Uma IDE para geradores de Código de Granularidade Fina

José Vinnicius dos Santos Oliveira; Rodrigo Vilar (Orientador)[1]

[1] Departamento de Ciências Exatas - Centro de Ciências Aplicadas e Educação - Universidade Federal da Paraíba - Rio Tinto - Paraíba - Brasil

{vinnicius.santos, rodrigovilar}@dcx.ufpb.br

Abstract. *This article describes the process of producing an interface for an Integrated Software Development Environment. The interface was developed to fix usability issues that directly affect the process of using a code generator with fine-grained templates. The main contribution of this work is to make it easier to understand the structure of templates and their interaction, so that developers become more productive and correct errors faster, resulting in an improvement in productivity.*

Resumo. *Este artigo descreve o processo de produção de uma interface para um Ambiente Integrado de Desenvolvimento de Software. A interface foi desenvolvida para corrigir problemas de usabilidade que afetam diretamente o processo de utilização de um gerador de código com templates de granularidade fina. A principal contribuição deste trabalho é facilitar a compreensão da estrutura dos templates e sua interação, de modo que os desenvolvedores se tornem mais produtivos e corrijam erros mais rapidamente, resultando em uma melhora na produtividade.*

1. *Introdução*

A produtividade tem cada vez mais destaque no campo do desenvolvimento de aplicações corporativas. A utilização de IDEs (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) com autocompletar e até a utilização de padrões de projeto, são exemplos que ajudam a reduzir o tempo de desenvolvimento das aplicações. Acelerar o processo de desenvolvimento tornou-se um dos tópicos mais importantes para pesquisadores da área [Wagner 2018], entretanto, as soluções em busca da produtividade nem sempre produzem apenas bons resultados. Por exemplo, ao utilizar-se padrões de projeto, como o *Template Method*, aumenta-se o reuso de código, diminuindo assim o tempo necessário para escrever código e seu tamanho final [Gamma *et al.* 1995]. Mas, nem sempre, escrever menos código ou demandar menos tempo de codificação, significa ser mais produtivo, pois o código escrito em diversas classes pode ser difícil de ser entendido, atrapalhando a visualização do *software* e consequentemente sua manutenção [Hegedűs 2012]. Uma alternativa para melhora na produtividade é a geração de código baseada em *templates*, que surge propondo a automatização do processo de desenvolvimento com a construção de modelos reutilizáveis de código. Em

Trabalho de conclusão de curso, sob orientação do professor Rodrigo de Almeida Vilar de Miranda submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO.

termos abstratos, produtividade é uma medida de eficiência na conversão de recursos em bens econômicos, isto é, a relação entre o que é produzido (bens e/ou serviços) e recursos que são usados para produzi-los [Wainer 2003]. Na engenharia de software, tempo é uma métrica fundamental para a produtividade, porém, deve-se levar em consideração também a qualidade do que foi produzido, visto que a correção dos problemas causados pela velocidade de codificação demandam muito esforço e custo [Case 1985]. A utilização dos *templates* economiza tempo na construção das soluções, inicialmente melhorando a produtividade, porém, afeta de forma significativa o processo de visualização do *software*, uma vez que utiliza componentes baseados em metadados.

As técnicas de geração de código provêm uma significativa redução no tempo de realização de tarefas básicas, mas na maioria das vezes ao custo da redução na compreensão do código. Além disso, ao quebrar o código em *templates* (principalmente os de granularidade fina [Vilar *et al*. 2015]), se fomenta a alta reutilização de componentes, porém eles tornam-se complexos demais para pessoas externas ao seu desenvolvimento e possuem uma difícil visualização, dificultando a depuração em casos de erros e *bugs*. Durante a realização deste trabalho, observou-se que a dificuldade de visualizar o código gera um problema quando novos integrantes do time de desenvolvimento precisam entender o que está sendo feito, e até mesmo o criador dos *templates* pode ter dificuldades de entender o fluxo que a geração está tomando em caso de pausas no processo de desenvolvimento.

A visualização do *software* por si só é uma área que tem ganhado importância na engenharia de software e pode ser definida como a utilização de artefatos gráficos a fim de melhorar a compreensão do código de um software e diminuir a sua complexidade [Gračanin 2005]. Ao utilizar técnicas de visualização do *software* a exigência de altas habilidades cognitivas, como o raciocínio lógico, faz-se menos necessária. Considerando-se a redução na quantidade de informações necessárias para a realização de tarefas [Koschke 2003] amplia-se o acesso à produção de *software*.

Para a execução deste trabalho, selecionou-se, devido à ausência de outras ferramentas na classe específica de geradores de código de granularidade fina, o *Potter-SaaS*, como objeto de estudo. Após uma análise do processo de utilização e de coleta de *feedback* por parte dos utilizadores, buscou-se identificar os principais problemas de usabilidade da ferramenta e seus impactos na produtividade.

Na engenharia de *software*, a usabilidade é definida como um conjunto de atributos que influenciam o esforço necessário para o uso de uma interface e a avaliação individual de tal uso por um conjunto declarado ou implícito de usuários. [ISO/IEC 9126 apud Bevan 2001].

Este trabalho apresenta uma proposta de interface alternativa para este gerador, que, devido ao curto tempo de execução, visa corrigir alguns dos problemas encontrados, analisa os impactos dessas correções no processo de utilização da ferramenta e verifica se a interface final adequa-se aos conceitos de boa usabilidade.

2. Descrição do processo

Para a execução deste trabalho, o processo foi dividido em três partes principais: Pesquisa, Desenvolvimento e Avaliação de Resultados. A pesquisa compreende-se como todos os passos do processo que buscaram identificar os problemas da ferramenta. Já a execução é a realização dos estudos e desenvolvimento das propostas. E por fim, a avaliação consiste em avaliar a proposta formulada após seu desenvolvimento. Ao longo desta Seção, todos estes passos serão descritos detalhadamente.

2.1. Pesquisa

A parte inicial do processo foi compreender a situação atual do problema a ser resolvido. Uma busca para conhecimento do estado da arte foi conduzida, utilizando-se das seguintes palavras chaves: *i) Generative programming; ii) Metaprogramming; iii) Automatic programming; iv) Software Visualization; v) Program Comprehension;* foi realizada para compreender os princípios fundamentais da geração de código. Uma vez finalizada, a pesquisa ajudou a produzir embasamento teórico para a produção deste artigo.

Após a realização do estudo, iniciou-se a primeira análise da ferramenta alvo, que será descrita na Seção três deste artigo, o *Potter-SaaS*. Nesta etapa foi necessário compreender o fluxo de utilização da ferramenta. Tal compreensão foi adquirida com a utilização ativa da ferramenta durante alguns meses, gerando-se artefatos testes, para aprendizado e contribuição com o ecossistema da ferramenta, e dialogando-se, informalmente, por meio de reuniões esporádicas, com a comunidade de usuários da ferramenta. A comunicação com os usuários deu-se para o compartilhamento de experiências e conhecimentos. Durante esta etapa, identificou-se problemas de usabilidade que impactaram de forma direta, aumentando a complexidade no uso da ferramenta e consequentemente o tempo necessário para realização de atividades na produção dos artefatos. Um exemplo é apresentado na Figura 1, onde a tela de listagem dos *workers* não possui opções de filtro e busca. Uma vez que o gerador quebra os componentes em grânulos finos, o número de *workers* aumenta consideravelmente (para algumas centenas), causando extrema dificuldade de visualização das partes de um projeto.

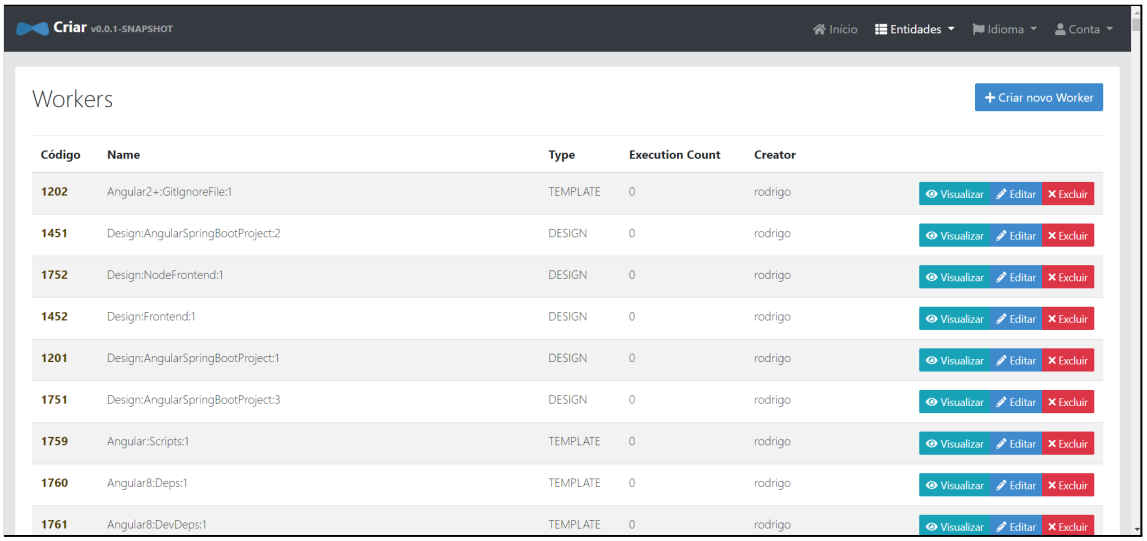

Figura 1. Tela de listagem de *workers*

Outro exemplo de problema encontrado, e que mostrou ser extremamente prejudicial a experiência de usuário, foi a ausência de mensagens de erro que explicassem as razões dos problemas, como podemos ver na imagem abaixo. Esta ausência de detalhes exige uma análise detalhada no código para compreender as causas, porém, como o erro é genérico o utilizador não consegue ter um ponto de partida claro para a realização da análise aumentando assim o tempo necessário para a execução das tarefas.

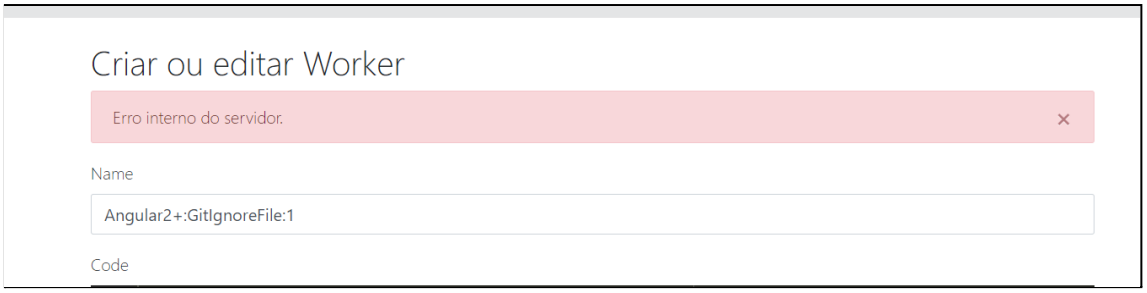

Figura 2. Erro na criação de *workers*

Seguindo a percepção inicial dos problemas, buscou-se validação por meio de coletas "rápidas e sujas", realizadas com outros utilizadores. Nessas conversas conseguiu-se compreender se os problemas coletados na análise inicial prejudicava a utilização dessas pessoas, quais os pontos positivos e os negativos da ferramenta, e sugestões para o desenvolvimento de uma nova interface.

2.2. Desenvolvimento

A parte inicial do processo de desenvolvimento consistiu em correlacionar os problemas de usabilidade com as heurísticas de Nielsen [Nielsen 1994]. As dez heurísticas, são princípios utilizados para avaliar a aplicação de usabilidade em uma ferramenta, sendo estas: *i) Visibilidade do Status do Sistema; ii) Compatibilidade entre o sistema e o mundo real; iii) Controle e liberdade para o usuário; iv) Consistência e Padronização; v) Prevenção de Erros; vi) Reconhecimento em vez de memorização; vii) Eficiência e flexibilidade de uso; viii) Estética e design minimalista; ix) Recuperação de Erros; x) Ajuda e documentação*. Uma avaliação empírica, baseada na vivência do pesquisador, foi realizada. Nessa avaliação, realizou-se uma nova análise da ferramenta, com o objetivo de confirmar a existência dos problemas coletados. Uma vez coletados, os problemas foram analisados individualmente, relacionando-se com as heurísticas, e avaliando também a severidade do impacto causado na utilização da ferramenta[1]. Foram encontrados dezenove problemas, sendo doze de alta, quatro de média e três de baixa severidade.

Com os problemas de usabilidade correlacionados e devidamente descritos, iniciou-se a produção da nova *interface* do sistema. Para acelerar o processo de desenvolvimento, foram utilizados um conjunto de ferramentas e *frameworks*, doravante denominado como *VIRTUS-Core*. A nova *interface* buscou replicar a base do fluxo de uso da ferramenta, a fim de reduzir a necessidade de uma brusca readaptação por parte dos utilizadores.

O processo prático iniciou-se com a produção de protótipos de alta fidelidade[2], utilizando o *Adobe XD*, e que foram sendo refinados ao longo do tempo. Uma vez que os protótipos foram validados com a comunidade, que fora consultada anteriormente, por meio de reuniões onde os mesmos foram apresentados e discutidos, iniciou-se o processo de desenvolvimento da *interface,* que está descrita na Seção 4 deste artigo.

2.3. Avaliação

O processo avaliativo teve como objetivo analisar se a solução proposta atende as heurísticas mencionadas. Para tal, uma nova avaliação empírica foi realizada. Esta avaliação está descrita na Seção 5 deste trabalho.

---

[1] Disponível em Problemas de usabilidade - Potter-SaaS

[2] Disponiveis em https://xd.adobe.com/view/a20dec39-f85d-467f-a33b-61c78b478c54-f75e/

3. Descrição das tecnologias

Para realização deste trabalho foram utilizadas algumas ferramentas, que serão descritas ao longo desta seção.

3.1. Potter-SaaS

O Potter-SaaS é uma ferramenta proprietária de geração de código de granularidade fina, que permite aos utilizadores a criação de componentes reutilizáveis que são agrupados em projetos e que geram artefatos usáveis. Os componentes são fragmentados em unidades extremamente pequenas, a fim de propiciar alto reuso, dando origem ao termo granularidade fina. Possui estruturas próprias, como *workers, designs* e *shapes*. Os componentes de um projeto são chamados de *workers*, estes são separados em:

- *Templates*, que são em suma, arquivos textuais que utilizam como modelo os códigos-fonte de projetos finalizados e bem sucedidos [Magno 2015];
- *Designs*, que são as regras de conexão e lógica dos *templates*; e
- *Shapes*, que manipulam os metadados do projeto em tempo de execução a fim de simplificar o código dos *templates*.

Os *workers,* quando isolados, não são auto-suficientes, visto que precisam de metadados para sua execução. Estes metadados são incluídos nos *projects*, que carregam as informações necessárias para a geração dos artefatos finais. O *Potter-SaaS*, diferentemente de geradores com outros tipos de granularidade, permite aos utilizadores uma evolução individual dos *templates* sem a necessidade de readaptação de todo o sistema, visto que cada grânulo é produzido para ser o mais desacoplado possível.

3.2. VIRTUS-Core

O *VIRTUS-Core* é um conglomerado de ferramentas que funcionam como *scaffolding* de projetos, sendo produzido e mantido pelo Núcleo de Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação, Comunicação e Automação da Universidade Federal de Campina Grande, o VIRTUS[3]. Oferece ao desenvolvedor um servidor e uma *interface*, conectados entre si, fornecendo uma série de funções pré-configuradas, tais como autenticação, conexão com banco de dados, telas de listagem, funções de buscas e serviços de traduções. Sua utilização traz agilidade ao processo de desenvolvimento, visto que não existe uma necessidade de realizar as configurações iniciais do projeto. O *VIRTUS-Core* utiliza as linguagens de programação *Java*, como servidor, e *Javascript* como *interface*.

3.2. Adobe XD

O *Adobe XD*[4] é um software proprietário para o design de protótipos navegáveis, que permitem uma simulação das interações dos usuários, contribuindo para uma melhor compreensão da usabilidade das soluções. Contém ferramentas para a criação de protótipos de média e alta fidelidade. A ferramenta foi utilizada neste trabalho para a produção dos protótipos iniciais.

---

[3] https://www.virtus.ufcg.edu.br/

[4] https://www.adobe.com/br/products/xd.html

4. Solução

A solução de interface proposta foi desenvolvida utilizando algumas das tecnologias descritas na Seção 3. A estrutura base da *interface* fornecida pelo *VIRTUS-Core*, foi mantida, pois esta já fornecia estética agradável e harmônica. A ênfase da refatoração foi a utilização de componentes extras, para que esta fornecesse melhora na usabilidade.

A primeira tela refatorada foi a de listagem de *workers*. Na *interface* anterior (Figura 1), o usuário precisava percorrer toda a lista para buscar itens específicos, uma vez que esta, por não possuir paginação e sofrer com a particularidade da granularidade fina, advinda do gerador, tinha seu tamanho incrementado demasiadamente. Na nova *interface* (Figura 3), foram inseridos campos de busca e opções de organização das colunas, permitindo que a ordem de exibição fosse facilmente alterada. Para solucionar o problema do aumento desenfreado do tamanho da lista, foram inseridas opções de paginação, permitindo ao usuário decidir quantos itens deseja visualizar. Oferecer o poder de adaptar a visualização faz-se necessário para atender a heurística de flexibilização de uso, visto que cada utilizador tem um perfil próprio de uso, e uma *interface* rígida tende a não democratizar seu uso.

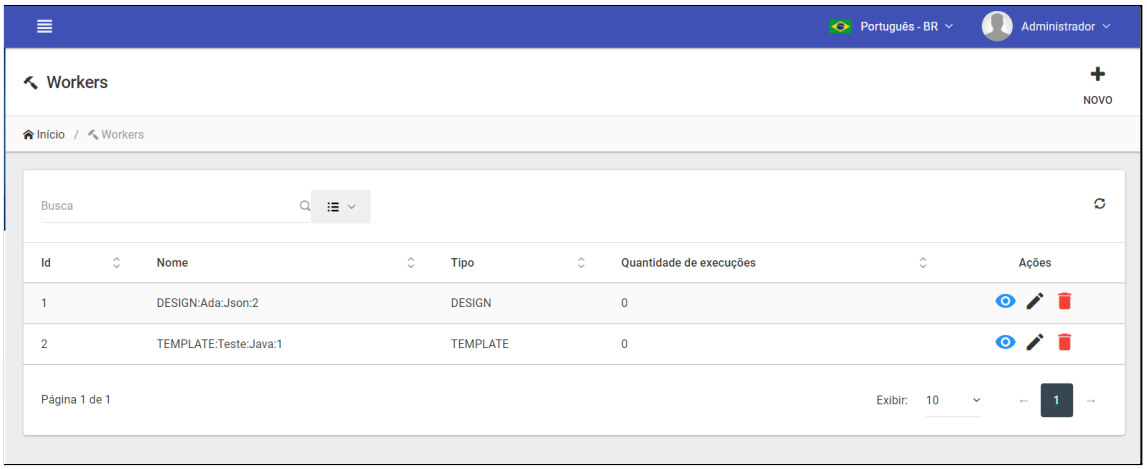

Figura 3. Nova tela de listagem de *workers*

Dando continuidade à reformulação das telas relacionadas aos *workers*, temos a tela de criação, que é uma das mais importantes do sistema, visto que é onde o desenvolvedor precisa inserir as informações e código dos *templates*, empregando assim uma grande quantidade de tempo. Na *interface* anterior (Figura 4), a tela de criação seguia o *layout* de formulário, porém, não fornecia validações claras acerca do preenchimento dos campos, o que induzia o usuário a erros. A falta de detalhamento nas mensagens de *status* feria a heurística de recuperação de erros, uma vez que o usuário recebia um erro genérico e precisava entender por si próprio o que causava tal comportamento. O espaço para inserção de código também não permitia uma boa visualização do conteúdo, exigindo a utilização de ferramentas externas como soluções paliativas. Para buscar correção para esse problema (Figura 5), dividiu-se o processo de criação em dois passos (*steps*), um contendo as informações de identificação do

workers, e outro apenas para a inserção de código, sendo este último objetivado a fornecer liberdade ao usuário de redigir seu código dentro da própria ferramenta e fornecer visibilidade do *status* do sistema.

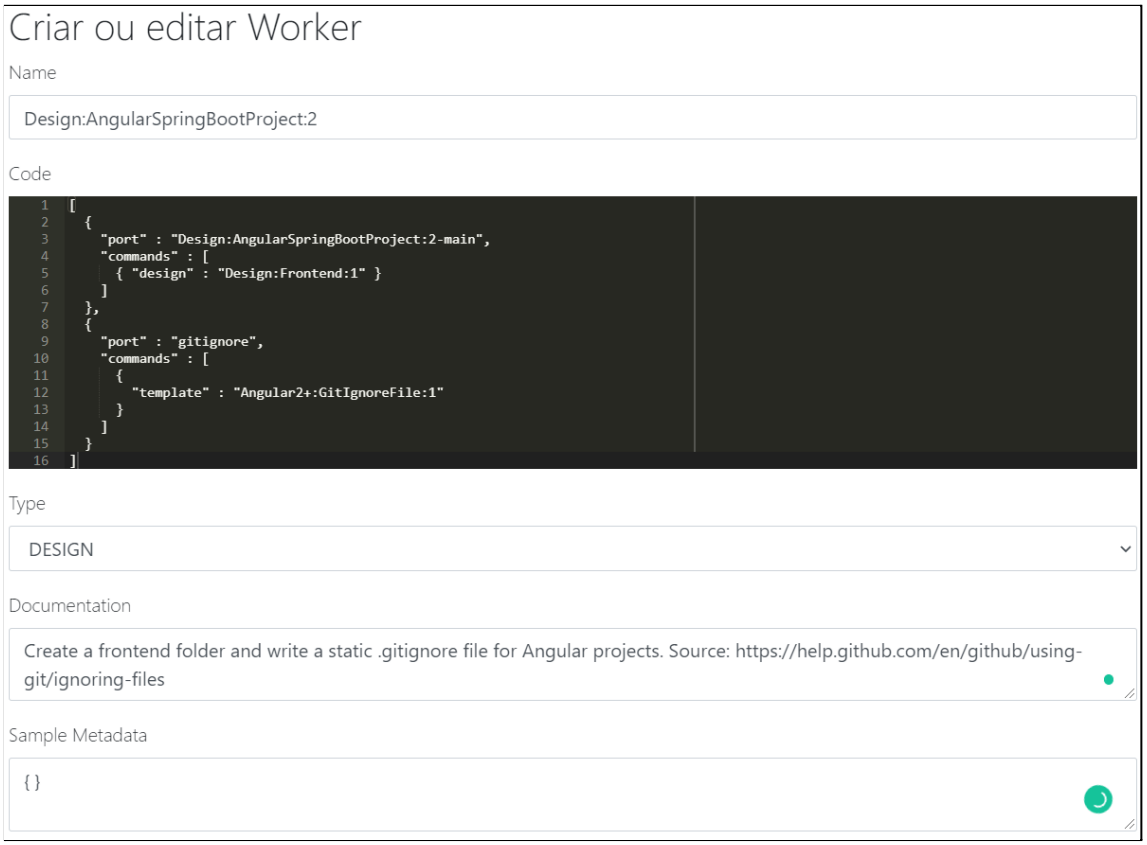

Figura 4. Tela antiga para a criação de *workers*

Um outro problema da tela de criação, era a necessidade de montagem de um nome único que seguisse o padrão **TIPO:Tecnologia:Nome:Versão**. Porém, com a falta de validação, qualquer valor poderia ser inserido, causando dificuldades na identificação e padronização de acordo com as regras da ferramenta. Para solucionar esse problema, subdividiu-se o campo (Figura 6), o que permitiu a validação individual, exibindo mensagens de erros concisas quando os tipos incorretos de informações eram inseridos, ou campos obrigatórios não eram preenchidos. Com o uso de mensagens de erros bem definidas e autoexplicativas, o usuário tende a compreender as causas dos erros e corrigi-los sem demandar muito esforço.

Figura 5. Nova tela de criação de *workers* - Espaço de código

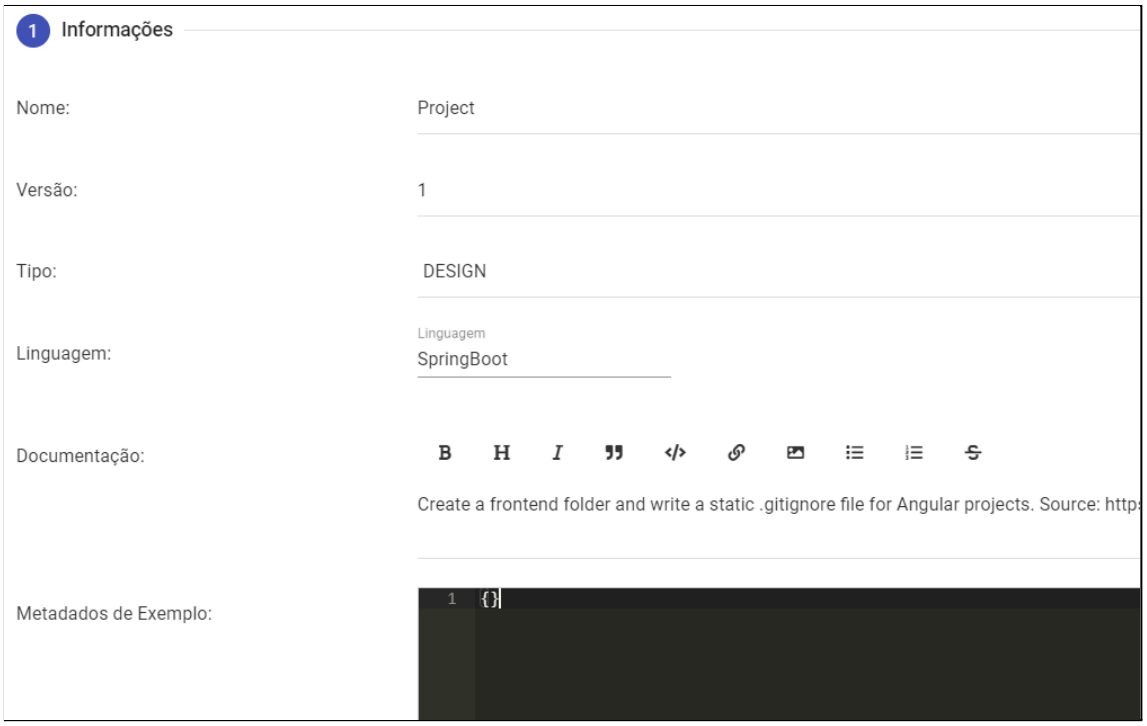

Figura 6. Nova tela de criação de *workers* - Campos de identificação

A tela de visualização dos *workers* (Figura 7) era uma das maiores reclamações dos usuários, visto que as informações eram exibidas em lista e o código em um campo

de texto que ignorava a formatação dos arquivos, causando uma enorme dificuldade de visualização e compreensão.

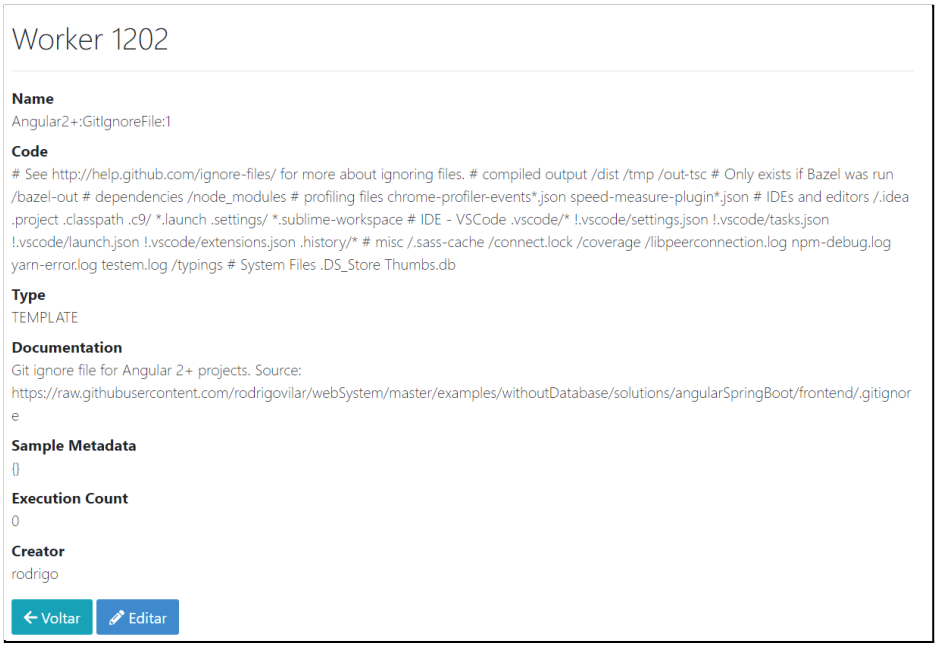

Figura 7. Antiga tela de visualização de *workers*

Na nova tela (Figura 8), buscou-se isolar as informações em abas, permitindo ao usuário selecionar o que ele deseja ver, evitando o excesso de informações. Para o problema da exibição do código utilizou-se de uma ferramenta que mantém a indentação do código e com suporte a *highlight*, permitindo visualizar estruturas das linguagens de programação. Para essa tela, buscou-se aplicar os conceitos da heurística de estética e design minimalistas, a fim de reduzir o tempo e esforço necessários para obter informações específicas.

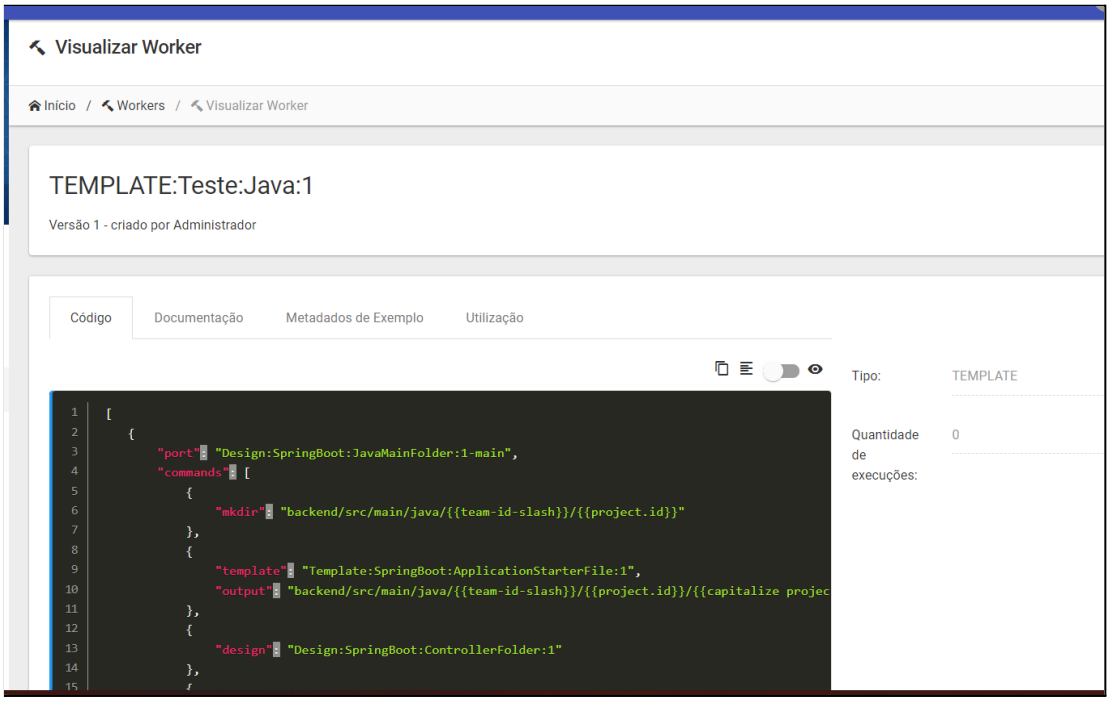

Figura 8. Nova tela de visualização de *workers*

As telas das demais entidades da ferramenta, como *projects* e *manufactures*, seguiram os mesmo princípio de *interface* e usabilidade, já que compartilham das mesmas características.

5. Avaliação

No processo avaliativo, uma análise empírica foi realizada na nova interface, para compreender se esta estava de acordo com as heurísticas apresentadas. Cada tela foi avaliada individualmente, analisando-se quais heurísticas foram aplicadas. A tabela a seguir (Tabela 1) apresenta os problemas de usabilidade que foram marcados como resolvidos.

Tabela 1. Problemas resolvidos na nova interface

Módulo	Problema
Listagem de Worker	Falta de filtros e paginação
Listagem de Worker	Falta de padrão na organização das listas
Criação/Edição de Worker	Falta de validação nos campos
Criação/Edição de Worker	Falta de mensagens de erro e mensagens sem clarezas
Criação/Edição de Worker	Campos desnecessários
Criação/Edição de Worker	Campo de código sem suporta a tabulação e com péssima visibilidade
Visualização de Worker	Muitas informações na mesma página
Visualização de Worker	Código do worker exibido sem formatação
Listagem de Manufatura	Falta de filtros e paginação
Listagem de Manufatura	Falta de padrão na organização das listas
Criação/Edição de Manufatura	Falta de validação nos campos
Criação/Edição de Manufatura	Falta de mensagens de erro e mensagens sem clarezas
Criação/Edição de Manufatura	Campos desnecessários
Visualização de Manufatura	Resultado da execução exibido sem formatação
Listagem de Projetos	Falta de filtros e paginação
Listagem de Projetos	Falta de padrão na organização das listas
Criação/Edição de Projetos	Falta de validação nos campos
Criação/Edição de Projetos	Falta de mensagens de erro e mensagens sem clarezas
Criação/Edição de Projetos	Campos desnecessários

O resultado da avaliação demonstra que todos os problemas, listados no início do desenvolvimento do trabalho, foram corrigidos. Contudo, a análise também explicitou novos pontos de melhoria, que devem ser melhor avaliados e corrigidos em trabalhos

futuros. Testes com os usuários devem ser realizados, a fim de compreender de fato qual a contribuição da nova interface para estes, visto que são, em suma, os mais afetados pelos efeitos da usabilidade.

6. Conclusão

A construção deste trabalho, evidenciou que a usabilidade quando aplicada incorretamente, tende a impactar negativamente na produtividade. Uma ferramenta que busca fornecer produtividade precisa ter sua usabilidade bem aplicada, do contrário pode, em alguns casos, ter efeito contrário.

Outro ponto evidenciado, é que, na ausência de quórum, análises empíricas são suficientemente aplicáveis, uma vez que o norte desta seja embasado em regras objetivas, como as heurísticas de Nielsen. A realização de uma nova avaliação, uma vez terminado o desenvolvimento, é de suma importância para compreender se os problemas foram resolvidos e visualizar novos, que tendem a surgir.

Referências

Bevan, N. (2001). International standards for HCI and usability. International Journal of Human-Computer Studies, 55(4), 533–552. doi:10.1006/ijhc.2001.0483

Boulic, R., Renault, O. (1991) "3D Hierarchies for Animation", In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

Case, A. (1985). Computer-aided software engineering (CASE). ACM SIGMIS Database, 17(1), 35–43. doi:10.1145/1040694.1040698

Gamma, E. (1995) "Design patterns: elements of reusable object-oriented software". Pearson Education India.

Gračanin, D., Matković, K., & Eltoweissy, M. (2005). "Software visualization. Innovations in Systems and Software Engineering", 1(2), 221–230. doi:10.1007/s11334-005-0019-8

Hegedűs, P. et al. (2012). Myth or reality? analyzing the effect of design patterns on software maintainability. In: Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity. Springer, Berlin, Heidelberg, p. 138-145. doi: 10.1007/978-3-642-35267-6_18

Koschke, Rainer (2003). Software visualization in software maintenance, reverse engineering, and re‑engineering: a research survey. Journal of Software Maintenance and Evolution: Research and Practice, v. 15, n. 2, p. 87-109.

Nielsen, J. (1994) "Enhancing the explanatory power of usability heuristics." In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems. p. 152-158.

Rich, C., Waters, R. C. (1988). "Automatic programming: Myths and prospects." Computer, v. 21, n. 8, p. 40-51.

Vilar, R., Oliveira, D.; Almeida, H. (2015) “Rendering patterns for enterprise applications”. In: Proceedings of the 20th European Conference on Pattern Languages of Programs. p. 1-17.

Wagner, S., Ruhe, M. (2018) “A systematic review of productivity factors in software development” in arXiv preprint arXiv:1801.06475.