

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**BRUNO ROBERTO BÚRIGO**

**FLOW: Ferramenta web para criação de fluxogramas executáveis**

**TRABALHO DE CONCLUSÃO DE CURSO**

**CORNÉLIO PROCÓPIO  
2014**

**BRUNO ROBERTO BÚRIGO**

**FLOW: Ferramenta web para criação de fluxogramas executáveis**

Trabalho de conclusão de curso de graduação, apresentado a disciplina de Trabalho de Diplomação do curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do grau de Tecnólogo.

Orientador: Adriano Rívoli da Silva

**CORNÉLIO PROCÓPIO**  
**2014**

## AGRADECIMENTOS

A todos os professores da UTFPR Cornélio Procópio, pelo ensino transmitido e pelo exemplo passado.

Ao meu orientador, pelo (excesso) de confiança depositado em mim e pela constante ajuda oferecida.

A minha família, pela união, suporte, apoio e exemplos.

A todos os amigos que fiz na cidade de Cornélio Procópio, cujos nomes não serão listados, pois poderiam preencher vários pergaminhos. Pelos bons momentos, pelos empréstimos de placas, pelos passeios em cima de trens, pelas mentiras contadas, pelos migués aplicados, pelas coisas exóticas vistas, pelos churrascos firmeza, pelas festas aruêra, pelas colheres de canela em pó ingeridas e pelos **refrescos** tomados (vários).

Ao Engenheiro Analista, o cachorro mais desanimado que já pairou sobre a terra. E também um diabinho.

Ao meu irmão Tayllan, *we shall rule this world*.

A minha namorada Paula, por ser minha namorada.

BÚRIGO, Bruno Roberto. **Flow**: Ferramenta web para criação de fluxogramas executáveis. 2014. 64. Trabalho de Conclusão do Curso de Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2014.

## RESUMO

Algoritmos e lógica de programação são os mais importantes conceitos necessários para que uma pessoa possa se tornar um bom programador e, embora essenciais, estes não são assuntos simples. Dentre os diversos métodos utilizados em auxílio ao ensino e aprendizado desses conceitos, os fluxogramas se destacam como os mais utilizados devido a sua boa aceitação pelos alunos, o que ocorre em função de se tratarem de representações gráficas de algoritmos. Contudo, a utilização de fluxogramas como mera abstração gráfica pode ser vista como uma limitação desnecessária. Nesse contexto foi proposta e desenvolvida uma ferramenta *web* que permite ao usuário a criação de fluxogramas que representem algoritmos e podem ser executados de modo semelhante a um programa de computador. Esta execução de fluxogramas da ferramenta pode ser realizada de modo completo, no qual todo o fluxograma é percorrido e executado do início ao fim, e também de modo passo a passo, onde o usuário tem controle sobre os passos da execução; sendo o modo passo a passo especialmente eficaz para a demonstração do funcionamento lógico de um algoritmo. A ferramenta propicia também um alto nível de liberdade ao usuário quanto a construção dos fluxogramas e, por ter sido desenvolvida para ambientes *web*, apresenta várias possibilidades de expansão futura. Sua existência gera a possibilidade de auxiliar no processo de ensino e aprendizado de algoritmos.

**Palavras-chave:** Fluxogramas. Execução passo a passo. Algoritmos.

BÚRIGO, Bruno Roberto. **Flow**: Ferramenta web para criação de fluxogramas executáveis. 2014. 64. Trabalho de Conclusão do Curso de Análise e Desenvolvimento de Sistemas – Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2014.

## ABSTRACT

Algorithms and programming logic are the most important concepts that one person needs in order to become a good programmer but, although essential, those are not simple subjects. Among the different methods used in aid to the teaching and learning of those subjects, flowcharts stand out as the most used due to its good acceptance among students, what occurs due the fact that flowcharts are a graphical representation of algorithms. However, the use of flowcharts as a mere graphical abstraction can be seen as an unnecessary limitation. In this context was proposed and developed a web tool that allows a user to create flowcharts that can be run similarly to computer programs. This flowchart execution of the tool can be performed in a complete mode, in which all the flowchart is traversed from the beginning to the end and in a step by step mode, in which the user as complete control over the steps of the flowchart execution. The step by step mode is especially effective to demonstrate the logical operation of an algorithm. The tool also provides to the user a high level of freedom in the construction of flowcharts and because the tool has been developed for web environments, presents several possibilities of future expansion. The existence of this tool raises the possibility of assisting the teaching and learning process of algorithms.

**Keywords:** Flowcharts. Step by step execution. Algorithms.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Raptor .....	13
Figura 2 – Funcionamento lógico de um fluxograma.....	18
Figura 3 – Fases do PXP .....	24
Figura 4 - Ferramenta Jenkins .....	26
Figura 5 - Documentação gerada pelo phpDox.....	27
Figura 6 – Código inicial da função loadConnections.....	28
Figura 7 – Código da função loadConnections refatorada .....	30
Figura 8 – Fragmento de relatório do programa Codeception.....	32
Figura 9 – Relatório gerado pelo programa phpcs .....	33
Figura 10 - Relatório do programa Plato .....	35
Figura 11- Spike .....	36
Figura 12 - ClockingIT, atividades .....	37
Figura 13 - Diagrama de casos de uso .....	38
Figura 14 – Diagrama físico do banco de dados.....	39
Figura 15 - Diagrama de classes da framework desenvolvida .....	40
Figura 16 - Trigger de validação de elemento .....	46
Figura 17 – Flow, tela inicial .....	48
Figura 18 – Flow, Cadastro .....	49
Figura 19 – Flow, tela pós-login .....	50
Figura 20 – Flow, área do fluxograma expandida .....	51
Figura 21 – Flow, Configurações e execução .....	51
Figura 22 - Flow: execução e busca.....	52
Figura 23 – Protótipo de tela final da home.....	62
Figura 24 – Protótipo de tela final do fluxograma .....	63

## LISTA DE TABELAS

Tabela 1 - Símbolos geométricos para fluxogramas .....	16
Tabela 2 - Operadores .....	17
Tabela 3 - Cronograma .....	24

## LISTA DE SIGLAS

*API - Application Programming Interface*  
*ATDD - Acceptance Test Driven Development*  
*CRUD - Create Retrieve Update Delete*  
*CSS - Cascading Style Sheets*  
*DOM – Document Object Model*  
*HTML - HyperText Markup Language*  
*IDE – Integrated development environment*  
*JSON – JavaScript Object Notation*  
*MVC - Model View Controller*  
*PHP - PHP: Hypertext Processor*  
*PHPCPD - PHP Copy/Paste Detector*  
*PHPCS - PHP Code Sniffer*  
*PHPDOX - PHP Documentation Generator*  
*PHPLOC - PHP Lines of Code*  
*PSP - Personal Software Process*  
*PXP - Personal Xtreme Programming*  
*SGBD - Sistema Gerenciador de Bancos de dados*  
*SQL - Structured Querying Language*  
*TDD – Test Driven Development*  
*UI - User Interface*  
*UML - Unified Modeling Language*  
*XML - eXtensible Markup Language*  
*XP - Extreme Programming*



## SUMÁRIO

1. INTRODUÇÃO .....	11
1.1 OBJETIVO .....	12
1.2 JUSTIFICATIVA .....	12
1.3 ORGANIZAÇÃO DO TRABALHO .....	14
2. FUNDAMENTAÇÃO TEÓRICA .....	15
3. TECNOLOGIAS UTILIZADAS .....	19
3.1 LINGUAGENS:.....	19
3.2 FERRAMENTAS / PROGRAMAS .....	20
3.3 BIBLIOTECAS.....	22
4. DESENVOLVIMENTO DO PROJETO .....	23
4.1 METODOLOGIA .....	23
4.2 CRONOGRAMA.....	24
4.3 PRÁTICAS UTILIZADAS .....	25
4.3.1 Integração Contínua .....	25
4.3.2. Revisão de Código.....	27
4.3.3. Refatoração e Otimização .....	27
4.3.4. Desenvolvimento Orientado a Testes .....	31
4.3.5. Pequenas Versões.....	32
4.3.6. Padronização de Código.....	33
4.3.7. Mensuração de Tamanho de código.....	33
4.3.8. Registro de Defeitos .....	35
4.3.9. Spike Solutions .....	35
4.3.10. Planejamento de Atividade .....	36
4.3.11. Proposta de Melhoramento de Processo .....	37
4.4 DESENVOLVIMENTO DA FERRAMENTA.....	38

4.4.1 Levantamento de Requisitos .....	38
4.4.2 Planejamento .....	38
4.4.3 Desenvolvimento de um <i>Framework</i> .....	40
4.4.4 Iterações .....	41
4.4.4.1 Persistência dos elementos de um fluxograma .....	41
4.4.4.2 Execução do fluxograma .....	42
4.4.4.3 Retorno de estado .....	43
4.4.4.4 Segurança .....	44
5. FERRAMENTA FLOW .....	47
5.1 VISÃO GERAL.....	47
5.2 TELAS.....	48
6. CONSIDERAÇÕES FINAIS .....	53
7. REFERÊNCIAS BIBLIOGRÁFICAS .....	55
APÊNDICE A – Requisitos de Software .....	57
APÊNDICE B - Protótipos de tela finais .....	61
APÊNDICE C – Proposta .....	64

## 1. INTRODUÇÃO

Algoritmos são o ponto de partida no início do aprendizado de uma pessoa que almeje aprender os conceitos de programação e futuramente tornar-se um programador; como dito por CORMEN *et al.* (2009), possuir um conhecimento sólido de algoritmos é o que diferencia o verdadeiro programador de um iniciante. O problema, contudo, reside no fato de que muitas vezes esse aprendizado inicial é difícil, pois a compreensão de que são necessárias séries de instruções específicas e exatas em uma ordem bem definida e que juntas formem uma sequência lógica funcional não é algo trivial a maioria das pessoas. Ainda, segundo CHAUDHURI (2005), algoritmos tendem a ser excessivamente verbosos e, conseqüentemente, seu texto não é facilmente compreendido por todos.

Para contornar as dificuldades iniciais existentes no aprendizado de algoritmos são utilizados diversos métodos de ensino, sendo um desses o uso de fluxogramas como forma de representar graficamente o conceito. Fluxogramas tendem a ser úteis e bem aceitos por iniciantes, pois assim como colocado por BERG, FIGUEIRÓ (2006) as diferentes formas geométricas de um fluxograma implicam em ações distintas, facilitando a compreensão do conceito de um algoritmo.

Embora o uso de fluxogramas possa auxiliar o processo de aprendizado, sua utilização apenas como forma de representação abstrata de algoritmos pode ser vista como uma limitação desnecessária, pois, pseudocódigos usados na representação de algoritmos são também puramente conceituais e ainda assim existem ferramentas disponíveis que permitem a execução destes, exatamente com o intuito de demonstrar claramente o “resultado” daquela representação e se ela está correta ou não.

Diante do exposto, podemos observar as possíveis vantagens existentes no uso de fluxogramas para o ensino e aprendizado de algoritmos, estas ampliadas com a possibilidade do uso de fluxogramas que possam ser executados assim como programas de computador, sendo o desenvolvimento de uma ferramenta que permita tal a finalidade deste trabalho.

## 1.1 OBJETIVO

O foco central deste trabalho foi a construção de uma ferramenta que visa ultrapassar as limitações dos fluxogramas estáticos, possibilitando a criação de fluxogramas que possam ser executados de maneira similar a programas de computador, realizando operações lógico-matemáticas, aceitando entradas de dados por parte do usuário e exibindo saídas visíveis a este.

A execução de um fluxograma pode ser realizada de duas formas: completa, na qual todo o grafo de elementos do fluxograma é executado do início ao fim sem interrupções (excetuando-se as geradas pela necessidade de entrada de dados por parte do usuário); e passo a passo, onde um único elemento é executado por vez, sendo a ocorrência dessas execuções sequenciais controlada pelo usuário, de forma similar as ferramentas de *debug* presentes em *IDEs*. Em ambos os modos o fluxo de execução dos elementos é determinado avaliando-se cada elemento, seu significado e o pseudocódigo nele contido (quando existente).

## 1.2 JUSTIFICATIVA

Existem inúmeras ferramentas para criação de fluxogramas, tanto em ambiente *web* quanto *desktop*, contudo a grande maioria delas não é exclusivamente dedicada a este tipo de diagrama e, principalmente, não apresentam a possibilidade de sua execução, tendo sido encontrada apenas uma que disponibiliza este recurso.

A ferramenta identificada que possibilita a criação de fluxogramas executáveis denomina-se Raptor, sendo um programa de uso gratuito, distribuído livremente pelo website <http://raptor.martincarlisle.com/>. Seu enfoque central é idêntico ao proposto por este trabalho, a criação e manipulação de fluxogramas, com a possibilidade de executá-los tanto de maneira completa quanto passo a passo. Além da execução de fluxogramas, diversas funcionalidades adicionais são encontradas na ferramenta, as mais proeminentes são: exportação do algoritmo representado no fluxograma para algumas linguagens de programação, entre elas C# e Java; modos de utilização separados pela experiência do usuário (noviço, intermediário e orientado a objetos). Também relevante é o fato de a ferramenta possuir uma extensa documentação. Na Figura 1 é apresentada a tela de visualização de um fluxograma na ferramenta Raptor.

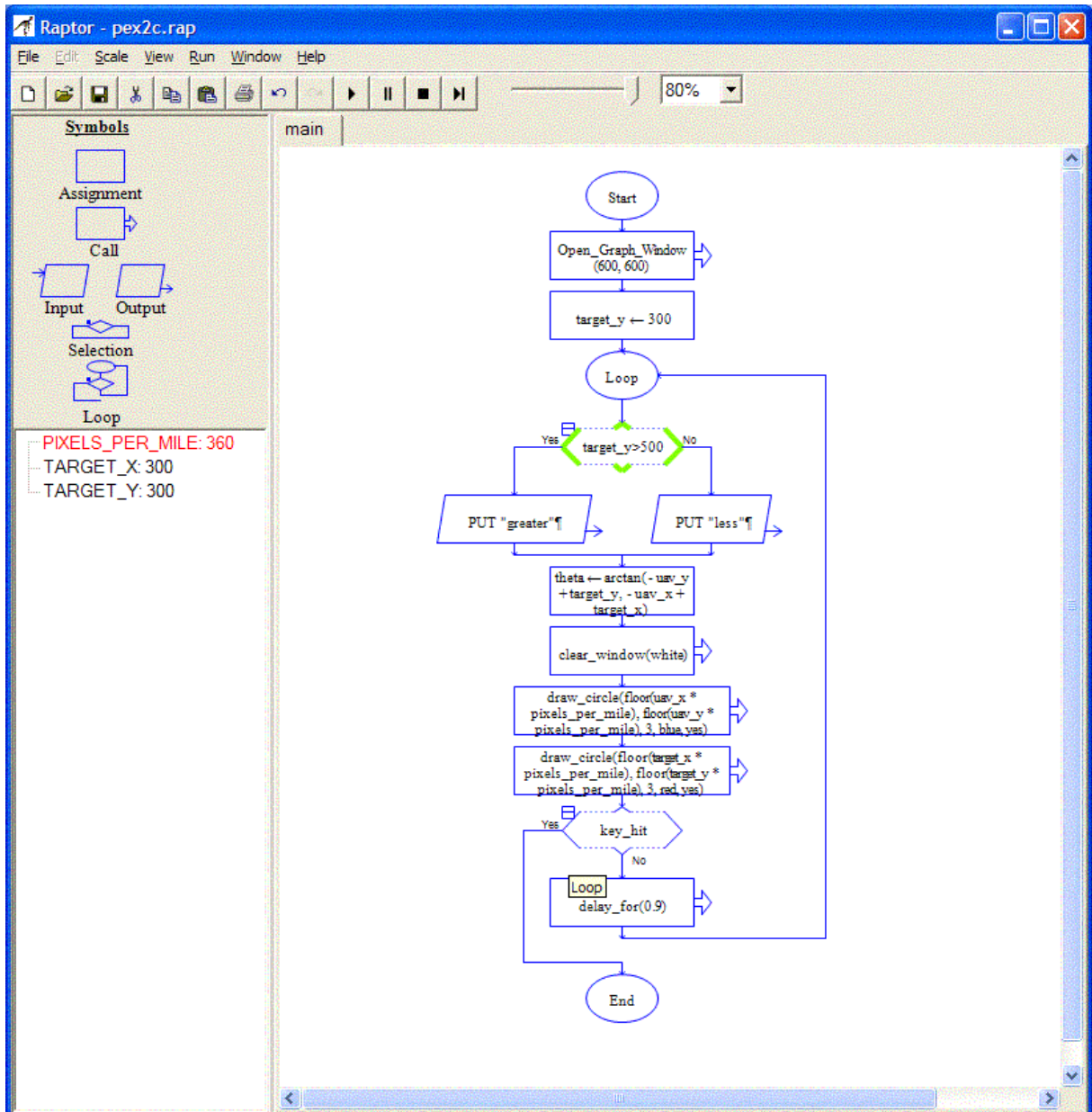


Figura 1 – Raptor

Fonte: <<http://raptor.martincarlisle.com/>> />

Após análise e uso das principais funcionalidades da ferramenta Raptor (execução, exportação de código e diferentes modos de utilização), nenhum problema foi detectado, podendo-se constatar que ela cumpre sua proposta. Contudo, mesmo com a aparente consistência observada na ferramenta, esta apresenta suas falhas e lacunas. A principal sendo sua baixa portabilidade, pois sua codificação foi realizada nas linguagens A# e C# na plataforma .NET. Tal característica torna a ferramenta dependente do sistema operacional Windows e

impede seu uso em outros sistemas operacionais relevantes, tais como Linux e MacOS. Outro ponto contra o programa Raptor, embora este possa ser considerado de menor importância e até mesmo subjetivo, é seu baixo apelo visual.

A ferramenta construída neste trabalho denomina-se Flow e sua codificação foi realizada com linguagens e tecnologias de ambiente *web*, sendo assim sua utilização independe de sistema operacional, podendo abranger um público maior que a ferramenta Raptor. Além disso, por ser uma ferramenta *web*, o programa Flow apresenta inúmeras possibilidades para expansões futuras, principalmente em relação a questões de colaboração entre usuários, as quais não são possíveis, ou ao menos possuem grandes limitações, na ferramenta Raptor.

Por fim, é necessário enfatizar as possibilidades de auxílio ao ensino e aprendizado de algoritmos e lógica de programação que a ferramenta Flow visa prover, sendo estas a maior motivação e justificativa para a realização deste trabalho.

### 1.3 ORGANIZAÇÃO DO TRABALHO

A organização dos subseqüentes tópicos deste trabalho está dividida da seguinte forma:

No capítulo 2 é apresentada a fundamentação teórica na qual se baseia este trabalho.

No capítulo 3 são enumeradas e explanadas as tecnologias que foram utilizadas no desenvolvimento.

No capítulo 4 encontra-se o detalhamento de como se deu o desenvolvimento do projeto e que aspectos foram enfatizados.

No capítulo 5 é demonstrado o resultado alcançado neste trabalho, a ferramenta Flow.

No capítulo 6 encontra-se a conclusão.

No capítulo 7 encontram-se as referências bibliográficas.

## 2. FUNDAMENTAÇÃO TEÓRICA

### Algoritmo

Um algoritmo pode ser descrito como uma série de instruções lógicas organizadas de modo a solucionar um problema específico. Ainda segundo CORMEN et al.(2009) temos a definição de que um algoritmo é qualquer procedimento computacional bem definido que receba valores de entrada, realize uma série de operações e produza valores de saída, a qual se encaixa perfeitamente no contexto de algoritmos utilizado para este trabalho.

### Fluxograma

Fluxogramas são diagramas que utilizam figuras geométricas para ilustrar uma série de passos a serem percorridos a fim de se completar uma tarefa. Esta definição é extremamente similar a de um algoritmo, tanto que segundo CHAUDHURI (2005), temos “[...] *Formally speaking, a flowchart is a diagrammatic representation of the steps of an algorithm.*”<sup>1</sup>

As figuras geométricas de um fluxograma são utilizadas como meio de representação do fluxo de ações a serem realizados por este, cada forma possui um nome e é responsável por ilustrar uma determinada operação. Elas são conectadas através de setas a fim de representar o fluxo de execução da tarefa que está sendo realizada pelo fluxograma em questão.

A ISO 5807-1985 define a existência de cinco tipos distintos de fluxogramas, são eles:

1. *data flowcharts* (diagramas de fluxo de dados);
2. *program flowcharts* (diagramas de fluxo de programa);
3. *system flowcharts* (diagramas de fluxo de sistema);
4. *program network charts* (diagramas de programas de rede);
5. *system resource charts* (diagramas de recursos de sistema).

Apesar de sua existência, essa norma é largamente ignorada devido a inúmeros fatores, entre eles, o fato de sua última revisão ter sido realizado em 1985, e como apontado por MANZANO (2004), ela é extremamente genérica e não definem de maneira clara os critérios para a elaboração de diagramas que

---

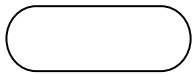

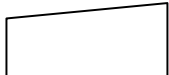

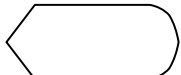
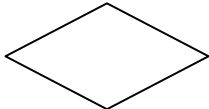
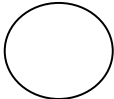
<sup>1</sup> Formalmente falando, um fluxograma é uma representação em forma de diagrama dos passos de um algoritmo.

representem a linha de raciocínio lógico a ser utilizada por um programador de computadores, além de sua definição para os símbolos geométricos ser demasiado simplista.

Como para este trabalho fez-se necessário um alto nível de detalhamento e asserção sobre os símbolos geométricos de um fluxograma e como esses podem ser utilizados em conjunto para formar um diagrama válido, a norma ISO 5807-1985 foi tomada meramente como base, não servindo como modelo literal para o *software* desenvolvido.

Na Tabela 1 são apresentados os símbolos geométricos definidos para utilização no fluxograma.

**Tabela 1 - Símbolos geométricos para fluxogramas**

Símbolo	Nome	Significado
	Início	Simboliza o início de um fluxograma.
	Fim	Simboliza o fim de um fluxograma.
	Entrada manual de Dados	Simboliza entrada manual de dados pelo usuário, previsivelmente pelo teclado.
	Processamento	Representa a execução de operações lógico-matemáticas ou a atribuição de valores a variáveis.
	Exibição / Saída de Dados	Simboliza a exibição de valores de forma que o usuário possa visualiza-los.
	Decisão	Simboliza um teste lógico condicional, provendo dois caminhos diferentes a serem seguidos.
	Conector	Tem a função de unir fluxos de execução previamente separados por uma ou mais operações de <b>decisão</b> .



	Seta	Representa a ligação entre dois elementos. A direção indica o fluxo de execução.
---	------	--

A conexão desses símbolos em uma sequência lógica aceitável, juntamente com a adição de trechos de pseudocódigos, possibilita a execução do fluxograma.

### Sintaxe

Em relação a linguagens de programação, sintaxe é essencialmente uma série de regras gramaticais que determinam se um determinado trecho de código está correto (MAK, 2009).

Visto que a utilização pura dos elementos de um fluxograma inviabiliza a criação de algoritmos, foi necessário possibilitar a adição de trechos de pseudocódigos nestes. Para tal foi definida a sintaxe do pseudocódigo a ser utilizado em conjunto com os fluxogramas, estando os operadores lógico-matemáticos presentes na Tabela 2:

**Tabela 2 - Operadores**

Operador	Função
+	Adição
-	Subtração
/	Divisão
*	Multiplicação
%	Módulo, resto de divisão
!	Negação
&&	“E” lógico
	“OU” lógico
<- ou =	Atribuição
==	Igual a
!=	Diferente de
<	Menor que
>	Maior que
<=	Menor ou igual a

>=	Maior ou igual a
----	------------------

Na Figura 2 encontra-se uma exemplificação de como os elementos do fluxograma podem ser conectados entre si, de modo a formarem uma sequência lógica válida. Nela também são demonstradas formas válidas de utilização dos operadores e do pseudocódigo.

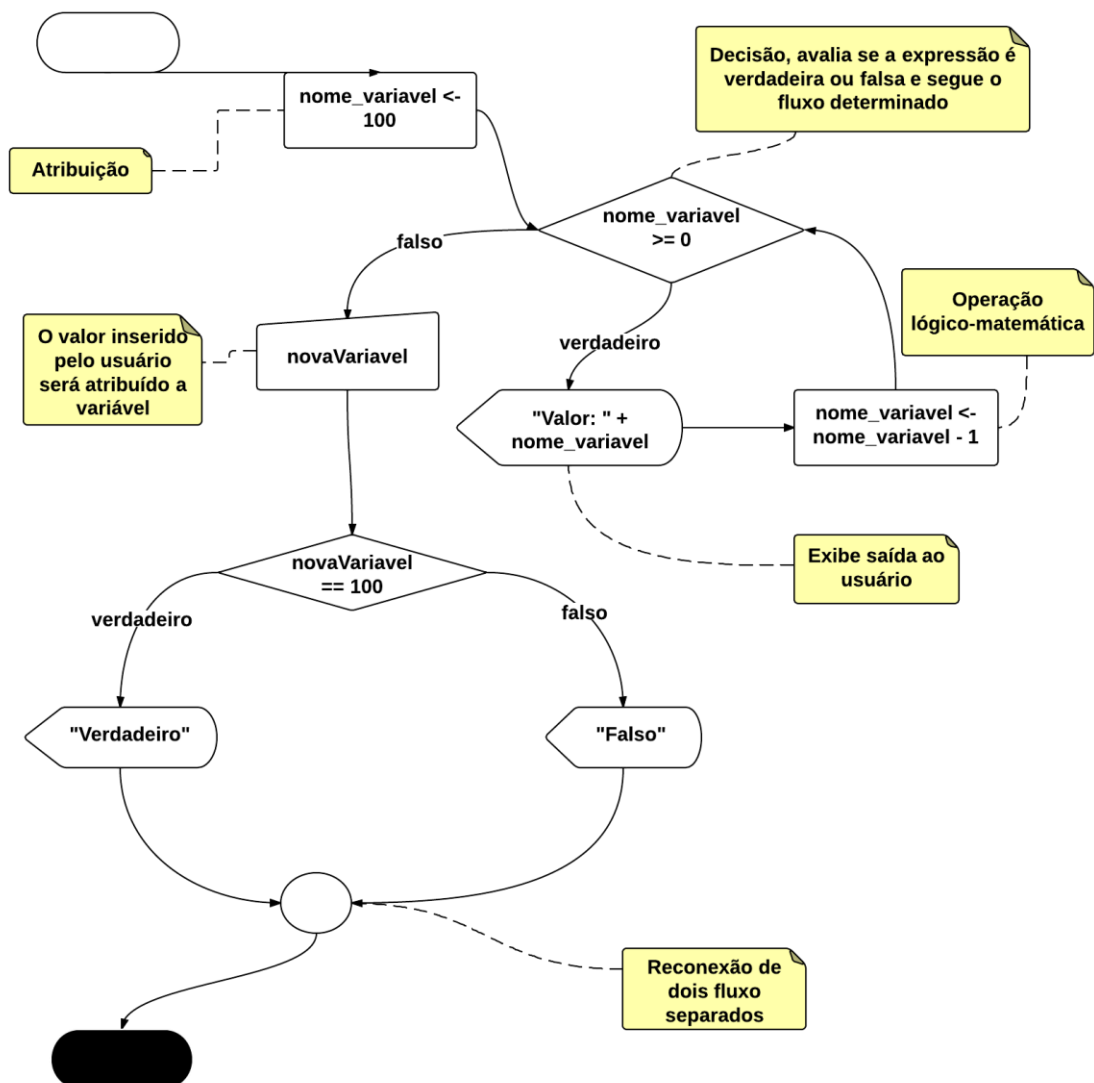


Figura 2 – Funcionamento lógico de um fluxograma

### 3. TECNOLOGIAS UTILIZADAS

Para o desenvolvimento deste trabalho procurou-se utilizar, além das tecnologias básicas (*HTML*, *CSS*, *JavaScript* e *PHP*), o maior número possível de bibliotecas e ferramentas que pudessem agregar qualidade e velocidade no desenvolvimento.

Muitas das ferramentas e bibliotecas selecionadas eram previamente desconhecidas, logo foi necessário dispendir uma quantidade considerável de tempo no estudo e aprendizado destas.

#### 3.1 LINGUAGENS:

- **Ajax:** Não constitui uma linguagem, mas sim uma série de técnicas de programação *web* que utilizam *scripts* para interagir com o protocolo *HTTP* a fim de carregar ou enviar dados para o servidor conforme necessário, sem que isso cause o recarregamento de toda a página (FLANAGAN, 2011);
- **CSS:** *Cascading Style Sheets*, linguagem utilizada para dar estilo à página, compreendendo a parte visual desta;
- **HTML:** linguagem de marcação interpretada pelo navegador, utilizada para estruturar as páginas;
- **JavaScript:** Linguagem de *script* interpretada pelo browser, utilizada para gerar comportamento dinâmico no lado do cliente;
- **JSON:** *JavaScript Object Notation*, formato de texto para troca de dados. Objetiva tornar possível sua leitura tanto para máquinas como para humanos, similarmente ao *XML*, com a diferença de possuir, teoricamente, um melhor nível de legibilidade;
- **PHP:** Linguagem de *script* utilizada no lado do servidor com o intuito de gerar conteúdo dinâmico;
- **SQL:** *Structured Query Language*, linguagem de pesquisa inspirada em álgebra relacional, utilizada para manipular e retornar dados do banco de dados;
- **UML:** *Unified Modeling Language*, linguagem de modelagem de diagramas para representação de sistemas orientados a objetos;

- **XML: Extensible Markup Language**, linguagem de marcação que define uma série de regras de codificação de documentos em um formato que possa ser lido tanto por máquinas quando por humanos.

### 3.2 FERRAMENTAS / PROGRAMAS

- **Apache Ant:** Ferramenta de uso gratuito e *open source* que permite a definição de diversas tarefas e programas em um arquivo *XML*, bem como execução destes. Foi utilizada como forma de definir e disparar a execução dos programas utilizados em conjunto com a ferramenta Jenkins. Página oficial: <http://ant.apache.org/>;
- **Apache HTTP Server:** Também conhecido como Apache, é um servidor *HTTP* (ou servidor *web*) de uso gratuito e com código *open source*. Página oficial: <http://httpd.apache.org/>;
- **Astah Professional:** Ferramenta de modelagem de diagramas *UML*, sendo neste projeto utilizada para criação dos diagramas de caso de uso, diagramas de classes e diagramas navegacionais. É uma ferramenta proprietária e de uso pago que, contudo, prove uma licença gratuita com duração de um ano para estudantes, tendo sido esta licença adquirida para o uso do programa no trabalho. Página: <http://astah.net/editions/professional/>;
- **Clocking IT:** Ferramenta *web* gratuita para gerenciamento de atividades. Página: <http://www.clockingit.com/>;
- **Codeception:** *Framework* de testes para *PHP*. Neste projeto foi utilizado para, em conjunto com o *framework* Selenium, executar testes de aceitação. Página: <http://codeception.com/>;
- **Dropbox:** Serviço gratuito de armazenamento de arquivos na nuvem. Foi utilizado como local de armazenamento do repositório criado no projeto com a ferramenta Git. Página oficial: <https://www.dropbox.com/>;
- **Git:** Sistema de controle de versionamento distribuído. Utilizado para manter o versionamento do código produzido durante o trabalho. A ferramenta é gratuita e *open source*. Página oficial: <http://git-scm.com/>;
- **Jenkins:** Sistema de integração contínua, gratuito e *open source*. O funcionamento e uso da ferramenta são devidamente delineados no capítulo 4.3.1 Integração Contínua. Página oficial: <http://jenkins-ci.org/>;

- **Lucidchart:** Ferramenta web para criação de diversos tipos de diagramas, entre eles os protótipos de tela que foram construídos neste trabalho. É uma ferramenta proprietária e seu uso é pago. A *home page* da ferramenta é: <https://www.lucidchart.com/>;
- **Netbeans:** IDE gratuita e *open source*. Página oficial da ferramenta: <https://netbeans.org/>;
- **pgModeler:** Ferramenta para modelagem de diagramas físicos para o SGBD PostgreSQL. Possibilita a geração de código SQL através de diagramas criados e também a engenharia reversa de uma *database* para um diagrama. A ferramenta é gratuita e possui código *open source*, a página oficial do projeto é: <http://www.pgmodeler.com.br/>;
- **plato:** Permite a mensuração de tamanho e análise estática de código JavaScript. Uso gratuito e código *open source*. Página: <https://github.com/es-analysis/plato>;
- **PHPCPD:** Ferramenta para detecção de código duplicado em arquivos PHP. Gratuita e de código *open source*, pode ser encontrada em: <https://github.com/sebastianbergmann/phpcpd>;
- **PHPDOX:** Ferramenta para geração de documentação de código PHP. Uso gratuito e código *open source*. Página oficial: <http://phpdox.de/>;
- **PHPLOC:** Ferramenta para mensuração de tamanho e análise estática de projetos PHP. É de uso gratuito e código *open source*. Página da ferramenta: <https://github.com/sebastianbergmann/phploc>;
- **PHP\_CodeSniffer:** Também referenciado como *phpcs*, permite detectar arquivos PHP, CSS e JavaScript que contenham código fora de um padrão, o qual é definido por meio de um arquivo XML que contém as especificações de acordo com as normas da ferramenta. Seu uso é gratuito e seu código *open source*, podendo este ser encontrado em: [https://github.com/squizlabs/PHP\\_CodeSniffer](https://github.com/squizlabs/PHP_CodeSniffer);
- **PostgreSQL:** Sistema Gerenciador de Bancos de Dados gratuito e *open source*. Página oficial do SGBD: <http://www.postgresql.org/>;
- **Selenium:** Framework gratuita e *open source* que disponibiliza uma linguagem específica para testes, sendo que estes podem ser executados diretamente em *browsers*. Página oficial: <http://docs.seleniumhq.org/>.

### 3.3 BIBLIOTECAS

- **Alertify:** Biblioteca de *UI* para JavaScript, disponibiliza componentes e efeitos para alertas, notificações e caixas de texto. Lançada sob a licença *MIT* e com código *open source*, o qual pode ser encontrado em <https://github.com/fabien-d/alertify.js/tree/0.3.11>;
- **jQuery:** Biblioteca JavaScript de múltiplas utilidades. Possui licença *MIT* e seu código é *open source*, podendo ser encontrado em <https://github.com/jquery/jquery>;
- **jQueryUI:** Biblioteca de *UI* para JavaScript, construída sobre a biblioteca jQuery; disponibiliza diversos componentes e efeitos de interface gráfica. Possui licença *MIT* e seu código é *open source*, podendo ser encontrado em <https://github.com/jquery/jquery-ui>;
- **jsPlumb:** Biblioteca para JavaScript, prove uma *API* para realizar a conexões entre elementos de um documento *HTML*. Foi a biblioteca de maior importância para o projeto por ter sido aquela que possibilitou a criação e manipulação de conexões entre os elementos de um fluxograma. Seu código é *open source* e possui uma licença dupla (*MIT* e *GPLv2*). O repositório do projeto, bem como o código fonte podem ser encontrados em <https://github.com/sporritt/jsplumb/>;
- **Mustache.php:** Biblioteca de *templates* para *PHP*, utilizada para produzir uma melhor estruturação e legibilidade no *HTML* da aplicação. Possui código *open source*, está disponível em <https://github.com/bobthecow/mustache.php>, e é distribuída sob a licença *MIT*;
- **RedBeanPHP:** Biblioteca de mapeamento objeto-relacional para *PHP*. Possui código *open source* e é lançada sob a licença *NEW BSD LICENSE*. Seu código pode ser encontrado em <https://github.com/gabordemooij/redbean>;
- **Semantic UI:** Biblioteca de componentes de *UI*, utilizada para tornar mais pratico o desenvolvimento da interface gráfica do projeto. Seu código é *open source*, sendo este acessível em <https://github.com/Semantic-Org/Semantic-UI>, e é disponibilizada sob a licença *MIT*.

## 4. DESENVOLVIMENTO DO PROJETO

### 4.1 METODOLOGIA

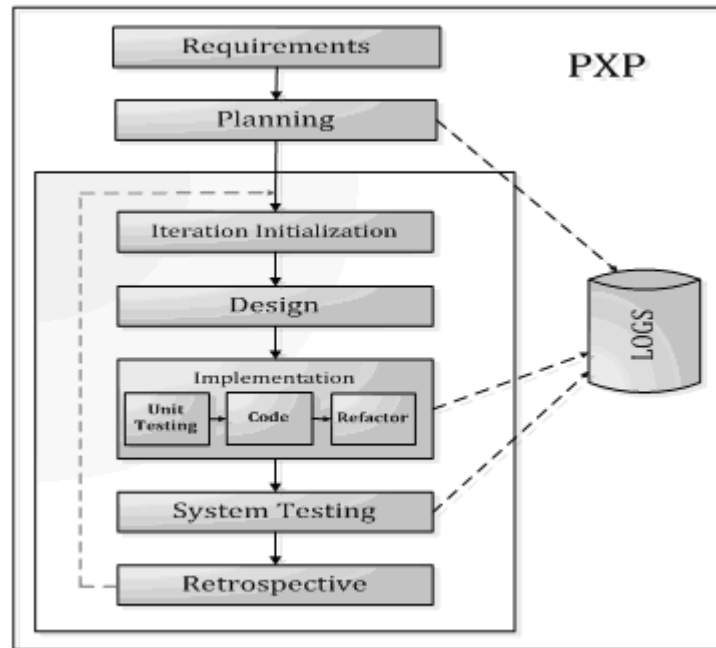
Como o projeto consistiu essencialmente da elaboração e construção de um *software* fez-se necessária a seleção de um processo de *software* adequado, o qual agregasse valor ao desenvolvimento. Devido à limitação de apenas uma pessoa envolvida no projeto, a gama de processos de *software* disponíveis se reduziu drasticamente, pois a maioria deles é focada no trabalho de equipes e, embora possam ser realizadas adaptações para ajustar estes processos, estas são extremamente passíveis de erros e podem distorcer diretrizes básicas do processo, tornando-o mais um empecilho no desenvolvimento que uma ferramenta de auxílio. Deste modo, foi selecionado o processo *Personal Extreme Programming (PXP)*, um processo ágil e iterativo que parte do pressuposto de que apenas uma pessoa estará diretamente envolvida no desenvolvimento do projeto (DZHUROV. KRASTEVA. ILIEVA. 2009).

O *PXP* é derivado dos processos *Personal Software Process (PSP)* e *Xtreme Programming (XP)* e suas diretrizes seguem uma combinação dos preceitos de ambos, propondo ser uma versão mais leve do *PSP* e introduzindo práticas do *XP* apropriadas ao uso por apenas uma pessoa.

As fases do processo, em ordem de execução, são:

1. Levantamento de requisitos;
2. Planejamento;
3. *Design*;
4. Implementação, a qual é subdividida em teste unitário, codificação e refatoração;
5. Teste do sistema; e,
6. Retrospectiva.

As iterações ocorrem a partir da fase de *design*, contudo como se trata de um processo ágil, as fases de levantamento de requisitos e planejamento também aceitam mudanças em etapas avançadas do projeto. Na Figura 3 pode ser visualizada uma ilustração das fases do *PXP*:



**Figura 3 – Fases do PXP**

Fonte: DZHUROV, Yani. KRASTEVA, Iva. ILIEVA, Sylvia. 2009.

## 4.2 CRONOGRAMA

Na Tabela 3 está presente o cronograma do projeto, o qual foi definido de acordo com o prazo delimitado e o processo selecionado (*PXP*):

**Tabela 3 - Cronograma**

	Setembro	Outubro	Novembro	Dezembro	Janeiro
Levantamento de Requisitos	∞				
Planejamento	∞				
Consulta bibliográfica	∞	∞	∞	∞	∞
Escrita da monografia					∞
1ª iteração		∞			



2ª iteração			∞	∞	
3ª iteração				∞	
4ª iteração					∞

- 1ª Iteração: manter fluxograma, manter elementos do fluxograma;
- 2ª Iteração: executar fluxograma, executar fluxograma passo a passo (“debugar”);
- 3ª Iteração: realizar cadastro, *login* e *logout*;
- 4ª Iteração: retornar estado de elemento alterado no fluxograma (“ctrl+z”) e implementação de *checks* e *triggers* de validação no *SGBD*.

### 4.3 PRÁTICAS UTILIZADAS

No PXP, o processo de software selecionado, são definidas quatorze práticas de organização e desenvolvimento. Neste projeto foram selecionadas e utilizadas onze dessas práticas, sendo deixadas de lado aquelas que poderiam por em risco o desenvolvimento da ferramenta proposta dentro do prazo delimitado. Além disso, algumas práticas sofreram pequenas alterações, as quais foram realizadas visando seu melhor aproveitamento.

A seguir são apresentadas as práticas selecionadas, juntamente com os artefatos por estas gerados e também detalhes da utilização e a contribuição de cada uma das delas ao projeto.

#### 4.3.1 Integração Contínua

Prática comum em metodologias ágeis, a integração contínua se baseia na ideia de automatizar todo o trabalho possível dentro de um projeto de software, evitando assim perda de tempo e desgaste de recursos humanos com tarefas repetitivas.

Neste projeto, essa prática foi implementada com a utilização da ferramenta Jenkins. Na utilização desta ferramenta, inicialmente foi criado um *job* (tarefa ou

trabalho, termo utilizado pela ferramenta), sendo este configurado para verificar periodicamente o repositório do projeto em busca de alterações. Assim que uma alteração é detectada (logo após o envio de dados ao repositório) a execução completa do *job* é disparada, sendo que essa por sua vez dispara a execução da ferramenta Apache Ant, a qual por meio da leitura de um XML de configuração dispara a execução dos seguintes programas: *phploc*, *phpcs*, Codeception, *phpcpd*, Plato e *phpdox*.

Cada um dos programas executados pela ferramenta Jenkins (indiretamente, por meio do Apache Ant) foi configurado para gerar relatórios em formatos variados (*XML* e *HTML* em sua maioria). Após o fim da execução de todos os programas estes relatórios são processados e publicados na página inicial do *job*. Na Figura 4 é demonstrada a página inicial do *job*, nela podem ser visualizados *links* que direcionam para os relatórios publicados, bem como gráficos que demonstram a tendência da quantidade de arquivos contendo código duplicado e quantidade de linhas de código fora do padrão encontradas no projeto durante o tempo.

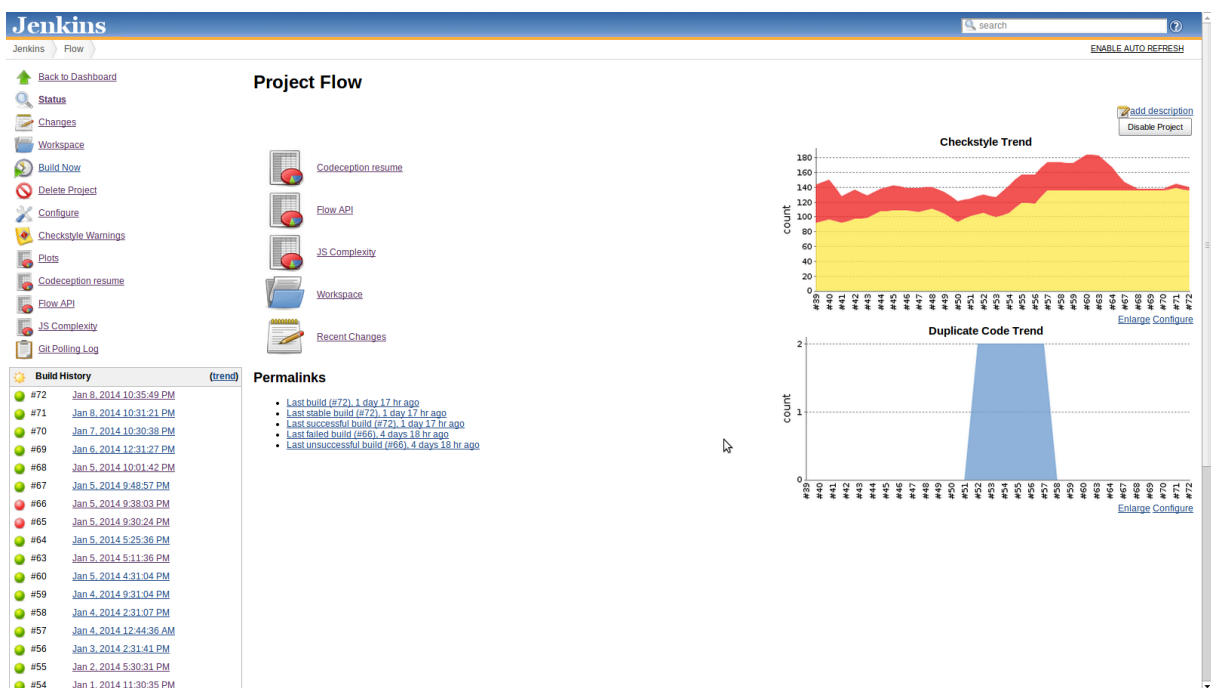


Figura 4 - Ferramenta Jenkins

Um dos relatórios publicados pela ferramenta Jenkins é a documentação do código *PHP* do projeto. Esta documentação lista todas as classes, interfaces e *namespaces* presentes no código, além de permitir a visualização de classes ou

interfaces específicas, exibindo assim os atributos, assinaturas dos métodos e comentários destas.

A ferramenta responsável pela geração da documentação denomina-se *phpDox* e na Figura 5 pode-se visualizar a página inicial da documentação gerada para o projeto Flow.

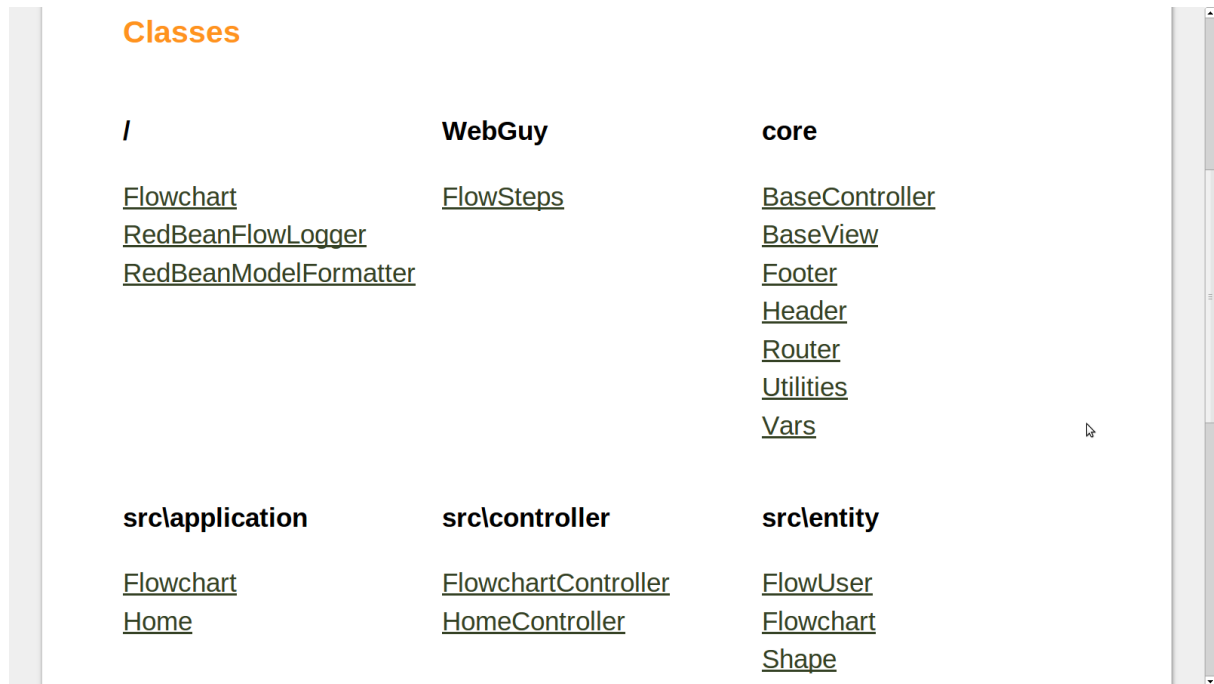


Figura 5 - Documentação gerada pelo phpDox

#### 4.3.2. Revisão de Código

A prática consiste em, após a codificação, uma revisão manual do código de modo que se possa realizar uma triagem básica à procura de erros.

Durante todo o desenvolvimento deste projeto a prática foi utilizada com sucesso, poupando tempo e evitando erros futuros com deslizes recorrentes, tais como: a ocorrência de *typo* (erro de digitação) em variáveis e *strings* e erros simples de lógica.

#### 4.3.3. Refatoração e Otimização

O processo de refatoração ocorre após a codificação. Ele institui que quaisquer melhorias possíveis referentes a modularização, *design*, legibilidade, simplicidade e cortes de duplicações devem ser aplicadas ao código. Nesta prática

é também onde todos os conhecimentos sobre bom *design* de *software* podem ser aplicados. Podendo-se aumentar a coesão, diminuir o acoplamento, separar conceitos, garantir a expressividade do código, e minimizar o número de classes e métodos (MARTIN 2008).

O processo de otimização consiste em modificar o código de modo a que este seja executado mais eficientemente ou consuma menos recursos.

Embora refatoração e otimização sejam duas práticas distintas e por vezes conflitantes, julgou-se que dado os recursos disponíveis e o tempo delimitado o melhor resultado seria alcançado aplicando-se as duas simultaneamente, tentando-se achar um ponto de equilíbrio entre ambas.

As práticas foram utilizadas continuamente neste trabalho, e constituíram-se nas mais custosas do projeto, em termos de tempo e esforço dispendidos. Tal consistência no uso gerou resultados, pois o código final se tornou mais coeso, legível, expressivo e com um desempenho melhor que aquele inicialmente escrito.

Como exemplo de uma atividade de refatoração e otimização relevante ocorrida no projeto, podemos demonstrar o código JavaScript presente na Figura 6, o qual é referente a função *loadConnections*, responsável pela criação das conexões dos elementos de um fluxograma durante o carregamento deste.

```

for (var index in retval["shapeconnections"]) {
    var shapeConnData = retval["shapeconnections"][index];

    var $source = $("div.shape[data-flow-id='" + shapeConnData['fk_source_shape'] +
        "'"]);
    var $target = $("div.shape[data-flow-id='" + shapeConnData['fk_target_shape'] +
        "'"]);

    var conn = jsPlumb.connect({source: $source, target: $target});

    if (shapeConnData.value !== "") {
        var labelId = $source[0].id + "_" + $target[0].id;
        var label = "<p id='" + labelId + "' class='connector-text' >" +
            shapeConnData.value + "</p>";

        conn.setLabel(label);

        jsPlumb.draggable(labelId, {
            containment: $(document.getElementById("flowchart-area")).
                find(document.getElementsByClassName("flowchart active"))
        });
    }
    that.listenDbClickOnConn(conn);
}

```

**Figura 6 – Código inicial da função *loadConnections***

Alguns pontos que foram levantados quanto ao código presente na Figura 6:

1. A existência de duplicação de código na atribuição de valores para as variáveis *\$source* e *\$target*;
2. O excessivo tamanho da função;
3. O laço do código é ineficiente, visto que a cada iteração a propriedade “*shapeconnections*” do objeto *retval* é buscada duas vezes, uma na resolução do laço e outra na atribuição da variável *shapeConnData*. Também, embora isso não possa ser inferido apenas pela figura, a propriedade “*shapeconnections*” contém um *array* numerado (0...N), logo a iteração poderia ser realizada utilizando-se o laço *for* tradicional em lugar do laço *for...in*, o qual possui um custo computacional maior;
4. A chamada da função *jsPlumb.draggable* está recebendo como segundo parâmetro um objeto que contém a chave *containment*, cujo valor passado é o fluxograma ativo, o qual é buscado através do código “*\$(document.getElementById(“flowchart-area”)).find(docume... (“flowchart active”))*”. Como o fluxograma ativo permanece inalterado durante a criação das conexões, essa repetida busca é ineficiente e pode ser facilmente substituída;
5. O método *listenDbfClickOnConn* define, internamente, a adição de um evento de clique duplo a cada conexão recém-criada. Esta adição de eventos a vários elementos similares em JavaScript é ineficiente e pode ser otimizada.

Em vista destes pontos a função *loadConnections* foi refatorada, podendo o novo código ser visualizado na Figura 7.

```

var connections = retval["shapeconnections"];
var $flowchart = $("#flowchart-area").find(".flowchart.active");
for (var i = connections.length; i--;) {
    var shapeConnData = connections[i];

    var $source = this.findShapeById(shapeConnData['fk_source_shape']);
    var $target = this.findShapeById(shapeConnData['fk_target_shape']);

    var connection = jsPlumb.connect({source: $source, target: $target});

    if (shapeConnData.value !== "") {
        this.setConnectionLabel(connection, shapeConnData.value, $flowchart);
    }
}

```

**Figura 7 – Código da função *loadConnections* refatorada**

O código refatorado da função *loadConnections* se tornou menor, mais coeso, expressivo e possui também um melhor desempenho. As alterações feitas foram:

1. O laço *for...in* foi substituído por uma laço *for* otimizado, sendo também criada a variável *connections* com o intuito de evitar a repetida busca do valor presente na propriedade *shapeconnections*;
2. O trecho de busca dos elementos para atribuição das variáveis *\$source* e *\$target* foi movido para o método *findShapeById*;
3. O trecho de código interno ao condicional *if* foi movido para o método *setConnectionLabel*, sendo um dos parâmetros passados a este o fluxograma ativo, o qual se encontra na variável *\$flowchart*, esta sendo definida apenas uma vez, fora do laço;
4. Os métodos nativos de JavaScript *getElementById* e *getElementByClassName* foram substituídos pelos métodos de busca da biblioteca jQuery, pois estes apresentam um nível de legibilidade maior;
5. O método *listenDbClickOnConn* foi excluído. Em seu lugar foi criado um novo método, o qual é invocado apenas uma vez durante o carregamento da página de fluxogramas. Este novo método possui as mesmas funcionalidades do método *listenDbClickOnConn*, com a diferença de atribuir o evento de clique duplo a

quaisquer conexões, independente de se essas forem criadas antes ou após a invocação do método. O código do método não será exibido para fins de simplicidade.

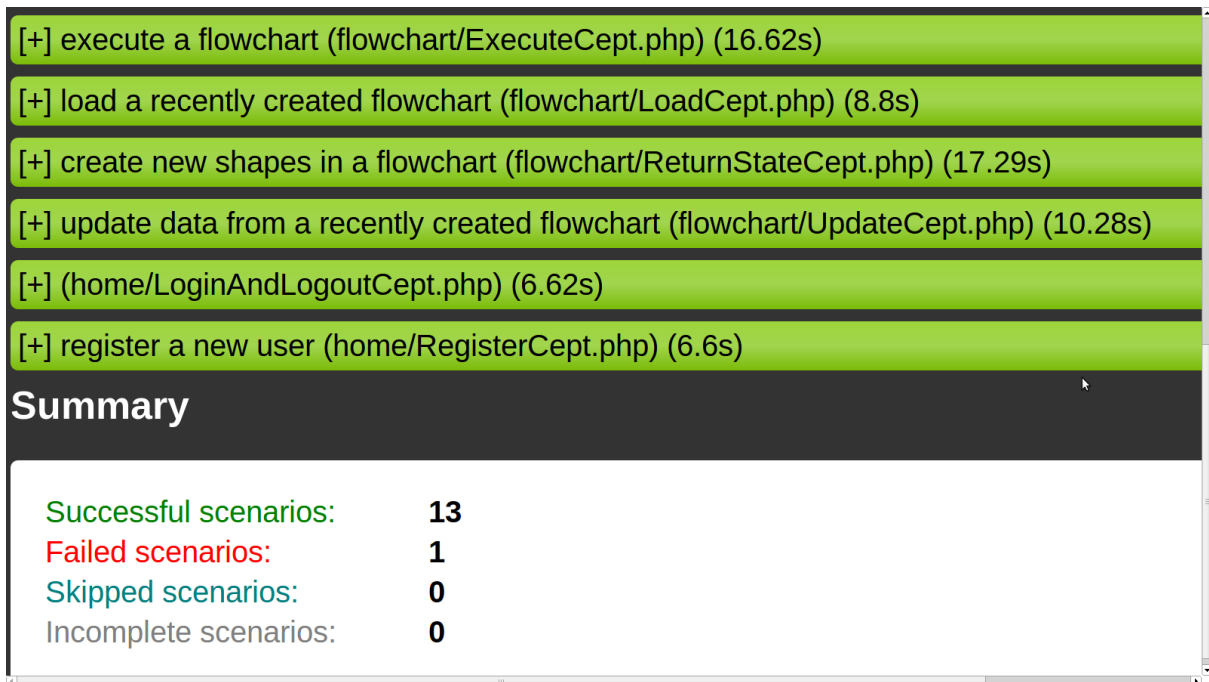
A implementação do método que substituiu o excluído *listenDbClickOnConn* foi fundamentada na técnica JavaScript conhecida como *delegation* (delegação). A técnica de delegação consiste em, ao invés de se atribuir o mesmo evento diretamente a diversos nós (*tags HTML*) similares, atribui-se este evento para um dos nós “pais” (superiores na *DOM*) e dentro do *callback* deste evento é realizado um teste condicional para verificar se, quando o evento for disparado, o nó que disparou o evento é aquele desejado. Tal técnica possibilita uma melhor estruturação de código, além de um ganho considerável de desempenho.

#### 4.3.4. Desenvolvimento Orientado a Testes

Uma constante em processos ágeis, o TDD (*Test-Driven Development*) também está presente no processo *PXP*. Seu objetivo é garantir que a maior parte possível das funcionalidades possuam testes automatizados, anteriormente ao próprio desenvolvimento de tais funcionalidades, e que o processo de criação destes testes guie um melhor *design* de código e interface.

Tradicionalmente o *TDD* é baseado em testes unitários, contudo neste projeto foi realizada uma pequena modificação e adotada a prática do *ATDD* (*Acceptance Test-Driven Development*), a qual tem um enfoque maior em testes baseados no ponto de vista do usuário. Os testes da prática *ATDD* são chamados *Acceptance Tests* (testes de aceitação) e podem ser vistos basicamente como as ações tomadas por um usuário descritas em código, de modo que essas possam ser realizadas a qualquer momento em um *browser*. Tal desvio ocorreu principalmente em razão do quesito tempo, pois, embora os testes unitários sejam geralmente vistos como mais importantes e benéficos e os testes de aceitação não sejam substitutos para eles, a manutenção da prática de *TDD* exigiria um esforço muito maior devido a frequente alteração do código criado em função do processo constante de refatoração, tornando assim inviável sua utilização efetiva no curto espaço de tempo e na quantidade de trabalho envolvida neste projeto. Contudo, os testes de aceitação também possuem seu valor e auxiliaram em muito o processo de desenvolvimento, detectando erros no funcionamento da ferramenta, na interface e muitas vezes guiando o *design* desta.

Para criação dos testes de aceitação, foi utilizado o programa/*framework* de testes Codeception, o qual permite descrever as ações tomadas pelo usuário em quaisquer páginas da aplicação (através da linguagem *PHP*) e depois executa-las em diferentes *browsers*. Na Figura 8 é exibida parte do relatório gerado pela ferramenta Codeception após a execução de todos os testes. Neste relatório são exibidos os testes executados e seu resultado (sucesso, falha, pulado ou ignorado).



[+] execute a flowchart (flowchart/ExecuteCept.php) (16.62s)
[+] load a recently created flowchart (flowchart/LoadCept.php) (8.8s)
[+] create new shapes in a flowchart (flowchart/ReturnStateCept.php) (17.29s)
[+] update data from a recently created flowchart (flowchart/UpdateCept.php) (10.28s)
[+] (home/LoginAndLogoutCept.php) (6.62s)
[+] register a new user (home/RegisterCept.php) (6.6s)
<b>Summary</b>
Successful scenarios: <b>13</b>
Failed scenarios: <b>1</b>
Skipped scenarios: <b>0</b>
Incomplete scenarios: <b>0</b>

**Figura 8 – Fragmento de relatório do programa Codeception**

#### 4.3.5. Pequenas Versões

A cada iteração do processo foi lançada uma versão funcional do sistema contendo partes específicas dos requisitos. Assim na primeira iteração o módulo de *CRUD* do fluxograma e de seus elementos estava disponível. Na segunda iteração a execução do fluxograma estava disponível, e assim por diante, seguindo o calendário presente na Tabela 3.

Embora esta prática tenha um enfoque maior no sentido comercial, este sendo a possibilidade de entregar um sistema ou ferramenta semi-funcional em curtos períodos de tempo ao cliente, ela ainda assim apresentou benefícios a este projeto, pois a existência constante de uma implementação parcial da ferramenta



propiciou a contínua checagem de usabilidade da ferramenta, gerando uma melhora gradativa e significativa neste aspecto.

#### 4.3.6. Padronização de Código

No início do projeto foi definido um padrão de código a ser seguido durante todo o desenvolvimento, a fim de uniformizar o código e melhorar sua legibilidade.

Para a aplicação do padrão de código neste trabalho, foi utilizado o programa *phpcs*, o qual realiza a validação dos arquivos de código *PHP* e *JavaScript* por meio de um arquivo *XML* que contém a especificação do padrão. Além de validar o padrão do código, o programa *phpcs* gera relatórios e gráficos quanto aos arquivos que contêm trechos fora do padrão. Um dos relatórios gerado pelo *phpcs* pode ser visualizado na Figura 9, a qual demonstra dois erros de alta prioridade encontrados em arquivos da pasta *controller*.

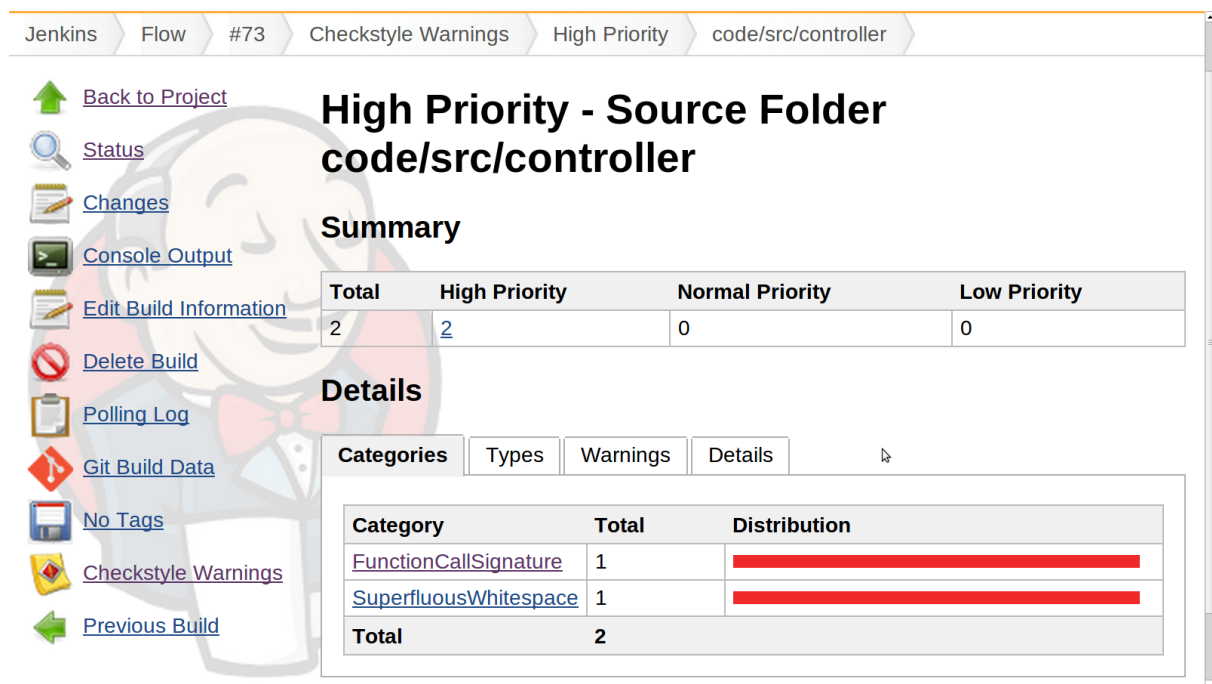


Figura 9 – Relatório gerado pelo programa *phpcs*

#### 4.3.7. Mensuração de Tamanho de código

A prática de mensuração de tamanho de código institui que deve ser definida e aplicada uma forma de se mensurar a quantidade de código produzido ao longo do projeto.

Adicionalmente a mensuração de tamanho, foi utilizada, apenas para o código JavaScript (devido a sua maior importância no projeto), a métrica de nível de manutenibilidade do código. O nível de manutenibilidade é determinado de acordo com o cálculo de diversas métricas de código, através deste cálculo é definido um valor de 0 a 100, o qual indica a dificuldade de manutenção existente no código. Basicamente quando maior o valor, melhor é o nível de manutenibilidade. Embora esta métrica não seja totalmente acurada, ela proveu um auxílio constante no esforço de se manter uma boa qualidade do código.

Para a mensuração de tamanho do código produzido em *PHP*, foi utilizada a ferramenta *phploc*, através da qual foram obtidos os seguintes dados quantitativos (referentes a última mensuração realizada no trabalho): 33 arquivos, 2020 linhas de código, 24 classes, 7 *namespaces*, 0 interfaces, 3 classes abstratas, 21 classes concretas, 119 métodos, 92 métodos não estáticos, 27 métodos estáticos, 75 métodos públicos e 27 métodos não públicos.

A mensuração de tamanho e do nível de manutenibilidade do código JavaScript foi realizada por meio da ferramenta Plato. Na Figura 10 é exibido um dos relatórios gerados por essa ferramenta, onde pode ser visualizado o total de linhas de código (2546), a média de linhas por arquivo (146), a média de manutenibilidade (72.13, um valor considerado alto) e também gráficos exibindo a tendência de linhas de código e a média de manutenibilidade em função do tempo.

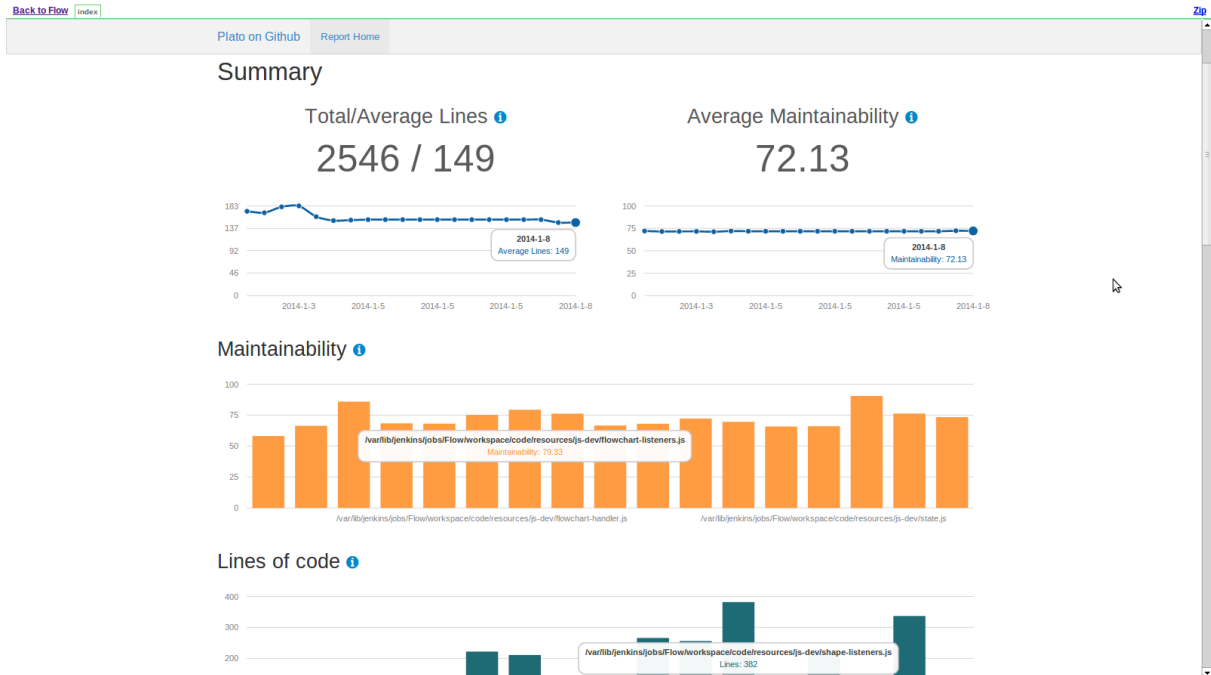


Figura 10 - Relatório do programa Plato

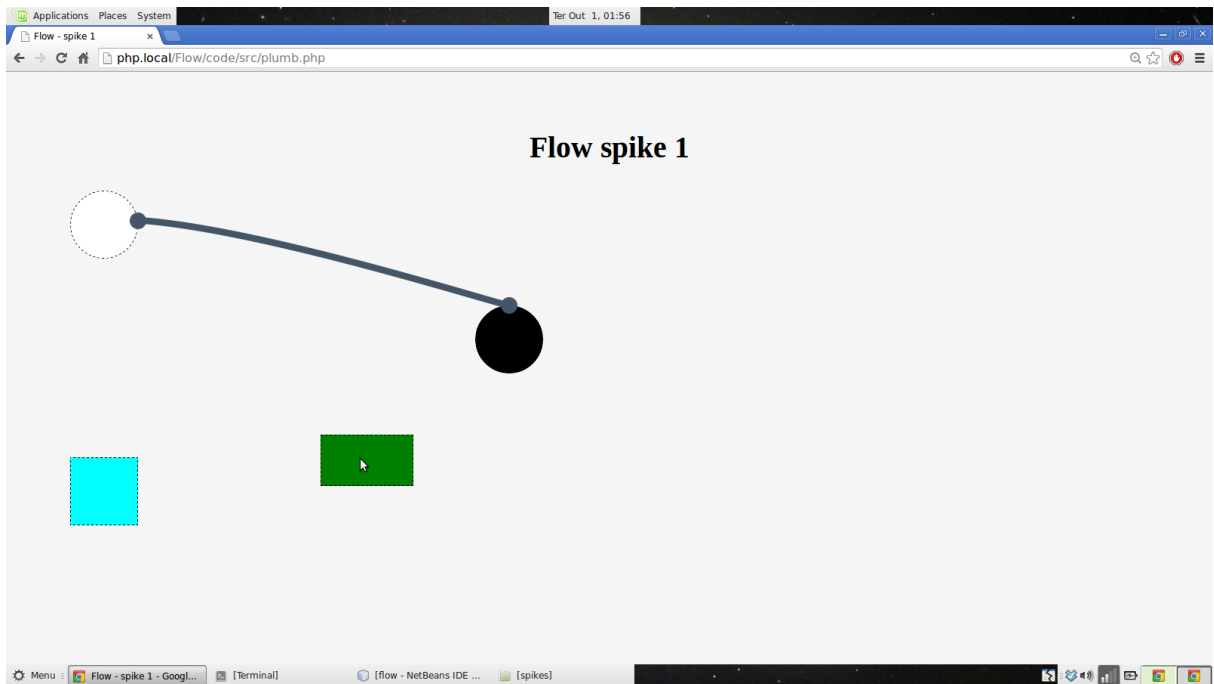
#### 4.3.8. Registro de Defeitos

Defeitos ou *bugs* relevantes encontrados durante o desenvolvimento foram documentados na ferramenta ClockingIT. Tal prática teve o objetivo de forçar o reconhecimento dos erros mais frequentemente cometidos durante a codificação e evitar sua repetição futura.

#### 4.3.9. Spike Solutions

*Spike Solution* é uma prática simples cujo intuito é verificar potenciais soluções para um determinado problema.

A utilização da prática consistiu em, previamente ao desenvolvimento de alguma funcionalidade crucial ao projeto, criar-se um *spike*. O resultado de um dos *spikes* criados pode ser visualizado na Figura 11, tendo sido o propósito específico deste, verificar a viabilidade da utilização da biblioteca jsPlumb para a realização da conexão dos elementos de um fluxograma.



**Figura 11- Spike**

#### 4.3.10. Planejamento de Atividade

No início e durante as iterações foram definidas atividades a serem realizadas durante a iteração. O objetivo do uso desta prática foi obter-se uma melhor produtividade e estimativas do andamento do projeto. Para o controle destas atividades foi utilizada a ferramenta *web* ClockingIT. Algumas das atividades definidas durante o trabalho podem ser visualizadas na Figura 12.

18:34 #59 Definir tarefas para melhoria das funcionalidades da segunda iteração  
Flow TD / 2ª Iteração / [Bruno Roberto]  
- Status: Open -> Closed

14:25 #72 Ajustar layout de acordo com novo protótipo  
Flow TD / 2ª Iteração / [Bruno Roberto]

Monday, 02 December 2013

17:09 #71 Proposta de melhoria de processo da 2 iteração (referente ao fluxo não alterad  
Flow TD / 2ª Iteração / [Bruno Roberto]

17:08 #70 Spike para execução passo a passo  
Flow TD / 2ª Iteração / [Bruno Roberto]

17:03 #69 Obrigar a existência de dois fluxos para elementos condicionais e de laço  
Flow TD / 2ª Iteração / [Bruno Roberto]

16:56 #68 Início da iteração adiado para dia 16  
Flow TD / 3ª Iteração / [Bruno Roberto]

16:53 #67 Nova versão do protótipo da tela de execução do fluxograma  
Flow TD / 2ª Iteração / [Bruno Roberto]

16:47 #66 Teste para deleção de conexões (seguindo novo modelo)  
Flow TD / 2ª Iteração / [Bruno Roberto]

16:47 #64 Alteração da deleção de conectores  
Flow TD / 2ª Iteração / [Bruno Roberto]  
- Due: Thursday, 05 December 2013 -> Sunday, 08 December 2013

Clocking IT v0.99.3  
Feedback? Suggestions? Ideas? Bugs? Online (1)

Figura 12 - ClockingIT, atividades

#### 4.3.11. Proposta de Melhoramento de Processo

No fim de cada iteração foi escrita, de modo informal e quando se julgou necessário, uma pequena proposta contendo sugestões sobre os pontos que poderiam ser melhorados na utilização do *PXP*. Tais considerações foram feitas levando-se em conta a experiência recém-adquirida no uso do processo, e com a finalidade de promover a constante evolução no uso deste. As propostas sugeridas foram:

- 1º Iteração: “O desenvolvimento de testes na primeira iteração, previamente a codificação, mostrou-se penoso e ineficaz. Muitos detalhes referentes à implementação só puderam ser descobertos durante esta, logo é razoável mencionar que a aplicação do Desenvolvimento Orientado a Testes deve ser por vezes “afrouxada”, principalmente em relação a projeto em que detalhes da implementação são difíceis de planejar com antecedência”.
- 2º Iteração: “Não foi possível desenvolver um *Spike* nesta iteração, pois o mesmo não se mostrou necessário. Sendo um artefato menor e que visa auxiliar o trabalho de desenvolvimento sua obrigatoriedade não deve existir”.
- 3º Iteração e 4º Iteração: Nenhuma melhoria sugerida.

## 4.4 DESENVOLVIMENTO DA FERRAMENTA

### 4.4.1 Levantamento de Requisitos

Seguindo o modelo do processo selecionado (*PXP*), o desenvolvimento começou com a etapa de levantamento de requisitos, na qual foi produzido o documento de requisitos de *software* (apêndice A). Devido ao excessivo detalhamento que este documento apresenta, foi criado também um diagrama de casos de uso, o qual pode ser visualizado na Figura 13. Este diagrama de casos de uso possibilita uma visão mais geral e abstrata das funcionalidades da ferramenta Flow, tornando-o adequado para fins de consultas rápidas ou demonstrações das funcionalidades da ferramenta a pessoas não familiarizadas com esta.

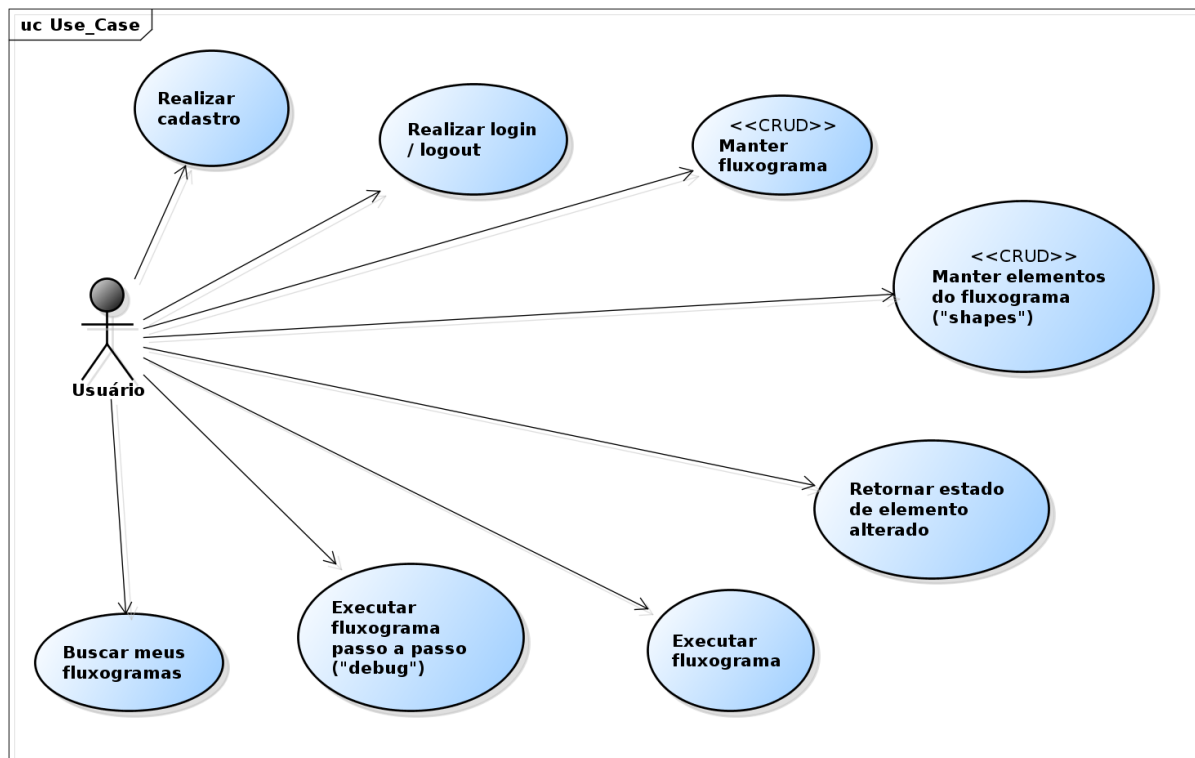


Figura 13 - Diagrama de casos de uso

### 4.4.2 Planejamento

A primeira ação na etapa de planejamento foi a configuração da ferramenta Jenkins e das ferramentas por ela automatizadas (maiores detalhes podem ser encontrado no capítulo 4.3.1 Integração Contínua). Embora estas configurações

tenham tomado uma quantidade considerável de tempo, a automatização provida pela ferramenta Jenkins foi crucial para o bom andamento do projeto.

Após as configurações da ferramenta de integração contínua, foi criado um diagrama de entidade relacionamento do banco de dados da aplicação. A modelagem deste diagrama acabou por revelar-se dispensável, em vista da pequena quantidade de entidades levantadas para a ferramenta Flow, assim este diagrama foi descartado, focando-se a modelagem do banco de dados exclusivamente no diagrama físico.

O diagrama físico do banco de dados da ferramenta Flow pode ser visualizado na **Erro! Autoreferência de indicador não válida..** Nele são demonstradas as tabelas do banco de dados da aplicação.

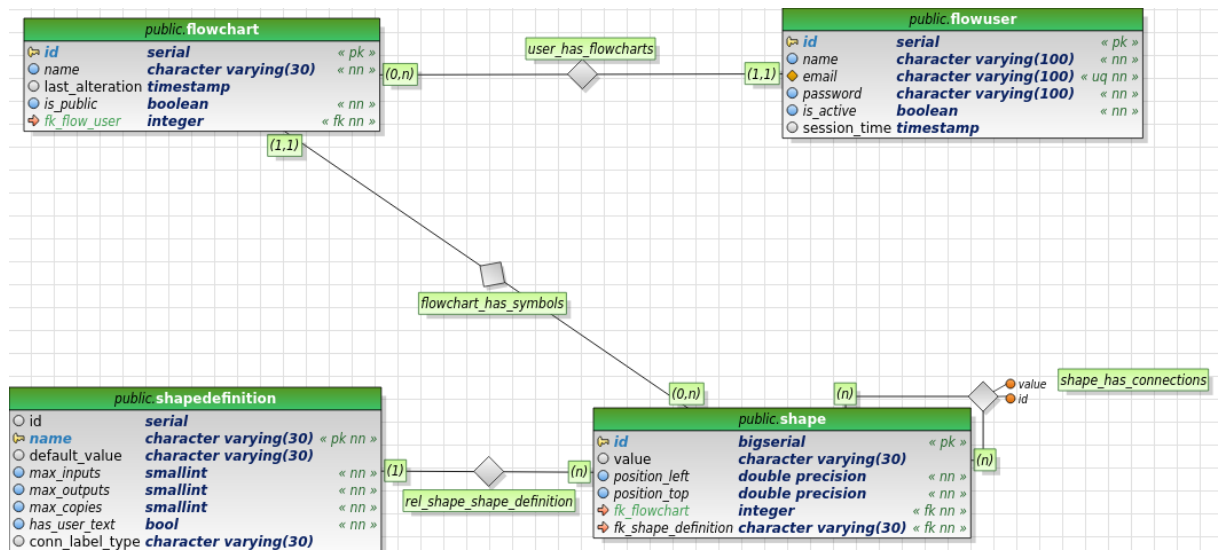


Figura 14 – Diagrama físico do banco de dados

As tabelas exibidas na **Erro! Autoreferência de indicador não válida.** são: *flowuser*, representa o usuário da aplicação; *flowchart*, representa os *N* fluxogramas que um determinado usuário pode possuir; *shape*, representa os *N* elementos que um determinado fluxograma pode possuir; *shapeconnection*, representa as conexões que os elementos (*shapes*) do fluxograma podem possuir entre si; *shapedefinition*, representa os diferentes tipos de elementos (*shapes*) existentes e também determinadas características sobre cada elemento, tais como número máximo de conexões, número máximo de cópias possíveis do elemento, valor padrão, se o elemento pode ou não possuir texto interno, entre outras.

#### 4.4.3 Desenvolvimento de um *Framework*

Previamente ao início da primeira iteração, nenhuma estrutura de código havia sido definida, assim fazendo-se necessário a criação de uma. Com os objetivos de se prover esta estrutura, uma boa organização e promover a reusabilidade de código, foi desenvolvida uma *framework* para a codificação *back-end* (realizada na linguagem *PHP*). A estrutura desta *framework* pode ser visualizada por meio do diagrama de classes exibido na Figura 15.

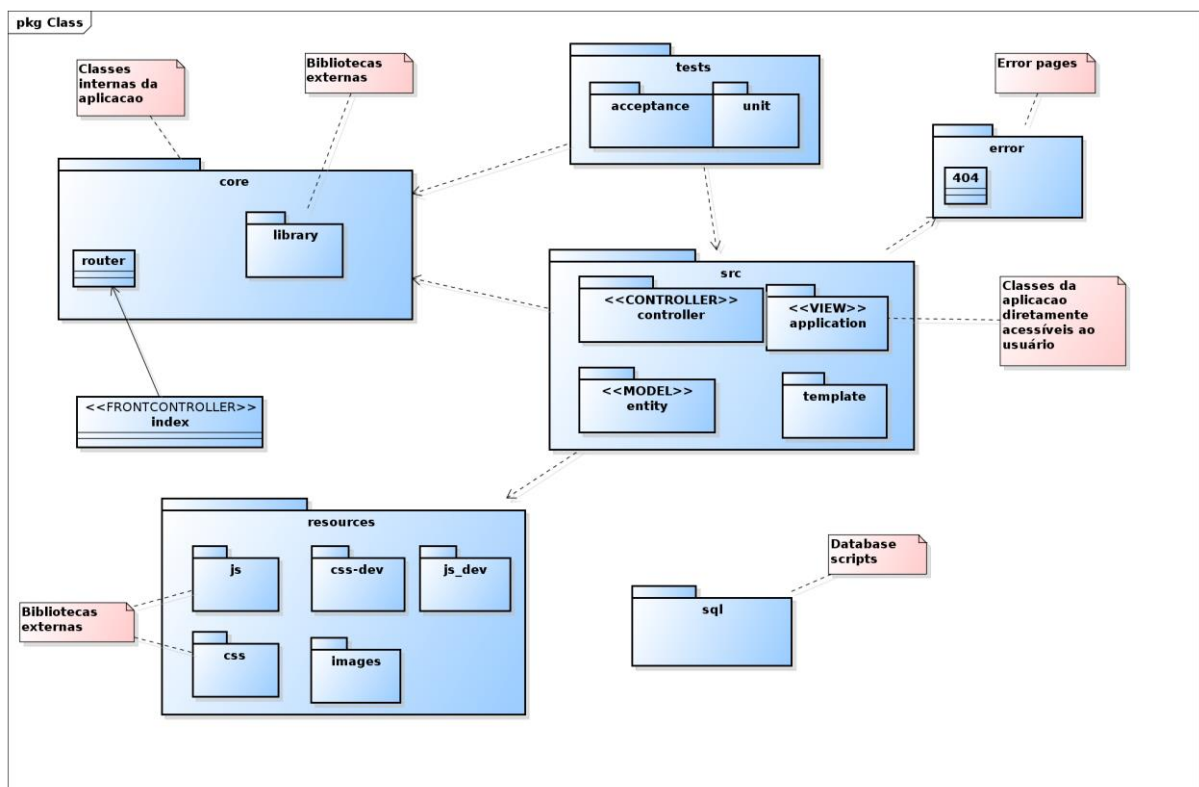


Figura 15 - Diagrama de classes da *framework* desenvolvida

Essa *framework* desenvolvida é baseada no *design pattern Front Controller*, sendo este implementado da seguinte forma: toda requisição feita ao servidor é redirecionada ao *script index.php*, no qual estão contidas as dependências e operações comuns a todos os *scripts PHP*, facilitando assim a definição de configurações globais da aplicação e prevenindo duplicações de código. Ela também utiliza o modelo *MVC*, este estruturado do seguinte modo: os *scripts* dentro da pasta *application* constituem a *view* e são os únicos diretamente acessíveis ao usuário, sua função é invocar o *controller* e o método apropriado deste, receber os dados



retornados e renderizá-los ao usuário; os *controllers* estão localizados na pasta *controller* e sua função é interagir com o banco de dados, manipular e retornar entidades para a camada da *view*; as entidades representam o *model* e ficam inseridas na pasta *entity*, são objetos mapeados do banco de dados relacional e seu propósito é basicamente agir como estruturas de dados a serem manipuladas pelos *controllers*.

#### 4.4.4 Iterações

Essencialmente as iterações foram realizadas seguindo-se o cronograma, as fases do processo *PXP* e com a constante aplicação das práticas definidas. Contudo algumas iterações, bem como pontos cruciais no desenvolvimento delas, merecem ser delineados.

##### 4.4.4.1 Persistência dos elementos de um fluxograma

Tradicionalmente em aplicações *web*, alterações ou requisições de dados remetem ao recarregamento da página. Contudo, na ferramenta Flow a constante interação do usuário com os elementos do fluxograma geraria recarregamentos constantes da página, os quais degradariam, ou até mesmo inviabilizariam, os quesitos de bom nível de interação e usabilidade da ferramenta. Sendo assim, para contornar este problema, utilizou-se da tecnologia Ajax, a qual permite que, por meio de código JavaScript, sejam feitas requisições assíncronas ao servidor, evitando recarregamento da página e o bloqueio do uso da ferramenta para o usuário.

Mais detalhadamente, a persistência dos elementos foi implementada de modo que qualquer interação do usuário com os elementos de um fluxograma dispara um evento, sendo que este evento realiza uma requisição Ajax ao servidor, enviando os dados do elemento recém-alterado. Após os dados serem recebidos e validados pelo servidor, eles são persistidos no banco de dados e uma resposta de confirmação é enviada ao *browser* do cliente no formato *JSON*. Através desta implementação foi possível permitir que o usuário em nenhum momento precise salvar suas alterações no fluxograma e também que este não tenha sua interação com a ferramenta interrompida devido a ocorrência constante de requisições.

#### 4.4.4.2 Execução do fluxograma

Durante a segunda iteração ocorreu a implementação da execução do fluxograma. Para tal foi inicialmente adotada uma estratégia de baixo nível, na qual todo o grafo de elementos do fluxograma era percorrido através de um algoritmo recursivo e, durante esse percurso, era gerado o código para execução. Para a validação deste código foram utilizados um *lexer* e um *parser*.

Segundo MAK (2009), a função de um *lexer* é quebrar um código fonte nos menores pedaços possíveis e ler os caracteres sequencialmente, para que assim possa construir *tokens*, os quais são os elementos de baixo nível de uma linguagem (operadores aritméticos e palavras reservadas, por exemplo).

A função do *parser* é invocar continuamente o *lexer*, fazendo com que este retorne *tokens*, os quais são analisados pelo *parser* e transformados em elementos de alto nível da linguagem (uma estrutura de repetição, por exemplo). Cabe ao *parser* também verificar se os elementos de alto nível estão sintaticamente corretos (MAK, 2009).

Para a criação do *lexer* e do *parser* foi utilizada a ferramenta *Jison*, a qual permite a geração de ambos em código JavaScript. A geração do *lexer* pelo *Jison* se deu por meio da definição de uma série de expressões regulares que determinam como o código deve ser “quebrado” em pedaços que possam formar *tokens*. Para a geração do *parser* pelo *Jison* foi necessária a criação de uma gramática livre de contexto, escrita na norma *Backus-Naur Form*.

De forma geral, uma gramática livre de contexto é um conjunto finito de regras gramaticais, sendo a norma *Backus-Naur Form* uma das notações mais utilizadas para descrever tais gramáticas.

Embora a abordagem de baixo nível tenha possibilitado a execução completa do fluxograma satisfatoriamente, sua complexidade tornou a implementação da execução passo a passo inviável no tempo disponível. Dessa forma, optou-se pelo descarte desta abordagem em prol de uma nova, com um enfoque em alto nível.

Na nova, e corrente, abordagem utilizada na implementação da execução (tanto completa quanto passo a passo), cada elemento é validado e executado em separado, sendo o “caminho” a ser percorrido definido em tempo de execução. A validação ocorre logo antes da execução, sendo a função desta detectar erros na

sintaxe do pseudocódigo contido no elemento, ou detectar inconsistências no fluxo de ligação do elemento com outros. Tal feito é alcançado por meio de diversas técnicas, sendo a principal o uso de expressões regulares. Como exemplo temos a checagem de uma variável válida, dada pela expressão regular “/[a-zA-Z\_\$][0-9a-zA-Z\_\$]\*/g”. Essa abordagem de alto nível se mostrou extremamente satisfatória e possibilitou a finalização da execução passo a passo do fluxograma.

É importante salientar que, embora a abordagem de baixo nível para a execução do fluxograma tenha sido substituída, ela agregou valor ao projeto. Muitos dos detalhes na implementação da estratégia de alto nível foram frutos de conhecimento adquirido durante a implementação previa. Além disso a estratégia de baixo nível permanece como a possibilidade de um trabalho futuro, visto que sua implementação possibilita checagens de código mais rígidas.

#### 4.4.4.3 Retorno de estado

A implementação do retorno de estado dos elementos do fluxograma ocorreu na quarta iteração e foi a última funcionalidade adicionada à ferramenta. Para possibilitar sua implementação, foi criado o objeto *FlowchartState*, o qual encapsula todas as características e comportamentos referentes ao estado dos elementos do fluxograma. Este objeto possui métodos específicos para cada tipo de alteração realizada nos elementos de um fluxograma (criar, arrastar, alterar valor, deletar conexão, etc) e, a cada alteração ocorrida, são armazenados em uma pilha os dados que contêm o estado do elemento imediatamente antes de sua alteração, assim conforme alterações são realizadas os dados são empilhados. Quando o retorno de estado é solicitado os dados são retirados do topo da pilha e a última alteração dos elementos do fluxograma é desfeita.

Um detalhe importante é como ocorre a chamada dos métodos do objeto *FlowchartState*. Inicialmente esses métodos eram invocados nas funções de sucesso das chamadas Ajax, as quais ocorrem a cada alteração dos elementos do fluxograma. Contudo, na etapa de refatoração da quarta iteração, percebeu-se que esta implementação estava deixando o código excessivamente poluído e com um alto acoplamento. Assim, analisando-se este problema, foi encontrada uma solução que utiliza o *design pattern Observer*.

O padrão de projeto *Observer* (observador) permite a um objeto observar mudanças em outro objeto, sendo que são providos meios para que o objeto

observador seja notificado sempre que o objeto observado sofra alguma alteração (OSMANI, 2012).

Tendo sido encontrada a alternativa da utilização do padrão *Observer*, o código foi refatorado. Na nova implementação, antes da ocorrência de cada alteração em um elemento do fluxograma, é disparada uma notificação em *broadcast*, a qual faz com que o método observador, que fica continuamente em busca de tais notificações, seja invocado. Assim os dados do elemento, previamente a sua alteração, são empilhados para posterior retirada da pilha, a qual ocorre por meio de um evento que é invocado quando o usuário pressiona simultaneamente as teclas *ctrl* e *z*.

#### 4.4.4.4 Segurança

Em aplicações aplicações *web*, nenhum dado advindo de fontes externas é confiável, sempre deve-se assumir o cenário de pior caso, no qual os *Cookies*, requisições Ajax e valores de formulários enviados por *POST* estão corrompidos, incorretos ou contêm dados maliciosos, enviados com propósitos de danificar o servidor e o sistema (MACINTYRE, DANCHILLA, GOGALA, 2011).

Devido as relevantes possibilidades que uma aplicação em ambiente *web* possui de sofrer ataques ou receber dados maliciosos, o desenvolvimento da ferramenta Flow possui também um enfoque na questão de segurança.

Uma das principais formas de ataque a aplicações *web* se dá por meio de *SQL Injection*, técnica na qual o atacante injeta código *SQL* malicioso na aplicação, geralmente através campos de formulários. A prevenção desse tipo de ataque foi implementada através da filtragem de todos os dados recebidos na aplicação por requisições *GET* ou *POST*, sendo que esta filtragem elimina quaisquer caracteres enviados que possam ser usados como forma de injetar código malicioso no servidor da aplicação.

Outra questão, no contexto de segurança, é a possibilidade de que um usuário possa tentar visualizar ou alterar dados de outros usuários. Por exemplo, os dados de um fluxograma que possua a ID 4 podem ser obtidos por meio da seguinte requisição *GET*: <http://flow.com/flowchart/load/4>. Também, a deleção de um elemento de um fluxograma, sendo que este elemento possui a ID 40, pode ser realizada pela requisição *POST*: <http://flow.com/flowchart/deleteShape/40>. Deste modo, um usuário qualquer pode tentar visualizar fluxogramas de outros usuários,

ou até mesmo alterar dados referentes a estes fluxogramas. Assim, para evitar este tipo de ação, todas as operações que envolvem o banco de dados são precedidas de checagens que validam se o usuário que está fazendo a requisição está logado e se ele possui a permissão para realizar a operação requisitada.

A última validação de segurança abordada no projeto diz respeito a manipulação dos elementos de um fluxograma (criar, arrastar, alterar valor, etc). Cada elemento possui restrições quanto a que ações podem ser realizadas sobre ele. Estas restrições estão definidas na tabela *shapedefinition*, a qual pode ser visualizada na **Erro! Autoreferência de indicador não válida.**, e dizem respeito as seguintes características dos elementos: máximo de conexão de entrada aceitas pelo elemento (*max\_inputs*); máximo de conexões de saída possíveis para o elemento (*max\_outputs*); número máximo de elementos de cada tipo que podem existir em um fluxograma (*max\_copies*); e se o elemento pode ou não possuir texto interno (*has\_user\_text*).

As restrições contidas na tabela *shapedefinition* são aplicadas aos elementos do tipo correspondente diretamente em JavaScript, de modo que o usuário não possa, por exemplo, criar mais de um elemento do tipo “Inicio” em um determinado fluxograma, ou inserir mais de uma conexão de entrada em um elemento do tipo “Fim”. Contudo a validação realizada puramente em JavaScript é passiva de falhas, pois o código desta linguagem é executado diretamente no *browser* do usuário<sup>2</sup>, sendo desse modo possível que o usuário realize alterações no código que possam permitir a quebra das restrições inicialmente impostas. Para impedir inconsistências advindas deste tipo de situação, foram criadas uma séria de *triggers* no banco de dados. Estas *triggers* são executadas antes da criação ou atualização de elementos de um fluxograma, e verificam a validade dos dados recebidos, impedindo inconsistências entre os dados salvos e as restrições impostas na definição dos elementos.

Na Figura 16 encontra-se o código da *trigger* que valida se um determinado elemento do fluxograma criado ou alterado pode ou não possuir texto interno. Caso possa o elemento é criado ou alterado normalmente, caso não possa a operação é interrompida e invalidada. Esta *trigger*, bem com todas as outras foram codificadas na linguagem PL/pgSQL.

---

<sup>2</sup> No contexto deste trabalho, visto que JavaScript também é uma linguagem utilizada em servidores.

```

CREATE OR REPLACE FUNCTION validateShapeText() RETURNS TRIGGER AS $$
  DECLARE
    newValue CHARACTER VARYING(30) := NEW.value;
    shapeName CHARACTER VARYING(30) := NEW.fk_shape_definition;
    hasUserText BOOLEAN;
  BEGIN

    SELECT has_user_text INTO hasUserText FROM shapedefinition WHERE name =
    shapeName;

    IF (hasUserText = FALSE AND newValue IS NOT NULL AND newValue != '') THEN
      RAISE NOTICE 'Error on trigger validateShapeText. Shape ca\'t have
        text';
    RETURN NULL;
    END IF;

    RETURN NEW;
  END;
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS validateShapeText ON shape;
CREATE TRIGGER validateShapeText BEFORE INSERT OR UPDATE ON shape FOR EACH ROW
EXECUTE PROCEDURE validateShapeText();

```

**Figura 16 - Trigger de validação de elemento**

## 5. FERRAMENTA FLOW

### 5.1 VISÃO GERAL

Flow é uma ferramenta *web* que permite a criação de fluxogramas que possam ser executados similarmente a programas de computador.

Para a utilização da ferramenta o usuário deve inicialmente realizar um cadastro, o qual irá possibilitar seu *login* na aplicação. Uma vez que o usuário esteja logado será possível que ele crie, altere, liste e delete seus fluxogramas, bem como os elementos geométricos contidos nestes (presentes na Tabela 1).

Os elementos geométricos podem ser criados através da ação de arrastar e soltar, partindo do menu de elementos para o interior do fluxograma. Dentro do fluxograma eles podem ser livremente movimentados, sendo a área do fluxograma expandida automaticamente conforme o necessário. As operações possíveis nestes elementos são diversas, entre elas: arrastar livremente, alterar valores internos, deletar, criar conexões com outros elementos, atribuir valores a estas conexões, entre outras. Todas as operações realizadas em um elemento geométrico no interior de um fluxograma são automaticamente salvas no banco de dados. Estas alterações também podem ser desfeitas pelo usuário com o uso conjunto das teclas *ctrl* e *z*.

Após a criação de um fluxograma o usuário pode realizar a sua execução, a qual pode ser feita no modo completo ou passo a passo.

No modo de execução completa, todos os elementos do grafo do fluxograma são percorridos do início ao fim, ocorrendo pausas apenas quando um elemento de entrada manual de dados é encontrado.

O modo de execução passo a passo difere do completo no sentido de que cada execução é realizada apenas após a ação do usuário requisitando tal. A execução passo a passo pode ser interrompida a qualquer momento.

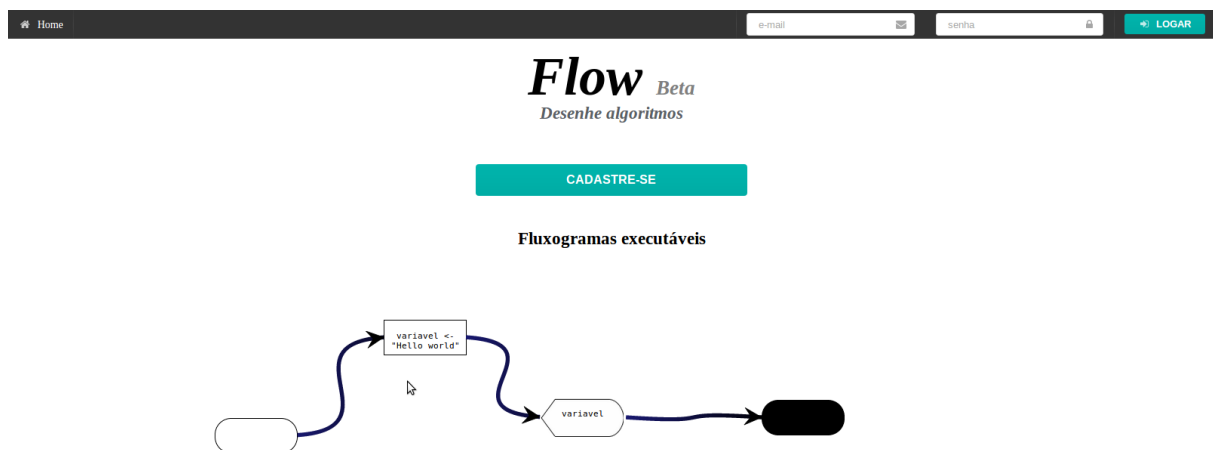
Embora os modos completo e passo a passo sejam visualmente diferentes, sua lógica de execução é essencialmente a mesma. Ambos são iniciados no elemento início, partindo dele para os demais elementos. O fluxo de execução é determinado pelas setas que conectam os elementos, sendo que todos os elementos, exceto o de decisão, possuem apenas um fluxo de saída. Quando um elemento de decisão é encontrado, seu pseudocódigo interno é avaliado, caso a expressão contida no pseudocódigo seja avaliada como verdadeira, o fluxo

“verdadeiro” é seguido, caso contrário o fluxo “falso” é seguido. Elementos de processamento realizam operações lógico-matemáticas ou a atribuição de valores a variáveis. Ao encontrar-se um elemento de entrada manual de dados é exibida uma caixa de entrada requisitando ao usuário a entrada de dados. Na execução de um elemento de saída de dados, o valor da variável ou o valor literal contido neste elemento é exibido ao usuário por meio do *console* da ferramenta. O símbolo “fim” denota o término do algoritmo representado pelo fluxograma.

## 5.2 TELAS

Previamente ao início do desenvolvimento das interfaces da ferramenta Flow foram desenvolvidos protótipos de tela, os quais, assim como a própria interface, sofreram diversas alterações durante o desenrolar do trabalho. No apêndice B podem ser encontradas as versões finais dos protótipos produzidos.

Na Figura 17 é exibida a tela inicial da ferramenta, sendo que nesta é possível que o usuário possa realizar o *login*, caso já possua o cadastro, ou clicar no botão “cadastre-se”, o qual irá levá-lo a tela de cadastro.

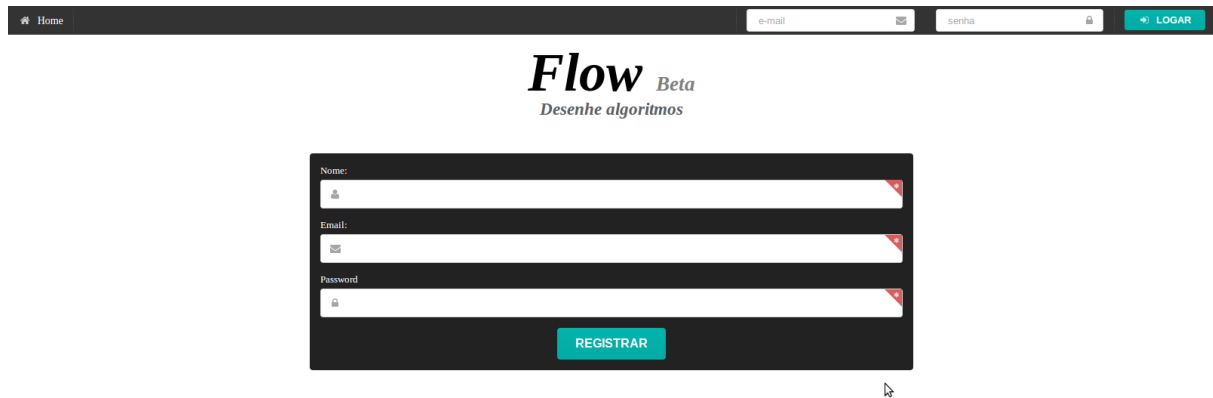


**Figura 17 – Flow, tela inicial**

A Figura 18 apresenta a tela de cadastro da ferramenta. Apenas o nome, e-mail e uma senha são requisitados para que o cadastro seja realizado. Após o



registro o usuário é automaticamente redirecionado para a tela inicial do fluxograma (exibida na Figura 19).



The image shows a web registration form for 'Flow Beta'. At the top, there is a dark navigation bar with a 'Home' link on the left, and search and login fields on the right. The search field is labeled 'e-mail' and the login field is labeled 'senha'. A 'LOGAR' button is located to the right of the login field. Below the navigation bar, the logo 'Flow Beta' is displayed in a large, stylized font, with the tagline 'Desenhe algoritmos' underneath. The main content area features a registration form with three input fields: 'Nome:' (Name), 'Email:', and 'Password'. Each field has a small icon to its left and a red 'X' in the top right corner, indicating a validation error. Below the password field is a teal 'REGISTRAR' button. A mouse cursor is visible at the bottom right of the form area.

**Figura 18 – Flow, Cadastro**

Na Figura 19 é exibida a tela principal da ferramenta, para a qual o usuário é redirecionado após o *login*. A área central é exibida sem nenhum elemento, visto que não há nenhum fluxograma selecionado. É possível observar a paginação lateral, pela qual são acessíveis todos os fluxogramas do usuário. Nesta tela o usuário também pode realizar a filtragem dos fluxogramas paginados, por meio do campo de busca localizado acima da paginação, ou ainda realizar o *logout* por meio do botão “SAIR”.

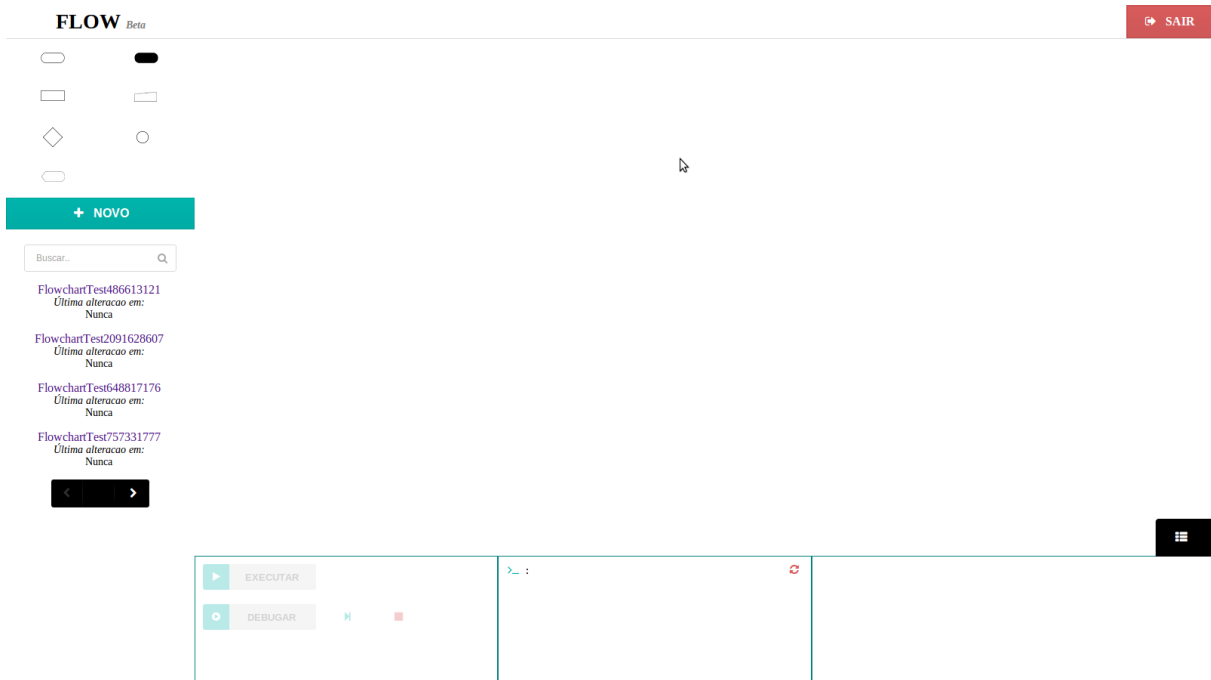


Figura 19 – Flow, tela pós-login

Na Figura 20 é novamente exibida a tela principal da ferramenta. Nesta figura pode-se notar que existem diversas abas abertas, as quais representam fluxogramas selecionados pelo usuário. Estas abas podem ser clicadas para que o fluxograma a qual correspondam seja exibido. O ícone “x” das abas também pode ser clicado, o que irá fechar determinada aba, bem como o fluxograma a que esta corresponde.

É possível notar também que na Figura 20 a área que contém os botões de execução não está visível, tornando maior a área disponível do fluxograma.

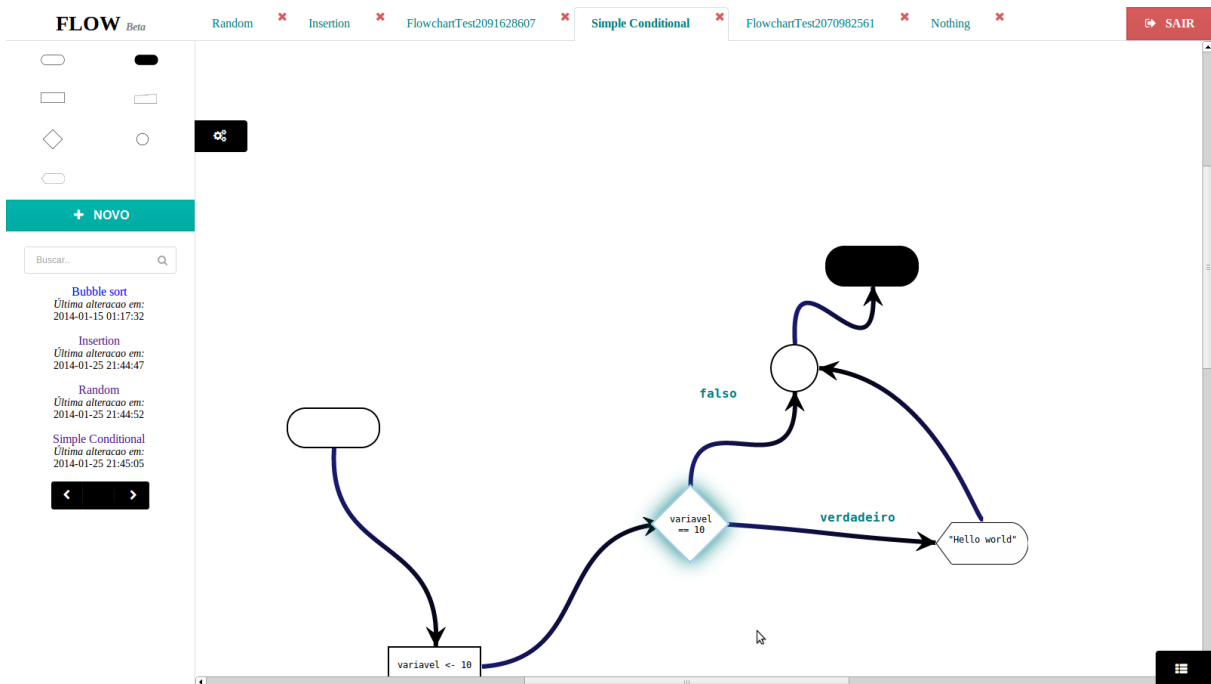


Figura 20 – Flow, área do fluxograma expandida

Na Figura 21 é exibido o *menu* lateral, o qual permite a alteração do nome de um fluxograma ou a sua deleção; este menu lateral é exibido após o clique no botão que possui ícones de engrenagens. Pode-se notar também que é visível uma caixa de diálogo pedindo a confirmação do usuário para que o fluxograma corrente possa ser deletado.

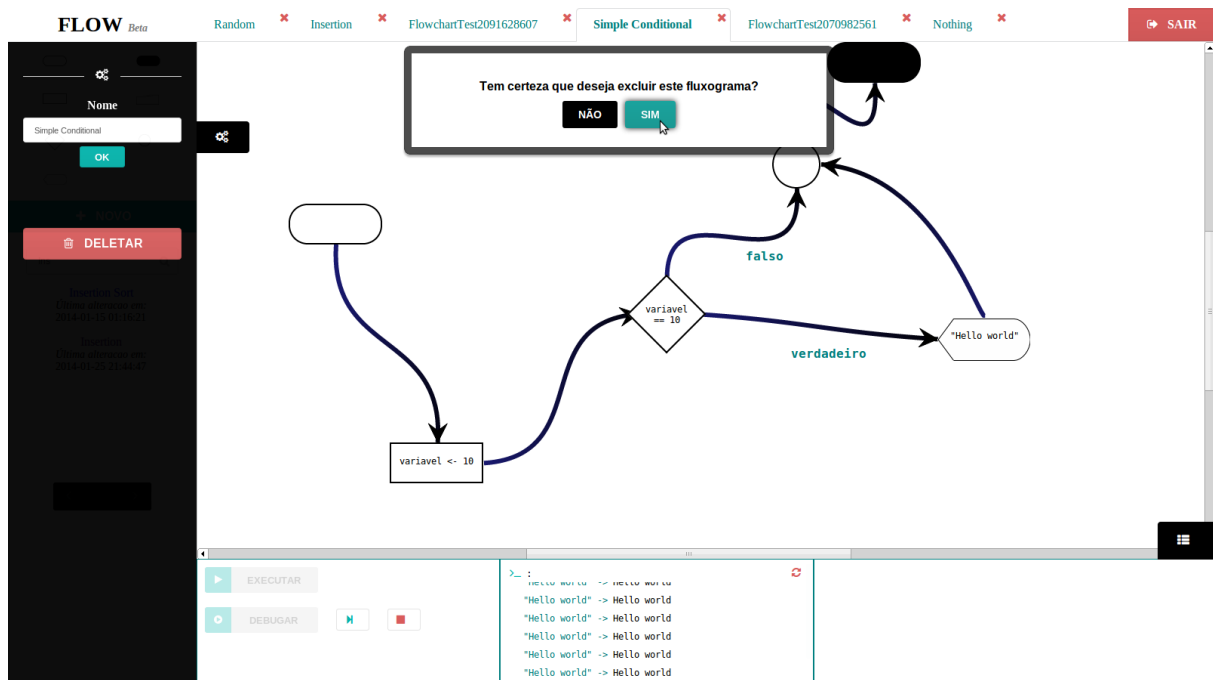


Figura 21 – Flow, Configurações e execução

A Figura 22 exibe o fluxograma “Simple Conditional” durante o processo de execução passo a passo deste. O elemento que está correntemente seleccionado pela execução é exibido em destaque e as saídas geradas pela execução do fluxograma podem ser visualizadas na área do *console* onde encontra-se a barra de rolagem.

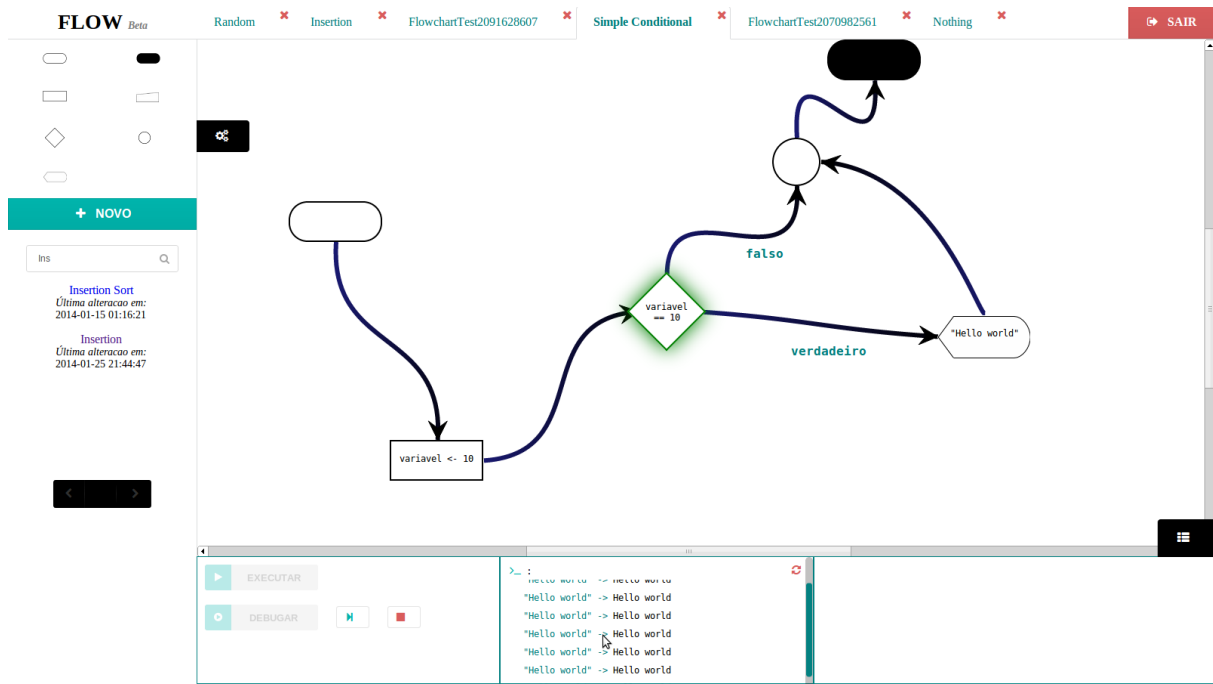


Figura 22 - Flow: execução e busca

## 6. CONSIDERAÇÕES FINAIS

Este trabalho foi realizado com o auxílio financeiro de uma bolsa advinda do Programa de Bolsas de Formento às Ações de Graduação. O trabalho apresenta uma nova ferramenta para o auxílio ao ensino e aprendizado de algoritmos e lógica de programação, esta ferramenta é denominada Flow.

A ferramenta Flow permite que um usuário crie fluxogramas e adicione a estes elementos geométricos que contenham trechos de pseudocódigo, de modo que os fluxogramas possam representar com precisão diversos algoritmos. Estas representações de algoritmos possibilitam que os fluxogramas possam ser executados de forma similar a programas de computador, tornando possível ao usuário visualizar graficamente os conceitos e funcionamento de algoritmos. Além disso, os elementos geométricos de um fluxograma permitem um alto nível de interação com o usuário, sendo passíveis de serem arrastados, deletados, terem seus valores internos alterados, serem conectados uns com os outros, possuírem suas conexões deletadas ou transferidas de um elemento para outro, entre outras. Adicionalmente, todas as manipulações realizadas nos elementos geométricos de um fluxograma são automaticamente salvas no banco de dados, sem que seja necessária ação do usuário para tal. Essas alterações também podem ser revertidas pelo usuário, desde que o fluxograma permaneça aberto entre suas ocorrências.

O desenvolvimento deste trabalho apresentou diversas dificuldades, sendo as mais proeminentes: a disciplina necessária para a aplicação do processo de *software* selecionado, juntamente com as práticas por este definidas; o desenvolvimento do código JavaScript da ferramenta.

A dificuldade encontrada na aplicação do processo de *software* selecionado se deu principalmente devido a não familiaridade prévia com este. Embora a razão de sua escolha tenha sido o fator de ser um processo direcionado para projetos em que apenas uma pessoa está envolvida, não foi levado em conta a possível dificuldade que poderia surgir com a utilização de um processo desconhecido, e que impõe a utilização de diversas técnicas, também desconhecidas previamente ao trabalho. Assim quantidades consideráveis de tempo foram gastas no estudo do processo, das práticas que este agrega e das diversas tecnologias e ferramentas que estas práticas demandam para uma utilização adequada.

Em relação a dificuldade encontrada na codificação JavaScript, esta ocorreu devido ao fato de que a linguagem possui uma relevante complexidade de estruturação e, apesar de sua dinamicidade, apresenta também uma incomum implementação de orientação a objetos sem a existência de classes e uma possibilidade de mistura de diferentes paradigmas de programação, características que acabam por facilitar com que um desenvolvedor inexperiente com a linguagem produza código de má qualidade. Adicionalmente, por ser executada no *browser* do usuário, o código JavaScript demandou um cuidado extra em questões de desempenho e consumo de recursos.

Além de prover uma ferramenta que possa vir a auxiliar o ensino e aprendizado de algoritmos e lógica de programação, a existência da ferramenta Flow gera diversas possibilidades futuras:

- Avaliações da eficácia da ferramenta em seus objetivos primários (ensino e aprendizado de algoritmos e lógica de programação);
  - Avaliação do nível de usabilidade da ferramenta;
  - Exportação de imagens do fluxograma;
  - Possibilidade de ampliação da diversidade de elementos geométricos, o que em conjunto com a exportação de imagens tornaria a ferramenta útil para demonstrações de fluxogramas para fins não relacionados a algoritmos;
- Possibilitar a execução de diversos fluxogramas de demonstração na página inicial da ferramenta;
- Possibilitar a geração de código-fonte em diferentes linguagens de programação através dos fluxogramas;
- Possibilitar a geração de fluxogramas através de código-fonte de diferentes linguagens de programação;
- Permitir o uso colaborativo de fluxogramas entre usuários.

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

BERG, Alexandre. FIGUEIRÓ, Joice Pavék. **Lógica de Programação**. 3ed. ULBRA, 2006.

BERGMANN, Sebastian. **Integrating PHP Projects with Jenkins**. Sebastopol. O'Reilly Media, 2011.

CHAUDHURI, Anil Bikas. **The Art of Programming Through Flowcharts & Algorithms**. Laxmi Publications. pág 1-32, 2005.

CODECEPTION: documentação. Disponível em: <<http://codeception.com/>>. Acesso em: 10 de out. 2012.

CORMEN, Thomas H (at all). **Introduction to Algorithms**. 3 ed. EUA. The MIT Press. pág 5-14, 2009.

CROCKFORD, Douglas. **Javascript: The Good Parts**. Sebastopol. O'Reilly Media, 2008.

DAVID, Flanagan. **Javascript: The Definitive Guide**. 6 ed. Sebastopol. O'Reilly Media, 2011.

DZHUROV, Yani. KRASTEVA, Iva. ILIEVA, Sylvia. **Personal Extreme Programming – An Agile Process for Autonomous Developers**. Demetra EOOD, 2009.

FOWLER, Martin. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3 ed. Boston. Addison-Wesley Professional, 2003.

HUMPREY, Watts S. **The Personal Software Process (PSP)**. Software Engineering Institute, 2000.

JSPLUMB: documentação. Disponível em: <<http://jsplumbtoolkit.com/doc/>>. Acesso em: 15 de out. 2012.

KRUG, Steve. **Don't Make Me Think !: A Common Sense Approach to Web Usability**. 2 ed. Berkeley. New Riders, 2006.

MANZANO, José Augusto Navarro Garcia. **Revisão e Discussão da Norma ISO 5807 – 1985 (E) Proposta para Padronização Formal da Representação Gráfica da Linha de Raciocínio Lógico Utilizada no Desenvolvimento da Programação de Computadores a ser Definida no Brasil**. THESIS, São Paulo, ano I, v. 1, p. 1-31, 1o Semestre, 2004.

MACINTYRE, Peter. DANCHILA, Brian. GOGALA, Mladen. **Pro PHP Programming**. New York. Apress, 2011.

MAK, Ronald. **Writing Compilers and Interpreters: A Software Engineering Approach**. 3 ed. Indianapolis. Wiley, pág 1-60, 2009.

MARTIN, Robert C (at all). **Clean Code: A handbook of Agile Craftsmanship**. Stoughton. Prentice Hall, 2008.

OSMANI, Addy. **Learning Javascript Design Patterns**. Sebastopol. O'Reilly Media, 2012.

POSTGRESQL: documentação. Disponível em: <<http://www.postgresql.org/files/documentation/pdf/9.2/postgresql-9.2-US.pdf>>. Acesso em: 01 de out. 2012.

REDBEANPHP: documentação. Disponível em: <<http://redbeanphp.com/>>. Acesso em: 20 de out. 2012.

SEMANTIC-UI: documentação. Disponível em: <<http://jsplumbtoolkit.com/doc/>>. Acesso em: 10 de set. 2012.

SOMMERVILLE, Ian. **Engenharia de Software**. 9 ed. São Paulo. Person Prentice Hall. pág 57-68, 2011.

STEFANOV, Stoyan. **Javascript Patterns**. Sebastopol. O'Reilly Media, 2010.

ZAKAS, Nicholas C. **High Performance JavaScript**. Sebastopol. O'Reilly Media, 2010.



## **APÊNDICE A – Requisitos de Software**

## **Alterações nos requisitos**

Os requisitos “Tornar fluxograma público”, “Exportar fluxograma como imagem”, “Compartilhar imagem de fluxograma em rede social” e “Exibir relatório” estão presentes na proposta deste trabalho (contida no APÊNDICE C – Proposta), contudo não foram implementados no desenvolvimento deste projeto, tendo sido substituídos pelo requisito “Executar fluxograma passo a passo”. Tal ocorrência se deu devido a complexidade da implementação deste novo requisito, o qual foi introduzido pela constatação de que traria uma funcionalidade que agregaria mais valor e utilidade a ferramenta que os requisitos substituídos.

## **Requisitos funcionais**

### **RF001 – Realizar cadastro**

#### **Descrição:**

Permite ao usuário a realização de cadastro no sistema.

#### **Restrições:**

∅

### **RF002 – Realizar login**

#### **Descrição:**

Permite que um usuário realize o login por meio de uma conta previamente cadastrada, vide **RF001**.

#### **Restrições:**

Usuário deve possuir cadastro no sistema.

### **RF003 – Manter fluxograma**

#### **Descrição:**

Permite que um usuário possa criar, visualizar, alterar e excluir fluxogramas.

#### **Restrições:**

Usuário deve estar logado, vide **RF002**.

### **RF004 – Manter elementos do fluxograma**

**Descrição:**

Permite que um usuário possa criar, visualizar, alterar e excluir elementos do fluxograma.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

**RF005 – Executar fluxograma****Descrição:**

Permite ao usuário executar um fluxograma, operação que validará se o fluxograma está correto e também irá gerar uma saída de dados visível ao usuário.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

Fluxograma deve estar aberto pelo no momento da operação.

A saída de dados só irá acontecer caso o fluxograma seja válido e seja utilizado, de forma correta, o símbolo adequado para a saída de dados durante a construção do fluxograma.

**RF006 – Executar fluxograma passo a passo****Descrição:**

Permite ao usuário executar um fluxograma passo a passo, de forma similar ao “debug” provido por diversas *IDEs*.

**Restrições:**

Mesmas restrição aplicadas ao **RF005**.

**RF007 – Retornar estado de elementos do fluxograma****Descrição:**

Permite que um usuário possa retornar elementos de um fluxograma a um estado anterior durante o processo de alteração destes. Assim se um símbolo qualquer for movido do ponto “X” para o ponto “Y” será possível retornar ao ponto “X” através da operação adequada.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

Ao salvar um fluxograma apenas seu último estado é salvo no banco de dados, logo fluxogramas recém-abertos não possuem estados anteriores, desse modo não podem ser revertidos.

Fluxograma deve ter sofrido ao menos uma alteração.

Fluxograma deve estar sendo editado pelo usuário no momento da operação.

### **Requisitos não funcionais**

RNF001 - A ferramenta deverá apresentar interface amigável

#### **Descrição:**

A interface deve ser o mais amigável possível ao usuário, sendo essa característica referente à utilização de elementos gráficos comuns em softwares similares e posicionamento de elementos também de modo comum a softwares similares fazendo o usuário se sentir familiar e confortável com essa interface.

#### **Restrições:**

Este é um requisito altamente subjetivo e embora sua implantação seja fortemente desejada ela não pode ser completamente garantida.

**RNF002 - A ferramenta deverá apresentar um bom nível de usabilidade**

#### **Descrição:**

A interface deverá ter um mínimo de cuidado com questões de usabilidade, possibilitando que os usuários possam utilizar a ferramenta com mais facilidade.

#### **Restrições:**

Esse requisito não abrange questões de acessibilidade.

## **APÊNDICE B - Protótipos de tela finais**

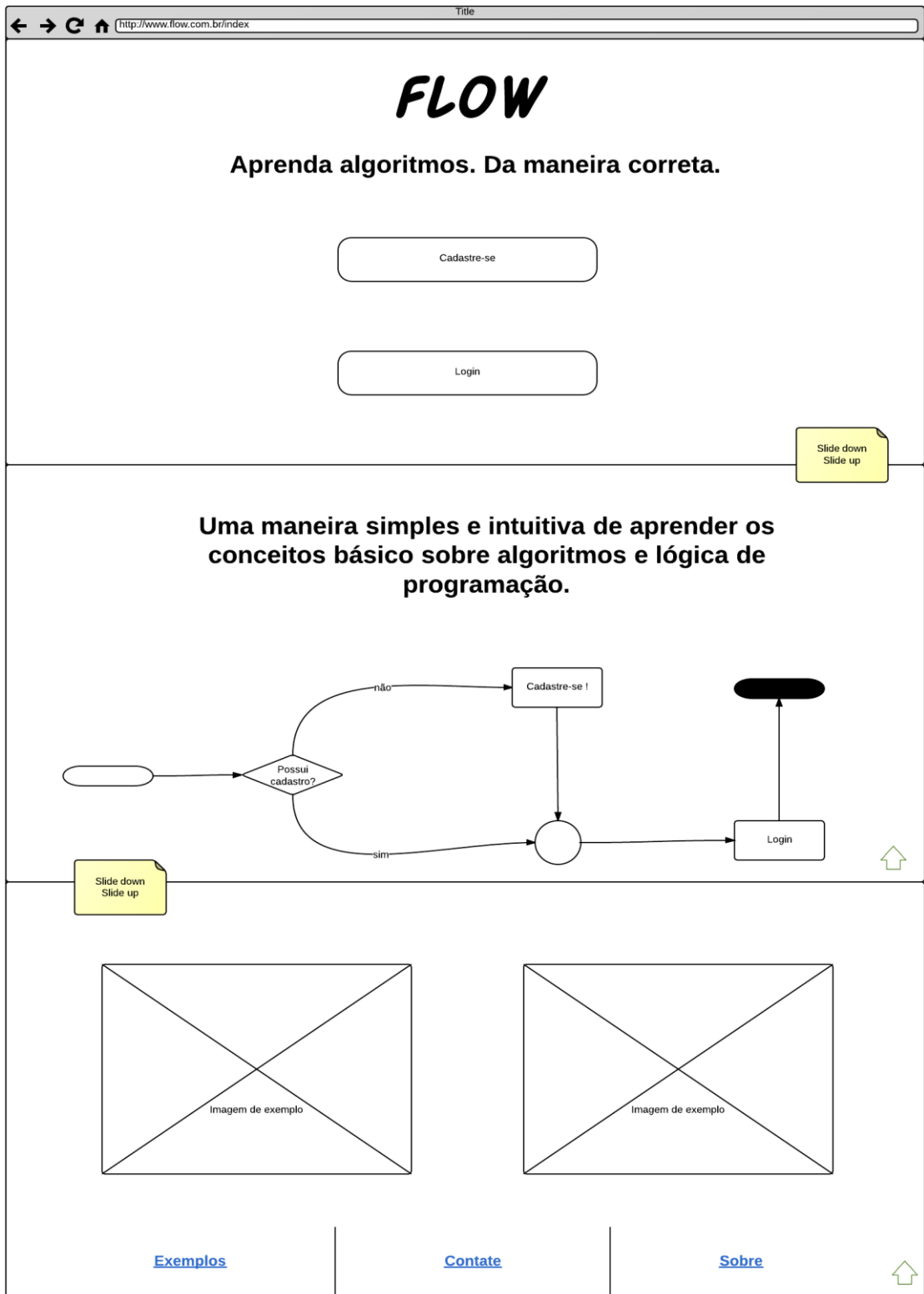


Figura 23 – Protótipo de tela final da *home*

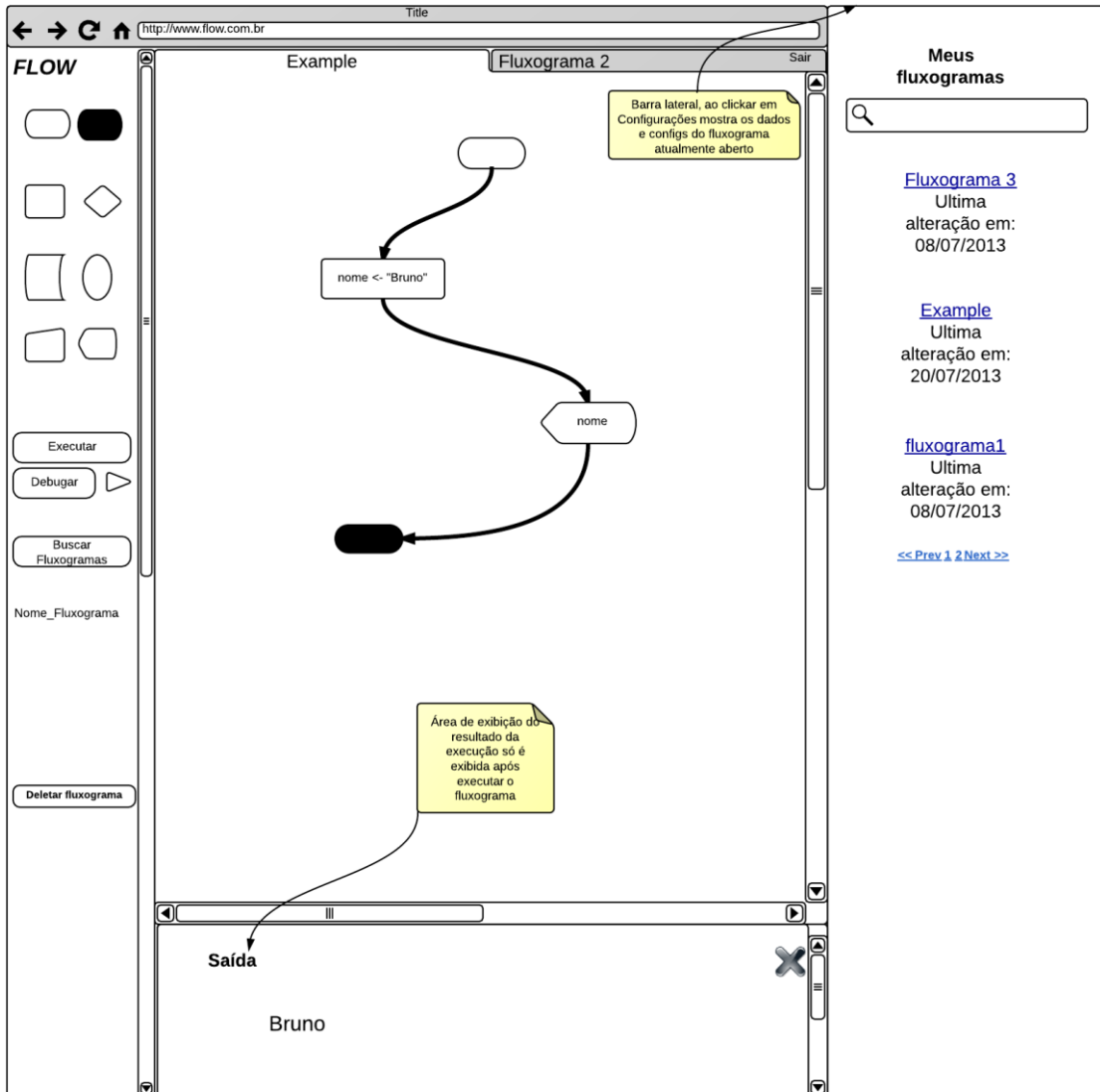


Figura 24 – Protótipo de tela final do fluxograma

## **APÊNDICE C – Proposta**



UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE  
SISTEMAS

BRUNO ROBERTO BÚRIGO

**FLOW – FERRAMENTA WEB PARA CRIAÇÃO DE FLUXOGRAMAS  
EXECUTÁVEIS**

PROPOSTA PARA TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2013

BRUNO ROBERTO BÚRIGO

**FLOW – FERRAMENTA WEB PARA CRIAÇÃO DE FLUXOGRAMAS  
EXECUTÁVEIS**

Proposta para Trabalho de Conclusão de Curso do  
Curso de Tecnologia em Análise e Desenvolvimento  
de Sistema da Universidade Tecnológica Federal do  
Paraná.

Orientador: Adriano Rívogli da Silva.

CORNÉLIO PROCÓPIO

2013

## RESUMO

Na área de tecnologia da informação, mais especificamente a de programação para computadores, seja em uma universidade ou qualquer outro meio educativo, lógica de programação e algoritmos se destacam como os primeiros e mais importantes conceitos dos quais uma pessoa necessita para se tornar um bom programador. Um dos recursos comumente usados no início do processo de aprendizado, ou ensino desses assuntos são os fluxogramas, sendo estes o enfoque deste trabalho, o qual pretende desenvolver um software que irá realizar uma integração maior entre algoritmos e fluxogramas, proporcionando uma ferramenta que permitirá a criação de fluxogramas que possam, por meio de uma mínima adição de pseudocódigo, ser executados de maneira similar a um código real, assim oferecendo um auxílio ao aprendizado de algoritmos e lógica de programação, principalmente (mas não restrito a) iniciantes nesses assuntos. Além disso, o software apresentará um enfoque nas questões de usabilidade e sociabilidade, sendo a primeira dedicada a prover uma interface amigável, intuitiva e que induza o usuário usufruir o máximo das funcionalidades disponíveis e a segunda permitindo que usuários possam compartilhar fluxogramas próprios com outros usuários ou até mesmo compartilhar fotos de fluxogramas próprios em redes sociais.

**Palavras-chave:** Fluxogramas. Execução. Algoritmos.

## Lista de ilustrações

Figura 1 - Protótipo da tela de criação/edição de fluxogramas .....	10
Figura 2 - Protótipo da tela de busca de fluxogramas .....	11
Figura 3 - Exemplos de estrutura de fluxogramas.....	21
Figura 4 - Fases do PXP .....	25

## Lista de tabelas

Tabela 1 - Símbolos geométricos para fluxogramas .....	19
Tabela 2 - Cronograma .....	28

## **Lista de abreviaturas e siglas**

PXP – *Personal Extreme Programming*

PSP – *Personal Software Process*

XP – *Xtreme Programming*

ISO – *International Organization for Standardization*

HTML – *HyperText Markup Language*

CSS – *Cascading Style Sheets*

## Sumário

1. INTRODUÇÃO .....	6
2. OBJETIVOS E RESULTADOS ESPERADOS .....	8
3. REQUISITOS DO SOFTWARE .....	13
3.1. Requisitos funcionais .....	13
3.2. Requisitos não funcionais .....	16
4. FUNDAMENTAÇÃO TEÓRICA.....	18
Algoritmo.....	18
Fluxograma .....	18
5. METODOLOGIA .....	23
6. TECNOLOGIAS .....	27
7. CRONOGRAMA.....	28
8. CONSIDERAÇÕES FINAIS .....	29
9. REFERÊNCIAS.....	30

## 1. INTRODUÇÃO

O conceito de algoritmos é profundamente enraizado no conhecimento de qualquer programador. Eles são também o ponto de partida no início do aprendizado de qualquer um que almeje adentrar na arte da programação e futuramente tornar-se um programador. O problema, contudo, reside no fato de que muitas vezes esse aprendizado inicial é difícil, pois a compreensão de que são necessárias séries de instruções específicas e exatas em uma ordem bem-definida e que juntas formem uma sequência lógica funcional não é algo trivial a maioria das pessoas. Além disso, algoritmos, em especial no contexto computacional aqui referenciado, devem ser compostos de passos exatos, não ambíguos e que não realizem inferências baseadas em passos anteriores<sup>3</sup>, novamente algo não trivial para a maioria das pessoas.

Para contornar as dificuldades iniciais existentes no aprendizado de algoritmos são utilizados diversos métodos de ensino a fim de facilitar o aprendizado, referenciando tanto salas de aula quanto formas mais heterodoxas como apostilas e livros para autodidatas, sendo um desses o uso de fluxogramas como forma de representar graficamente o conceito de algoritmos. Fluxogramas tendem a ser úteis e bem aceitos por iniciantes, afinal a preferência por figuras e representações gráficas em lugar de textos é quase uma unanimidade em qualquer área do conhecimento. Contudo a utilização de fluxogramas apenas como forma de representação abstrata de algoritmos pode ser vista como uma limitação desnecessária, pois, pseudocódigos usados na representação de algoritmos são também puramente conceituais e ainda assim existem ferramentas disponíveis que permitem a execução destes, exatamente com o intuito de demonstrar claramente o “resultado” daquela representação e se ela está correta ou não.

Nesse contexto encontra-se a motivação deste trabalho, cuja proposta central é a construção de uma ferramenta que venha a ultrapassar a limitação dos fluxogramas, descrita no parágrafo acima, possibilitando que estes possam ser executados de modo a gerarem uma saída de dados “real”.

---

<sup>3</sup> Embora existam algoritmos computacionais que realizem inferências e estes sejam perfeitamente válidos suas aplicações e objetivos fogem ao escopo deste trabalho, sendo assim sua existência é ignorada.



Embora existam inúmeras ferramentas, tanto em ambientes web quanto desktop, para criação de fluxogramas e muitas destas disponibilizarem diversos recursos que não estarão presentes no software aqui proposto, a grande maioria delas não é exclusivamente dedicada a este tipo de diagrama e, principalmente, não apresenta a possibilidade de execução destes<sup>4</sup>. Essa possibilidade de criação de diagramas executáveis almeja prover um grande auxílio ao ensino e aprendizado de algoritmos, pois como já dito este assunto pode apresentar um excessivo nível de complexidade a leigos. Com a utilização da ferramenta aqui proposta o primeiro contato com algoritmos poderá se tornar mais fácil, pois além de poder criar uma representação gráfica do conceito o usuário poderá também verificar se esta representação está correta e observar sua “saída”.

---

<sup>4</sup> Mais especificamente, foi encontrada apenas uma ferramenta com proposta similar a deste trabalho, o programa RAPTOR, disponível em <http://raptor.martincarlisle.com>, contudo se trata de uma aplicação desktop voltada exclusivamente ao sistema operacional *Windows*, o que limita suas funcionalidades e abrangência, excluindo possíveis usuários de sistemas operacionais diferentes e impossibilitando uma interação entre esses diretamente através da ferramenta, estando assim abertas diversas possibilidades para exploração pelo projeto aqui proposto.

## 2. OBJETIVOS E RESULTADOS ESPERADOS

Com a conclusão deste trabalho espera-se a construção da ferramenta FLOW, a qual interface amigável e será de utilização possível apenas por meio de um navegador web. Seu principal objetivo será permitir ao usuário a criação de fluxogramas “executáveis”, retirando assim a natureza puramente abstrata inerente a estes, desta maneira possibilitando que além de representações gráficas de algoritmos eles possuam também o comportamento de programas reais, computando operações lógico-matemáticas, aceitando entradas e realizando saídas de dados. Para que isso seja possível será disponibilizada uma gama restrita de símbolos para criação do fluxograma e será definida a sintaxe dos pseudocódigos que deverão ser utilizados conjuntamente com os símbolos gráficos, permitindo assim uma completa exatidão lógica na representação desse diagrama.

Para que possa utilizar a ferramenta será necessário que o usuário realize *login* no sistema, o que implica realização de um cadastro prévio, informando dados como nome, e-mail e senha (excetuando-se usuários que realizem login por meio de rede social ou serviço de e-mail externo, para esses não será requerida a realização desse cadastro). Caso o usuário venha a esquecer sua senha de login, existirá a possibilidade de recuperação desta através do e-mail informado no cadastro.

Outras funcionalidades importantes referentes a fluxogramas e que serão implementadas são as seguintes:

1. Durante o processo de alteração de um fluxograma criado todas as alterações feitas serão constantemente salvas de modo automático, sem a necessidade de uma ação explícita por parte do usuário para que isso ocorra (embora seja dada a eles esta opção) e sem que seja necessário a atualização da página no navegador;
2. Durante o processo de alteração de um fluxograma criado os “estados” deste serão constantemente salvos de modo a permitir que o usuário retorne a um estado anterior ou avance a um estado posterior (apenas após retorno), com a utilização de atalho do teclado ou botão na janela de edição, novamente, sem que seja necessária a atualização da página;
3. Será possível ao usuário exportar imagens de fluxogramas próprios ou compartilhados por outros, salvando-o no próprio computador;

4. Um usuário poderá compartilhar seus fluxogramas com outros usuários, tornando-os públicos.

5. Será possível publicar fotos de fluxogramas próprios na rede social *Facebook*.

6. O usuário poderá visualizar relatórios contendo dados como: quantos fluxogramas possui, quantos destes foram executados com sucesso e quantos falharam em sua última execução.

Os usuários que utilizarão o programa serão denominados “alunos” e é essencial deixar claro que embora seja empregado o termo “aluno” a ferramenta não irá restringir sua utilização à meios acadêmicos, assim um “aluno” nada mais é que um usuário comum, cadastrado de maneira convencional ou que efetue seus logins por meio de conta no Facebook ou Gmail.

A fim de possibilitar uma maior compreensão de como se dará o funcionamento da ferramenta foram elaborados protótipos de duas telas do sistema, seguem abaixo as imagens destes (Figura 1 e Figura 2):

Figura 25 - Protótipo da tela de criação/edição de fluxogramas

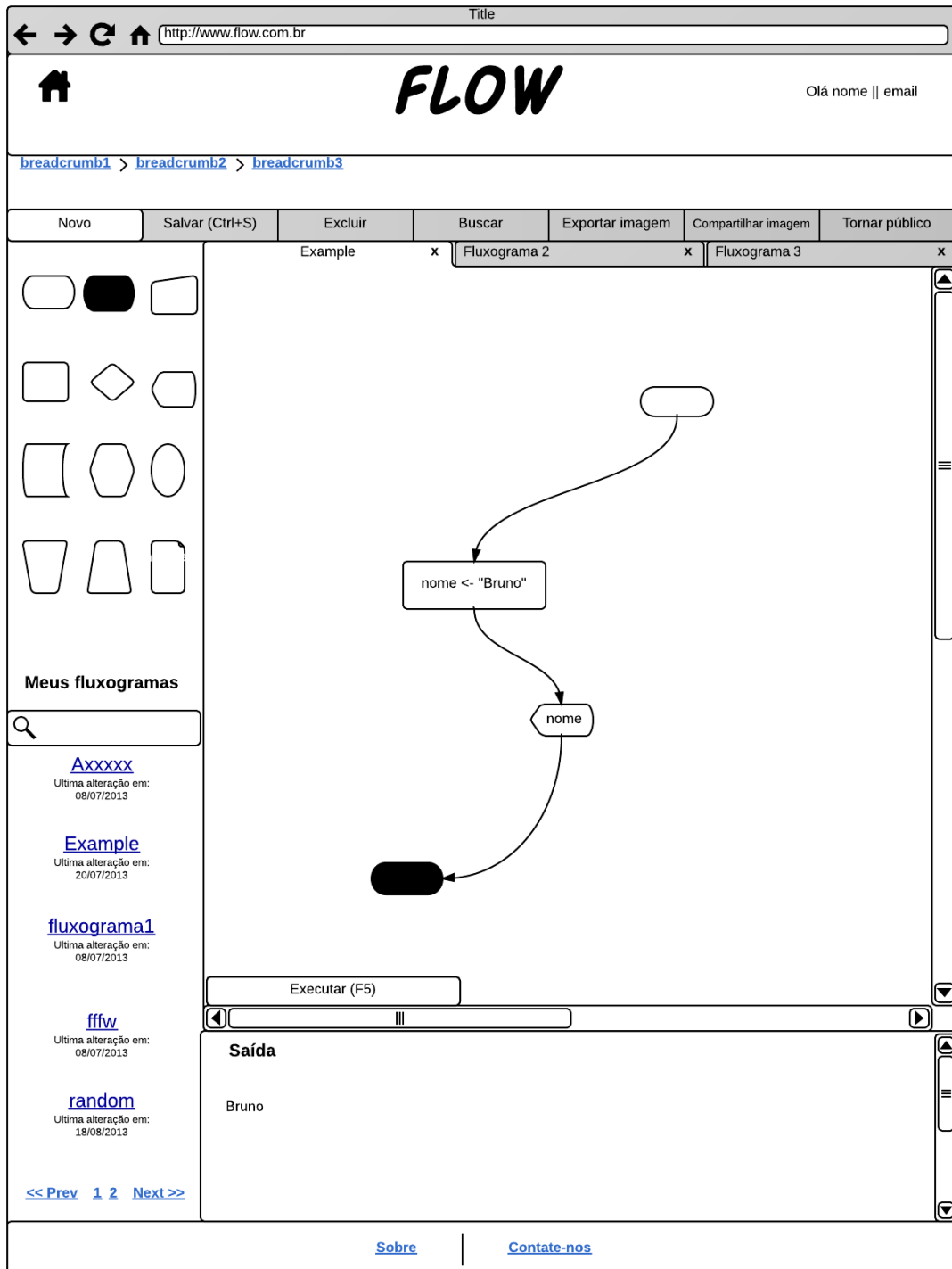
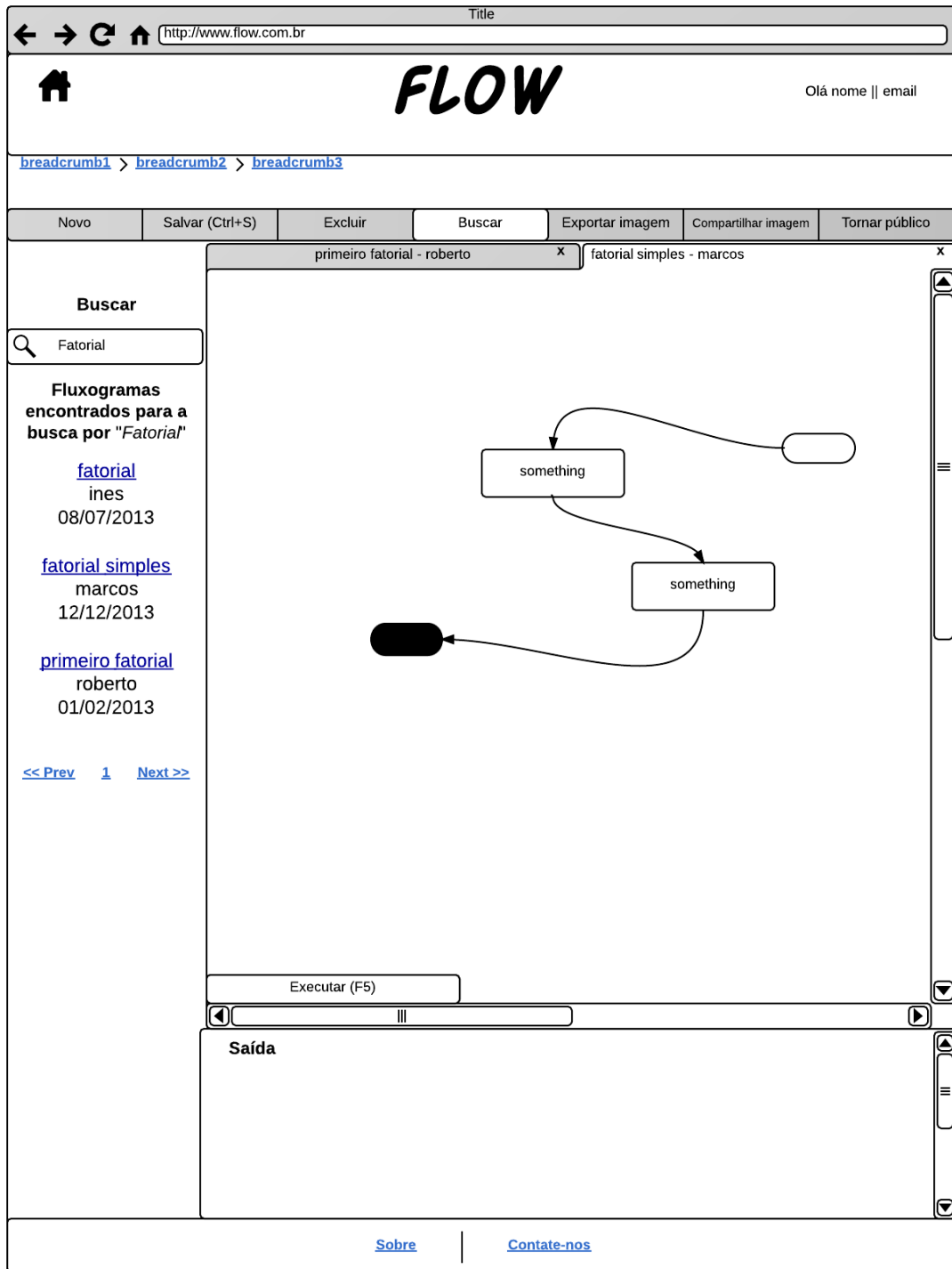


Figura 26 - Protótipo da tela de busca de fluxogramas



Por fim faz-se necessário afirmar que a ferramenta proposta visa ser um acréscimo ao ensino e aprendizado de algoritmos e lógica de programação e que foge ao escopo deste projeto a verificação prática da eficácia da ferramenta nestes propósitos, pois embora esta asserção possa ser uma questão essencial sua realização exigiria um projeto a parte, sendo assim deixada para trabalhos futuros.

### **3. REQUISITOS DO SOFTWARE**

#### **3.1. Requisitos funcionais**

##### **RF001 – Realizar cadastro**

###### **Descrição:**

Permite ao usuário a realização de cadastro no sistema.

###### **Restrições:**

∅

##### **RF002 – Realizar login**

###### **Descrição:**

Permite que um usuário realize o login por meio de uma conta previamente cadastrada, vide **RF001**, ou através de conta válida que possua na rede social Facebook ou no serviço de e-mail Gmail.

###### **Restrições:**

Usuário deve possuir cadastro no sistema ou conta válida na rede social Facebook ou conta válida no serviço de e-mail Gmail.

##### **RF003 – Recuperar senha**

###### **Descrição:**

Permite que um usuário possa recuperar sua senha utilizando o e-mail definido no cadastro.

###### **Restrições:**

A senha só poderá ser recuperada para usuários que venham a se cadastrar no sistema. Para usuários que utilizarem login pelos serviços do Facebook ou Gmail isso não será possível.

##### **RF004 – Manter fluxograma**

###### **Descrição:**

Permite que um usuário possa criar, visualizar, alterar e excluir fluxogramas.

###### **Restrições:**

Usuário deve estar logado, vide **RF002**.

As operações de alteração e exclusão são possíveis apenas para fluxogramas criados pelo próprio usuário.

A operação de visualização de fluxogramas é possível apenas para fluxogramas criados pelo próprio usuário ou para fluxogramas de outros usuários, desde que marcados como públicos.

### **RF005 – Tornar fluxograma público**

#### **Descrição:**

Permite que um usuário torne um de seus fluxogramas públicos, o que irá permitir a outros usuários visualizar, executar e exportar imagem deste.

#### **Restrições:**

Usuário deve estar logado, vide **RF002**.

### **RF006 – Buscar fluxograma**

#### **Descrição:**

Permite que um usuário busque fluxogramas criados por outros usuários.

#### **Restrições:**

Usuário deve estar logado, vide **RF002**.

Só serão visíveis na busca fluxogramas tornados públicos pelos usuários que os criaram, vide **RF005**.

### **RF007 – Executar fluxograma**

#### **Descrição:**

Permite ao usuário executar um fluxograma, operação que validará se o fluxograma está correto e também irá gerar uma saída de dados visível ao usuário.

#### **Restrições:**

Usuário deve estar logado, vide **RF002**.

Fluxograma deve estar sendo visualizado (no caso de o fluxograma estar sendo editado, fica implícito que ele está sendo visualizado) pelo usuário no momento da operação.

A saída de dados só irá acontecer caso o fluxograma seja válido e seja utilizado, de forma correta, o símbolo adequado para a saída de dados durante a construção do fluxograma.



## **RF008 – Retornar estado de fluxograma**

### **Descrição:**

Permite que um usuário possa retornar um fluxograma a um estado anterior durante o processo de alteração deste, assim se um símbolo qualquer de um fluxograma qualquer for movido do ponto “X” para o ponto “Y” será possível retornar ao ponto “X” através da operação adequada (atalho de teclado, botão, etc. Ainda a ser definido).

### **Restrições:**

Usuário deve estar logado, vide **RF002**.

Ao salvar um fluxograma apenas seu último estado é salvo no banco de dados, logo fluxogramas recém-abertos não possuem estados anteriores, desse modo não podem ser revertidos.

Fluxograma deve ter sofrido ao menos uma alteração.

Fluxograma deve estar sendo editado pelo usuário no momento da operação.

## **RF009 – Avançar estado de fluxograma**

### **Descrição:**

Permite que um usuário possa avançar um fluxograma a um estado posterior durante o processo de alteração deste, assim se um símbolo qualquer de um fluxograma qualquer for movido do ponto “X” para o ponto “Y” e esta mudança for revertida pelo retorno de estado (**RF008**) este retorno poderá ser desfeito, movendo o símbolo novamente ao ponto “Y”.

### **Restrições:**

Usuário deve estar logado, vide **RF002**.

Estado do fluxograma deve ter sido retornado ao menos uma vez, vide **RF008**.

Fluxograma deve estar sendo editado pelo usuário no momento da operação.

## **RF010 – Exportar fluxograma como imagem**

### **Descrição:**

Permite que um usuário possa exportar a imagem de um fluxograma para o próprio computador.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

Fluxograma deve estar sendo visualizado pelo usuário no momento da operação.

**RF011 – Compartilhar imagem de fluxograma em rede social****Descrição:**

Permite que um usuário possa compartilhar a imagem de um fluxograma próprio na rede social Facebook ou Gmail.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

Válido apenas para fluxogramas criados pelo próprio usuário.

Usuário deve possuir uma conta válida na rede social a ser utilizada.

Fluxograma deve estar sendo visualizado pelo usuário no momento da operação.

**RF012 – Exibir relatório****Descrição:**

Permite que um usuário possa visualizar informações de uso referentes a seus fluxogramas.

**Restrições:**

Usuário deve estar logado, vide **RF002**.

Apenas dados do próprio usuário.

**3.2. Requisitos não funcionais****RNF001 - A ferramenta deverá apresentar interface amigável****Descrição:**

A interface deve ser o mais amigável possível ao usuário, sendo essa característica referente à utilização de elementos gráficos comuns em softwares similares e posicionamento de elementos também de modo comum a softwares similares fazendo o usuário se sentir familiar e confortável com essa interface.

**Restrições:**

Este é um requisito altamente subjetivo e embora sua implantação seja fortemente desejada ela não pode ser completamente garantida.

**RNF002 - A ferramenta deverá apresentar um bom nível de usabilidade**

**Descrição:**

A interface deverá ter um mínimo de cuidado com questões de usabilidade, possibilitando que os usuários possam utilizar a ferramenta com mais facilidade.

**Restrições:**

Esse requisito não abrange questões de acessibilidade.

## 4. FUNDAMENTAÇÃO TEÓRICA

### Algoritmo

Um algoritmo pode ser descrito como uma série de instruções lógicas organizadas de modo a solucionar um problema específico. Ainda segundo CORMEN et al.(2009) em “Introduction to Algorithms” temos a definição de que um algoritmo é qualquer procedimento computacional bem definido que receba valores de entrada, realize uma série de operações e produza valores de saída, a qual se encaixa perfeitamente no contexto de algoritmos utilizado para este trabalho.

### Fluxograma

Fluxogramas são diagramas que utilizam figuras geométricas para ilustrar uma série de passos a serem percorridos a fim de se completar uma tarefa. Esta definição é extremamente similar a de um algoritmo, tanto que segundo CHAUDHURI (2005), p 2, temos “[...] *Formally speaking, a flowchart is a diagrammatic representation of the steps of an algorithm.*”<sup>5</sup>.

As figuras geométricas de um fluxograma são utilizadas como meio de representação do fluxo de ações a serem realizados por este, cada uma possui um nome e é responsável por ilustrar uma determinada operação. Elas são conectadas através de setas a fim de representar o fluxo de execução da tarefa que está sendo realizada pelo fluxograma em questão.

A ISO 5807-1985 define a existência de quatro tipos distintos de fluxogramas, são eles: *data flowcharts* (diagramas de fluxo de dados), *program flowcharts* (diagramas de fluxo de programa), *system flowcharts* (diagramas de fluxo de sistema), *program network charts* (diagramas de programas de rede) e *system resource charts* (diagramas de recursos de sistema). Apesar de sua existência essa norma é largamente ignorada devido a inúmeros fatores, entre eles o fato de sua última revisão ter sido realizado em 1985 e como apontado por MANZANO (2004), ela é extremamente genérica e não define de maneira clara os critérios para a elaboração de diagramas que representem a linha de raciocínio lógico a ser utilizada

---

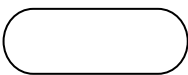
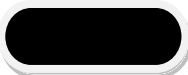
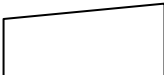
<sup>5</sup> Formalmente falando, um fluxograma é uma representação em forma de diagrama dos passos de um algoritmo.

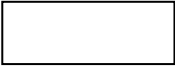
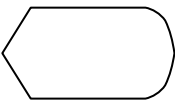
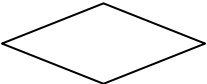
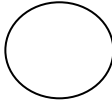

por um programador de computadores, além de sua definição para os símbolos geométricos ser demasiado simplista.

Como para este trabalho faz-se necessário um alto nível de detalhamento e asserção sobre os símbolos geométricos de um fluxograma e como esses podem ser utilizados em conjunto para formar um diagrama válido, a norma ISO 5807-1985 será tomada meramente como base, não servindo como modelo literal para o software aqui proposto.

Abaixo segue a Tabela 1, com os símbolos geométricos a serem utilizados neste trabalho para a construção de fluxogramas:

*Tabela 4 - Símbolos geométricos para fluxogramas*

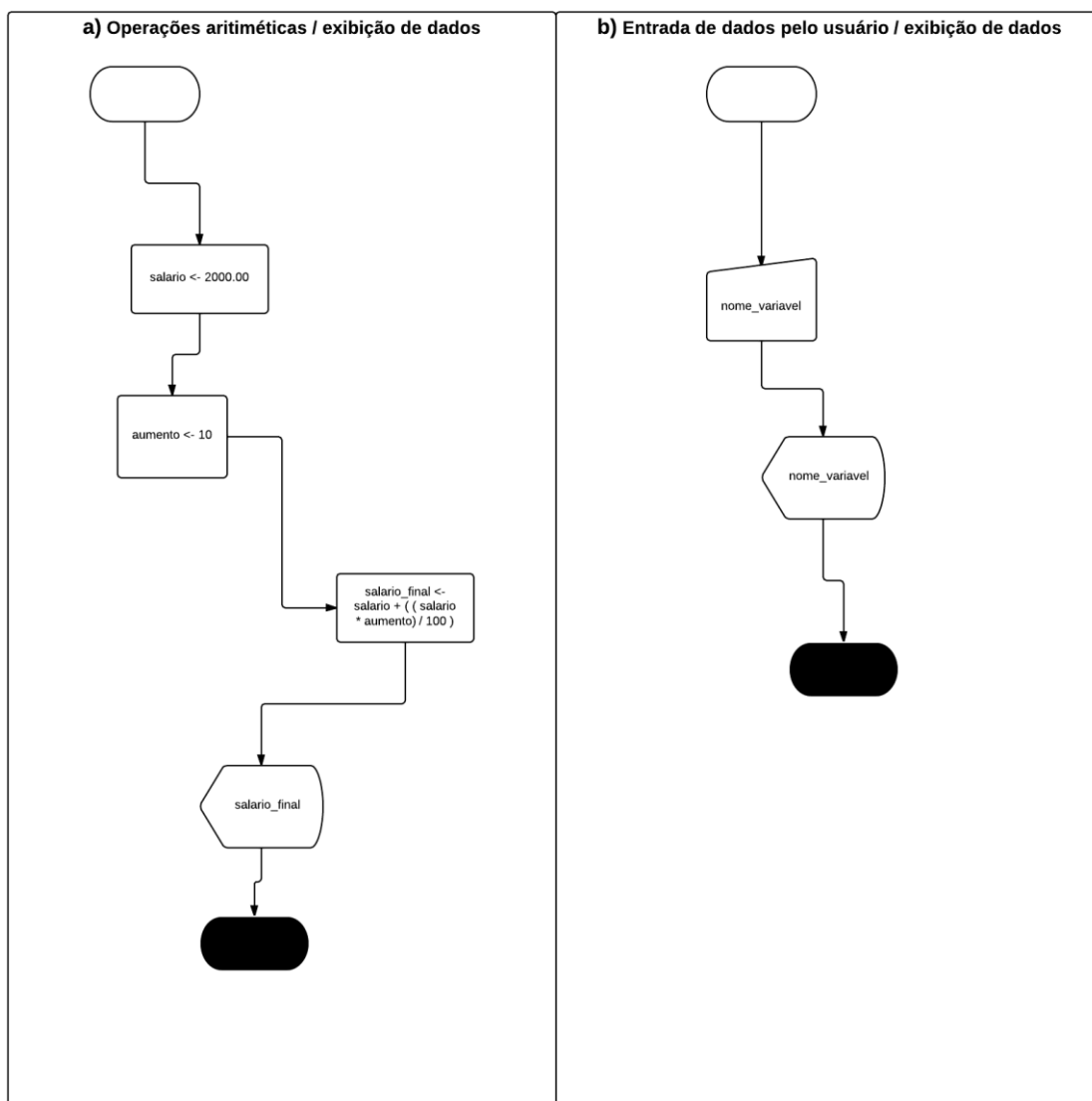
<b>Símbolo</b>	<b>Nome</b>	<b>Significado</b>
	Início	Simboliza o início de um fluxograma.
	Fim	Simboliza o fim de um fluxograma.
	Entrada manual	Simboliza entrada manual de dados pelo usuário, previsivelmente pelo teclado.

	Processamento	Representa a execução de operações lógico-matemáticas ou a atribuição de valores a variáveis.
	Exibição	Simboliza a exibição de valores de forma que o usuário possa visualizá-los.
	Decisão	Simboliza um teste lógico alternativo de instruções
	Conector	Tem a função de unir fluxos de execução previamente separados por uma ou mais operações de <b>decisão</b> .
	Seta	Representa a ligação entre dois elementos. A direção indica o fluxo de execução

A conexão desses símbolos em uma sequência lógica aceitável, sendo que a sintaxe adequada que define aceitável será devidamente definida e documentada,

juntamente com a adição de trechos de pseudocódigos, os quais também possuirão sua sintaxe definida, possibilitará a execução do fluxograma. Abaixo seguem alguns exemplos de estruturas possíveis, bem como o resultado esperado pela execução destas:

*Figura 27 - Exemplos de estrutura de fluxogramas*



Na estrutura “a” da Figura 3 o usuário atribui valores a variáveis, realiza uma operação matemática com estas e ao final imprime o resultado na tela, -no programa o resultado de execução seria algo similar a isto:

- Saída
  - “2200.00”

Na estrutura “b” da Figura 3, a variável “nome\_variavel” receberia um valor inserido pelo usuário (presumivelmente através do teclado) e então imprimiria este valor na tela, abaixo a saída esperada:

- Saída:
  - valor inserido pelo usuário



## 5. METODOLOGIA

O trabalho aqui proposto consiste essencialmente na construção e implantação de um software, logo faz-se necessária a escolha e utilização de um processo adequado, o qual se encaixe tanto no perfil do projeto a ser desenvolvido quanto no das pessoas envolvidas neste.

Como o projeto será desenvolvido exclusivamente por uma pessoa a gama de processos de software disponíveis é extremamente reduzida, pois a maioria deles é focada no trabalho de equipes e embora possam ser realizadas adaptações para ajustar estes processos, estas são extremamente passíveis de erros e podem distorcer diretrizes básicas do processo tornando-o mais um empecilho no desenvolvimento que uma ferramenta de auxílio. Deste modo, foi selecionado o processo *Personal Extreme Programming* (PXP), um processo ágil e iterativo que parte do pressuposto de que apenas uma pessoa estará diretamente envolvida no desenvolvimento do projeto (DZHUROV. KRASTEVA. ILIEVA. 2009).

O PXP é derivado dos processos *Personal Software Process* (PSP) e *Xtreme Programming* (XP) e suas diretrizes seguem uma combinação dos preceitos desses dois processos, propondo ser uma versão mais leve do PSP e introduzindo práticas do XP apropriadas ao uso por apenas uma pessoa.

No PXP são definidas quatorze práticas de organização e desenvolvimento, sendo as trezes primeiras advindas dos processos PSP e XP. Neste projeto serão utilizadas onze dessas práticas, sendo deixadas de lado aquelas que poderiam por em risco o desenvolvimento da ferramenta proposta dentro do prazo disponível, abaixo segue uma listagem dessas e uma breve descrição sobre cada uma:

- Padrão de código;
  - Antes do início da codificação do projeto deve ser definido um padrão de código a ser seguido;
- Mensuração de tamanho;
  - É definido um método de mensuração de tamanho do código para o projeto e este método é utilizado durante o projeto;
- Revisão de código;

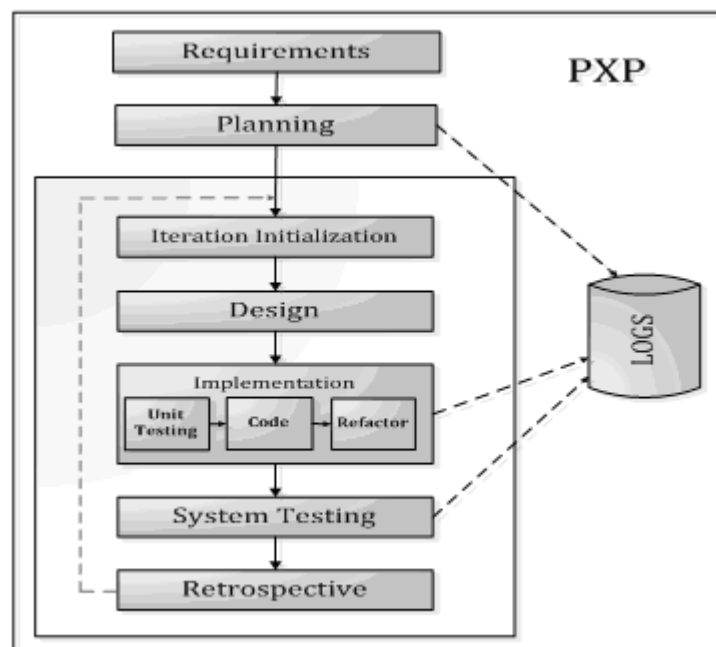
- Após completa a fase de codificação em um ciclo do processo todo esse código produzido deve ser revisto afim de que sejam encontrados erros;
- Refatoramento;
  - O processo de refatoração ocorre após a revisão do código e ele institui que quaisquer melhorias possíveis referentes a legibilidade, simplicidade e cortes de duplicações devem ser aplicadas ao código;
- Integração contínua;
  - Todo o código será integrado ao menos uma vez no dia em que for produzido, sendo que nessa integração ele será automaticamente verificado, através dos testes unitários produzidos previamente, e implantado;
- Pequenas versões;
  - A cada iteração do processo é lançada uma versão funcional do sistema contendo partes específicas dos requisitos;
- Desenvolvimento orientado a testes;
  - Antes da codificação de qualquer funcionalidade é criado um teste para esta, permitindo assim que possíveis erros sejam encontrados durante ou logo após o desenvolvimento desta;
- Registro de defeitos;
  - Qualquer defeito ou bug encontrado durante o desenvolvimento do software é registrado detalhadamente em um log;
- *Spike solutions*;
  - Previamente ao desenvolvimento de alguma funcionalidade relevante ao projeto será criado um “*spike*”, um código simples cujo único propósito é asserir que a ideia para o desenvolvimento desta funcionalidade esteja correta.
- Planejamento de atividade;
  - É realizado principalmente através da análise de relatórios de projetos passados ou iterações anteriores. Cada tarefa é designada a uma categoria (teste unitário, classe de regra de negócio, etc) e é realizada uma estimativa de tempo para que

esta seja terminada baseando-se em dados coletados de tarefas da mesma categoria em projetos ou iterações anteriores.

- Proposta de melhoramento do processo;
  - Após o fim do projeto ou de uma iteração do ciclo de vida do processo deste, é criado um documento contendo uma listagem de possíveis melhorias que podem ser feitas neste, sendo estas melhorias baseadas nos dados coletados e na observação do desenvolvimento;

Por fim, as fases do processo (em ordem de execução) são: levantamento de requisitos, planejamento, design, implementação (teste unitário, codificação, refatoração), teste do sistema e retrospectiva. As iterações ocorrem a partir da fase de design, contudo como se trata de um processo ágil as fases de levantamento de requisitos e planejamento também aceitam mudanças em etapas avançadas do projeto. Segue abaixo a Figura 4, ilustração contendo as fases em sua ordem de aplicação:

*Figura 28 - Fases do PXP*



Embora não especificados pelo processo serão também gerados os seguintes artefatos durante o desenvolvimento do projeto:

- Diagrama de caso de uso;

- Terá o propósito de definir as funcionalidades do sistema.
- Diagrama entidade relacionamento;
  - Servirá como modelo conceitual do banco de dados e base para a construção do diagrama físico.
- Diagrama físico do banco de dados;
  - Conterá as entidades, atributos e relacionamentos para implementação no banco de dados.
- Diagrama de classes;
  - Será utilizado apenas no nível de esboço, demonstrando classes e seus relacionamentos, porém sem especificar atributos e métodos.
- Protótipos de tela;
  - Serão utilizados como base para a construção das interfaces gráficas da ferramenta.
- Diagrama navegacional;
  - Será utilizado com o objetivo de ilustrar a navegação pelas páginas do sistema.

## 6. TECNOLOGIAS

Para desenvolvimento do projeto serão utilizadas as seguintes tecnologias:

- PHP: linguagem interpretada, “*server-side*” e voltada para desenvolvimento web geral. Será utilizada, em conjunto com as linguagens HTML, CSS e Javascript, para fornecer as interfaces que irão interagir com o usuário;

- Javascript: linguagem interpretada, “*client-side*”, utilizada principalmente para interação assíncrona com o usuário em aplicativos web. Será a linguagem que possibilitará a construção da interface para manipulação e execução dos fluxogramas;

- SQL: Linguagem utilizada para manipulação de dados contidos em um banco de dados relacional. Seu uso possibilitará a persistência de dados do usuário (cadastro, fluxogramas, etc).

Em vista da alta complexidade envolvida na manipulação de elementos dinâmicos em softwares web será utilizada uma biblioteca de Javascript que forneça uma API para manipulação de diagramas. Entre as existentes, as que inicialmente se mostram mais apropriadas são:

- jsPlumb: <http://jsplumbtoolkit.com>
- JointJS: <http://www.jointjs.com>

## 7. CRONOGRAMA

Tabela 5 - Cronograma

	Jul	Ago	Setem	Outu	Novem	Dezem	Jan	Feve
Consulta bibliográfica	■	■	■	■	■	■	■	■
Construção da monografia						■	■	■
Levantamento de requisitos	■	■						
Planejamento			■					
1ª iteração			■	■				
2ª iteração				■	■			
3ª iteração						■		
4ª iteração						■	■	
5ª iteração							■	■

Segue abaixo especificação das iterações definidas na Tabela 2:

- 1ª Iteração: manter fluxograma;
- 2ª Iteração: executar fluxograma;
- 3ª Iteração: retornar estado de fluxograma, avançar estado de fluxograma;
- 4ª Iteração: tornar fluxograma público, buscar fluxograma, exportar imagem de fluxograma, compartilhar imagem de fluxograma em rede social;
- 5ª Iteração: realizar cadastro, realizar login, recuperar senha, exibir relatório.

## 8. CONSIDERAÇÕES FINAIS

A manipulação de diagramas dinâmicos e a interpretação destes em um ambiente web estão longe de constituir tarefas triviais. Somando-se isso ao tempo limitado e a restrição de apenas uma pessoa responsável pelo projeto pode-se dizer que a construção da ferramenta aqui proposta representará um grande desafio. Contudo pode-se antever que o esforço necessário não será em vão, pois a ferramenta almeja trazer um benefício real, auxiliando quaisquer interessados tanto no ensino quanto no aprendizado de algoritmos e lógica de programação. Seu uso possibilitará uma oportunidade de talvez engajar uma pessoa ou aluno que de outro modo poderia se desinteressar por esses assuntos devido ao tortuoso caminho do aprendizado inicial, ou ainda poderá facilitar o aprendizado de quaisquer outros, sendo desse modo uma grande possibilidade de um recurso educacional de valor.

## 9. REFERÊNCIAS

CHAUDHURI, Anil Bikas. **The art of programming Through Flowcharts & Algorithms**. 1.ed. Laxmi Publications. pág 1-32. 2005.

CORMEN, Thomas H (at al). **Introduction to algorithms**. 3.ed. EUA. The MIT Press. pág 5-14. 2009.

DZHUROV, Yani. KRASTEVA, Iva. ILIEVA, Sylvia. **Personal Extreme Programming – An Agile Process for Autonomous Developers**. Demetra EOOD. 2009.

HUMPREY, Watts S. **The Personal Software Process (PSP)**. Software Engineering Institute. 2000.

MANZANO, José Augusto Navarro Garcia. **Revisão e Discussão da Norma ISO 5807 – 1985 (E) Proposta para Padronização Formal da Representação Gráfica da Linha de Raciocínio Lógico Utilizada no Desenvolvimento da Programação de Computadores a ser Definida no Brasil**. THESIS, São Paulo, ano I, v. 1, p. 1-31, 1o Semestre. 2004.

SOMMERVILLE, Ian. **Engenharia de Software**. 9 ed. São Paulo. Person Prentice Hall. pág 57-68. 2011.