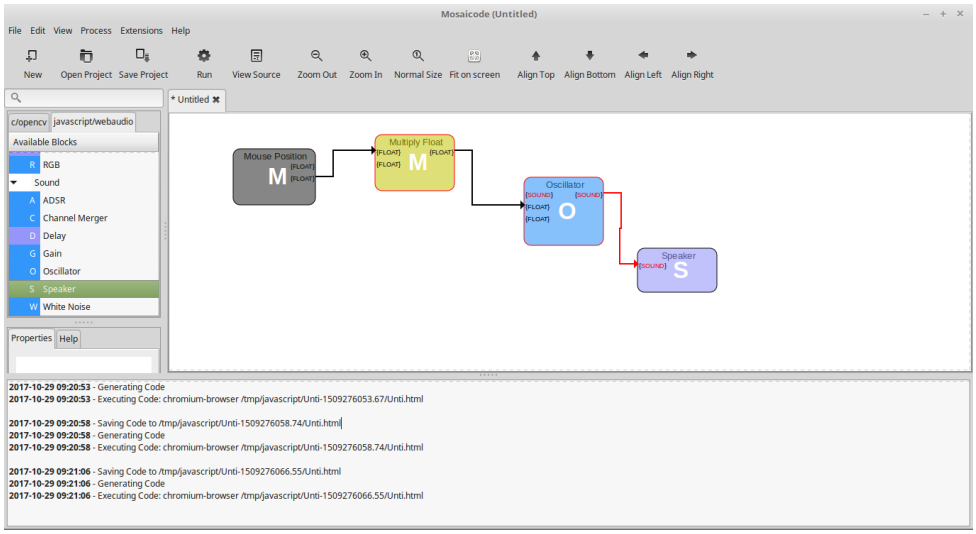
$$SF_n = \sum_{1}^{R} \quad (2)$$

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. Caso o empate persista uma nova iteração deve ser realizada com novas perguntas e pesos.

4 ESTUDO DE CASO

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

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



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. Este ambiente foi criado com base em um projeto denominado *Harpia* que não estava funcional devido a 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 a disposição.

Para definir as questões pertinentes ao contexto do projeto, reuniões foram realizadas com os membros participantes do mesmo, afim 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 que fizeram, ou faz, uso de bibliotecas gráficas e que fossem adequadas ao projeto. 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 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 perguntas 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 maior peso. Os demais pesos foram distribuídos de mais decrescente com base na ordem de relevância dos requisitos para a equipe.

A tabela 2 apresenta as perguntas 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.

Requisito	**Peso**	**Perguntas**	**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								

A experiência anterior, projeto *Harpia*, de ter uma ferramenta obsoleta devido a uma

dependência da mesma ter se tornado obsoleta fez com que o grupo optasse por considerar mais importante a confiabilidade e usabilidade do que desempenho ou funcionalidade. Além disto, quesitos de desempenho não importam para este projeto, pois o mesmo não tem preocupações de tempo real e visa apenas em ser confiável e disponível para o usuário em diferentes plataformas. A suportabilidade possui um valor intermediário, pois está diretamente relacionado 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 perguntas com base nos critérios definidos na subseção 3.3, onde as opções: Não Atende, Atende Parcialmente e Atente Completamente, valem 0, 1 e 2, respectivamente. A tabela 3 apresenta a relação das respostas efetuadas 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 onde 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*.

Perguntas	**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

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.

Perguntas	LibGlade	GTK+
É madura para o problema existente?	18	20
Atende as estratégias e demandas do projeto?	10	20
Portável para várias plataformas (Linux, Mac, Windows)?	32	40
Permite um fraco acoplamento ao projeto?	32	32
Necessita dependências externas para funcionamento?	28	32
O responsável pelo código-fonte é confiável e estável?	20	50
O código-fonte está disponível e sua manutenção e auditoria é possível?	40	45
Apresenta um desempenho satisfatório de resposta ao usuário?	6	10
É documentada e exemplificada?	21	30
Resultado:	207	279

Ao analisar o resultado apresentado na tabela 4 é possível determinar que ambas apresentam resultados bastante similares para algumas questões. 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. Como pode ser observado, a biblioteca **GTK+** totalizou 279 pontos na pesquisa, enquanto a biblioteca **LibGlade** totalizou 207 pontos, isto nos informa que com base nos requisitos avaliados pela equipe, a biblioteca **GTK+** deve ser adotada para utilização no projeto *Mosaicode*. 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 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 em todo 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.

Para que o modelo 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. À medida que o modelo for sendo aplicado uma base de conhecimento pode ser formada para decisões futuras e perguntas 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, J.; ROSEMANN, M. **Metodologia de Pesquisa**. [S.l.: s.n.], 2013. ISBN 9788565848367.

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, W de. **Engenharia de Software: Fundamentos, Métodos e Padrões**. [S.l.]: LTC,, 2008.

PMBOK, GUIDE. Um guia do conhecimento em gerenciamento de projetos. **Quarta Edição**, 2013.

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

SCHIAVONI, Flávio Luiz; GONCALVES, Luan Luiz. Programação musical para a web com o mosaicode. **XXVII Congresso da Associação Nacional de Pesquisa e Pós-Graduação em Música – Campinas - 2017**, 2017.

SEDGEWICK, A. Framework for improving critical infrastructure cyber-security. **NIST**, 2014.

SOMMERVILLE, Ian et al. **Engenharia de software**. [S.l.]: Addison Wesley São Paulo, 2003. v. 6.

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