

UNIVERSIDADE PAULISTA

WESLEY RODRIGUES DA SILVA

PARADIGMAS DA ENGENHARIA DE SOFTWARE:

Desenvolvimento de um framework para redes neurais artificiais aplicado
como ferramenta auxiliar no diagnóstico do autismo

SÃO PAULO

2014

WESLEY RODRIGUES DA SILVA

PARADIGMAS DA ENGENHARIA DE SOFTWARE:

Desenvolvimento de um framework para redes neurais artificiais aplicado
como ferramenta auxiliar no diagnóstico do autismo

Trabalho apresentado à disciplina Trabalho de Curso,
do curso de pós-graduação em Engenharia de
Software da Universidade Paulista - UNIP

Orientador: Prof. Dr. Marcelo Nogueira

SÃO PAULO

2014

WESLEY RODRIGUES DA SILVA

PARADIGMAS DA ENGENHARIA DE SOFTWARE:

Desenvolvimento de um framework para redes neurais artificiais aplicado
como ferramenta auxiliar no diagnóstico do autismo

Trabalho apresentado à disciplina Trabalho de Curso,
do curso de pós-graduação em Engenharia de
Software da Universidade Paulista - UNIP

Orientador: Prof. Dr. Marcelo Nogueira

Aprovado em:

BANCA EXAMINADORA

Prof.	_____	/	_____	/	_____
	Universidade Paulista - UNIP				
Prof.	_____	/	_____	/	_____
	Universidade Paulista - UNIP				
Prof.	_____	/	_____	/	_____
	Universidade Paulista – UNIP				

Paradigmas da engenharia de software: Desenvolvimento de um framework para redes neurais artificiais aplicado como ferramenta auxiliar no diagnóstico do autismo / Wesley Rodrigues da Silva – São Paulo, 2014.

98 f. il. Color

Trabalho de Conclusão de Curso (pós-graduação) – Apresentado ao Instituto de Ciências Exatas da Universidade Paulista, São Paulo, 2014.

Área de Concentração: Inteligência Artificial

Orientação: Prof. Marcelo Nogueira

1. Redes Neurais Artificiais. 2. Perceptron Multicamadas. 3. Sistemas Classificadores. 4. Diagnóstico de Autismo. I. Da Silva, Wesley Rodrigues.

AGRADECIMENTOS

Agradeço ao amigo-irmão Alan Drummond por ter me inspirado, e mais que isso, me mostrado que era possível se dedicar a uma causa. E assim, deixei um emprego seguro para cursar pós-graduação em engenharia de software, priorizando meus estudos. Fez toda a diferença.

Agradeço à minha esposa Aline, pelo apoio incondicional e pela motivação nos momentos difíceis, me mantendo sempre no caminho correto. Sempre criticando meu trabalho de forma construtiva, e sempre se esforçando para entender conteúdos que nem eram de sua área de atuação. Como sempre, foi uma companheira e amiga.

Agradeço especialmente ao professor Dr. Marcelo Nogueira, meu orientador, por ter rompido a barreira que existe entre professores e alunos, sendo um amigo e transmitindo conselhos de sua vasta experiência como profissional de tecnologia e educação. Isso não tem preço, e este conhecimento não se adquire em livros.

Agradeço também aos professores Fábio Luiz, Amanda Luíza Pereira, Expedito Júnior e Emerson Facunte, que foram simplesmente essenciais e especiais, e a todos os professores e colegas de classe da UNIP com quem dividi momentos preciosos de discussão e reflexão.

EPÍGRAFE

“O homem não é nada além daquilo que a educação faz dele.”

– Immanuel Kant

RESUMO

Este trabalho foi realizado para demonstrar que utilizando a Engenharia de Software, é possível realizar tarefas complexas, e o exemplo aqui utilizado é a construção de um framework que facilita a utilização de redes neurais artificiais como classificador de dados. Utilizando os paradigmas da Engenharia de Software somados à informações obtidas em estudos de medicina e psiquiatria, esta rede neural foi construída e aplicada a um questionário de avaliação de doenças do espectro autista. Após treinar a rede, ela foi capaz de reconhecer as classes de padrões esperadas. O método se mostrou funcional, e apesar de não substituir o médico, mostrou que é possível tornar mais acessível uma área de estudo complexa como a inteligência artificial.

Palavras-chave: Redes Neurais Artificiais, Perceptron Multicamadas, Sistemas Classificadores, Diagnóstico de Autismo.

ABSTRACT

This study has been conducted to demonstrate that using Software Engineering principles it is possible to perform complex tasks, and the example used here is the building of a framework that makes it easy to utilize the perceptron artificial neural networks as data classifiers. Using the paradigms of software engineering and information obtained through medicine and psychiatry studies, this neural network has been built and applied to a questionnaire, this way detecting autism spectrum disorders. After training the network, it was able to recognize the patterns classes expected. The method proved to be functional, and despite not replace the doctor, showed that it is possible to make it more accessible complex areas of study, like artificial intelligence.

Keywords: Artificial Neural Networks, Multilayer Perceptron, Classifier Systems, Autism Diagnosis.

SUMÁRIO

Agradecimentos.....	5
Epígrafe.....	6
Resumo.....	7
Abstract.....	8
Lista de abreviações.....	10
Lista de figuras.....	11
Lista de fórmulas.....	12
Introdução.....	13
Considerações Iniciais.....	13
Motivação e Justificativa.....	15
Objetivos.....	15
Metodologia da Pesquisa.....	16
Estrutura do Trabalho.....	16
Considerações finais do capítulo.....	17
Capítulo 1: Os paradigmas da engenharia de software.....	19
Capítulo 2: Redes neurais artificiais.....	21
Capítulo 3: O modelo perceptron multicamadas.....	26
Capítulo 4: Planejamento do Framework.....	29
Conclusão da Parte I.....	45
Capítulo 5: Características do autismo.....	47
Capítulo 6: Diagnóstico da doença.....	50
Capítulo 7: O questionário de diagnóstico.....	52
Capítulo 8: Aplicação da RNA.....	53
Conclusão da Parte II.....	56
Capítulo 9: Resultados da pesquisa.....	58
Capítulo 10: Possíveis aplicações do framework.....	59
Capítulo 11: Confrontamento de tecnologias (Lógicas Fuzzy e Paraconsistente).....	60
Referências.....	96
Leitura Adicional Utilizada.....	98

LISTA DE ABREVIACÕES

ADDM – Rede de Monitoramento de Autismo e Transtornos do Desenvolvimento

CDC – Centro de Controle e Prevenção de Doenças

CID – Classificação Internacional de Doenças e de Problemas Relacionados a Saúde

DSM-IV – Manual Diagnóstico e Estatístico de Transtornos Mentais

GPL – Licença Pública GNU

HTTP – Protocolo de Transmissão de Hipertexto

IA – Inteligência Artificial

PMC – Perceptron Multi Camadas

RNA – Rede Neural Artificial

TEA – Transtornos do Espectro Autistas

XOR – OU Exclusivo

LISTA DE FIGURAS

Figura 1: Trecho de código em linguagem Assembly. Fonte: Autor.....	13
Figura 2: Trecho de código em linguagem Ruby. Fonte: Autor.....	14
Figura 3: Representação do neurônio biológico. Fonte: (SILVA; SPATTI; FLAUZINO, 2010).....	22
Figura 4: Representação do neurônio artificial. Fonte: (SILVA; SPATTI; FLAUZINO, 2010).....	22
Figura 5: Separabilidade de problemas. Fonte: (COPPIN, 2012).....	25
Figura 6: Rede perceptron 3 camadas adiante. Fonte: (COPPIN, 2012).....	26
Figura 7: Saída do comando git log. Fonte: Autor.....	30
Figura 8: Mapa mental organizando as ideias. Fonte: Autor.....	31
Figura 9: Diagrama de casos de uso. Fonte: Autor.....	31
Figura 10: Diagrama de atividades. Fonte: Autor.....	32
Figura 11: Diagrama de sequência. Fonte: Autor.....	33
Figura 12: Esboço do Diagrama de classes. Fonte: Autor.....	34
Figura 13: Fração do backlog de atividades. Fonte: Autor.....	35
Figura 14: Gráfico Gantt. Fonte: Autor.....	36
Figura 15: Código para execução de testes com tabela XOR. Fonte: Autor.....	37
Figura 16: Execução do framework em linha de comando. Fonte: Autor.....	38
Figura 17: Inicialização do serviço HTTP. Fonte: Autor.....	39
Figura 18: Tela inicial de criação de rede. Fonte: Autor.....	39
Figura 19: Inserção dos dados de treinamento. Fonte: Autor.....	39
Figura 20: Rede treinada e pronta para execução. Fonte: Autor.....	40
Figura 21: Execução de 0,0 na tabela XOR. Fonte: Autor.....	41
Figura 22: Resposta correta para 0,0. Fonte: Autor.....	42
Figura 23: Execução de 1,0 na tabela XOR. Fonte: Autor.....	43
Figura 24: Resposta correta para 1,0. Fonte: Autor.....	44
Figura 25: Página com formulário HTML do questionário. Fonte: Autor.....	54
Figura 26: Resposta da rede versus resposta da média aritmética. Fonte: Autor.....	55
Figura 27: Reticulado representativo da Lógica Paraconsistente Anotada. Fonte: (INACIO,2014).....	60
Figura 28 - Seleção de Software. Fonte: Reprodução de tela.....	76
Figura 29 - Instalação do Papyrus. Fonte: Reprodução de tela.....	77
Figura 30 - Diagrama de Casos de Uso. Fonte: Reprodução de tela.....	78
Figura 31 - Processador de Texto Writer. Fonte: Reprodução de tela.....	79
Figura 32 - Editor Gráfico Draw. Fonte: Reprodução de tela.....	79
Figura 33 - Página de projeto no GitHub. Fonte: Reprodução de tela.....	80
Figura 34 - Diagramas suportados. Fonte: Reprodução de tela.....	82
Figura 35 - Mantis exibindo uma lista de bugs. Fonte: Reprodução de tela.....	83
Figura 36 - Gráfico Gantt no Planner. Fonte: Reprodução de tela.....	84
Figura 37: Mapa Mental no VYM. Fonte: Reprodução de tela.....	84
Figura 38 - Teste Unitário com JUnit. Fonte: Reprodução de tela.....	86

LISTA DE FÓRMULAS

Fórmula 1: Fórmula matemática do neurônio artificial. Fonte: (COPPIN, 2012).....	23
Fórmula 2: Função de ativação linear. Fonte: (COPPIN, 2012).....	23
Fórmula 3: Cálculo do erro. Fonte: (COPPIN, 2012).....	24
Fórmula 4: Atualização dos Pesos Sinápticos. Fonte: (COPPIN, 2012).....	24
Fórmula 5: Cálculo do gradiente de erro na retropropagação. Fonte: (COPPIN, 2012).....	27
Fórmula 6: Função Hiperbólica. Fonte: (COPPIN, 2012).....	28
Fórmula 7: Derivada da Função Hiperbólica. Fonte: (COPPIN, 2012).....	28
Fórmula 8: Função Logística. Fonte: (COPPIN, 2012).....	28
Fórmula 9: Derivada da Função Logística. Fonte: (COPPIN, 2012).....	28

INTRODUÇÃO

Neste capítulo, é detalhada a estrutura desta pesquisa, fazendo uma introdução ao assunto e apresentando o problema, as motivações, justificativas e os métodos científicos utilizados.

CONSIDERAÇÕES INICIAIS

Desenvolver software atualmente é uma tarefa muito mais acessível que nas décadas que se seguiram após 1940. Esse período, que durou cerca de 20 anos, ficou conhecido como a época de ouro dos programadores em linguagem de montagem — o Assembly, que utiliza instruções de baixíssimo nível, compreendidas pelo processador (BROWN, 2014).

```

1ad39:    a6             cmpsb  %es:(%di),%ds:(%si)
1ad3a:    61             popa
1ad3b:    00 00          add   %al,(%bx,%si)
1ad3d:    00 00          add   %al,(%bx,%si)
1ad3f:    00 f4          add   %dh,%ah
1ad41:    a5             movsl  %ds:(%si),%es:(%di)
1ad42:    01 00          add   %eax,(%bx,%si)
1ad44:    00 00          add   %al,(%bx,%si)
1ad46:    00 00          add   %al,(%bx,%si)
1ad48:    40             inc   %eax
1ad49:    0d 00 00 00 00 or     $0x0,%eax

1ad56:    00 00          add   %al,(%bx,%si)
1ad58:    20 00          and   %al,(%bx,%si)

1ad66:    00 00          add   %al,(%bx,%si)
1ad68:    01 00          add   %eax,(%bx,%si)
1ad6a:    00 00          add   %al,(%bx,%si)
1ad6c:    03 00          add   (%bx,%si),%eax

1ad7e:    00 00          add   %al,(%bx,%si)
1ad80:    f4             hlt
1ad81:    a5             movsl  %ds:(%si),%es:(%di)
1ad82:    01 00          add   %eax,(%bx,%si)
1ad84:    00 00          add   %al,(%bx,%si)
1ad86:    00 00          add   %al,(%bx,%si)
1ad88:    ef             out    %eax,(%dx)

1ad95:    00 00          add   %al,(%bx,%si)
1ad97:    00 01          add   %al,(%bx,%di)

```

Figura 1: Trecho de código em linguagem Assembly. Fonte: Autor

Com o passar dos anos, surgiram linguagens de programação cada vez mais intuitivas e parecidas com linguagem natural, e pessoas sem conhecimentos avançados em computação tornaram-se capazes de desenvolver aplicativos de baixa complexidade.

```

encoding: utf-8
puts "Digite seu peso: "
peso = gets.to_f
puts "Digite sua altura: "
altura = gets.to_f
imc = peso / (altura * altura)
situacao = case imc
  when 0..18.4 then "abaixo do peso ideal"
  when 18.5..24.9 then "ideal"
  when 25..25.9 then "sobrepeso"
  when 26..60 then "obesidade"
end
puts "Seu IMC é #{imc} e sua situação é: #{situacao}"

```

Figura 2: Trecho de código em linguagem Ruby. Fonte: Autor

Porém, quando o tamanho e a complexidade destes aplicativos aumentam, as dificuldades se tornam grandes obstáculos. Empresários que desenvolvem software como atividade comercial sabem o quanto é complicado manter uma equipe grande trabalhando em um mesmo projeto de modo harmonioso e integrado. Estas dificuldades para concluir os projetos de software existem há mais de 50 anos, e alguns estudiosos da área definem este fenômeno como “Crise do Software” (NOGUEIRA, 2009).

A Engenharia de Software vem para organizar o desenvolvimento de software, através de métodos, processos e ferramentas, que quando utilizados da forma correta, garantem a entrega do produto final dentro do prazo, com um alto grau de qualidade (PRESSMAN, 2011).

Dentro da Engenharia de Software há 10 paradigmas, que são considerados fatores críticos de sucesso quando adotados sistematicamente no projeto (NOGUEIRA; ABE, 2010). Segundo Nogueira e Abe (2010), estes paradigmas são:

1. Engenharia de requisitos;
2. Gestão de Configuração;
3. Gestão de Riscos;
4. Modelagem Visual;
5. Metodologias de Desenvolvimento;
6. Normas e Modelos;
7. Métricas;
8. Cronogramação;
9. Implementação e

10. Testes

Em paralelo aos avanços obtidos na área de Engenharia de Software, uma outra área de estudos ganhou notoriedade na última metade do século XX: A Inteligência Artificial (IA).

A IA é um ramo dos estudos em computação que busca produzir software com inteligência similar a humana, entre outros objetivos. Como exemplo de aplicação da IA, pode-se citar as Redes Neurais Artificiais (RNAs). As RNAs são modelos teóricos que simulam o funcionamento das redes de neurônios encontradas no cérebro. São amplamente utilizadas para reconhecimento de padrões e classificadores, e a escolha do melhor paradigma, tipo de lógica e algoritmo de treinamento não são atividades triviais, exigindo conhecimentos específicos sobre o funcionamento destas redes (SILVA; SPATTI; FLAUZINO, 2010).

A IA é uma área ampla, onde os estudos geralmente requerem sólidas bases teóricas em matemática, algoritmos e lógica (COPPIN, 2012).

MOTIVAÇÃO E JUSTIFICATIVA

Em suma, a motivação deste trabalho é mostrar que com processos definidos e organizados, com ferramentas e com a metodologia certa, é possível realizar feitos complexos, e enfim tornar mais acessível e desmistificada uma tecnologia que para muitos ainda parece obscura.

Este trabalho foi idealizado a partir de uma necessidade: diante de um problema que poderia ser resolvido com o uso de IA, percebeu-se que existe uma carência de material completo sobre o assunto. Apesar de encontrar uma infinidade de artigos e teses relatando experiências com aplicação de redes neurais artificiais, a grande maioria destes materiais não possuía o código fonte ou fornecia uma explicação completa sobre como reproduzir os resultados. Por isso, há uma preocupação em tornar o framework resultante deste trabalho disponível para utilização, garantindo aos interessados nos estudos em IA maior acessibilidade.

OBJETIVOS

O objetivo geral deste trabalho é demonstrar que mesmo atividades que não são triviais e simples, como aplicação prática de IA, por exemplo, podem ser implementadas e aplicadas na prática, se apoiando no uso dos paradigmas da Engenharia de Software.

Os objetivos específicos são:

- Produzir um framework que possa ser reutilizado facilmente para outras atividades de classificação, fornecendo de modo rápido uma rede neural artificial multicamadas pronta para ser treinada e utilizada.
- Demonstrar uma aplicação prática e útil para este framework, que neste trabalho será uma rede neural artificial conectada à saída de um questionário para diagnóstico de padrões de autismo. Não é um objetivo deste trabalho entrar no mérito de pesquisa sobre o autismo ou seus métodos de diagnóstico; o foco é apenas demonstrar que o framework pode ser aplicado à classificação de dados de diversas fontes.

METODOLOGIA DA PESQUISA

Para elaborar o framework e aplicá-lo no questionário, o seguinte roteiro foi seguido fielmente:

1. Estudo profundo sobre Engenharia de Software e seus paradigmas;
2. Estudo sobre metodologia científica;
3. Estudo aprofundado sobre Inteligência Artificial, com foco nas redes neurais artificiais do tipo perceptron multicamadas;
4. Estudo sobre autismo e métodos de diagnósticos;
5. Estudo sobre a linguagem de programação Ruby;
6. Elaboração das documentações;
7. Criação do MLP Framework;
8. Criação do questionário adaptado à rede neural artificial;
9. Treinamento da rede;
10. Medição e análise dos resultados;

ESTRUTURA DO TRABALHO

Este trabalho é dividido em três partes principais:

- A Parte 1 descreve a criação do framework.

- A Parte 2 trata da aplicação prática do framework.
- A Parte 3 mostra as conclusões obtidas com a pesquisa, possíveis aplicações e abre caminho para pesquisas futuras.

CONSIDERAÇÕES FINAIS DO CAPÍTULO

Espera-se que com a aplicação dos estudos feitos neste trabalho, os Paradigmas da Engenharia de Software sejam adotados por mais pessoas, e o acesso às tecnologias como as redes neurais artificiais se popularize.

PART E 1 – CONSTRUÇÃO DO FRAMEWORK

CAPÍTULO 1: OS PARADIGMAS DA ENGENHARIA DE SOFTWARE

A Engenharia de Software é um campo de estudo muito amplo, mas este estudo se limitou a utilizar os 10 paradigmas recomendados por Nogueira e Abe (2010). São eles:

Engenharia de Requisitos: “O processo de descobrir, analisar, documentar, e verificar as funções e restrições do sistema, é chamado de engenharia de requisitos” (SOMMERVILLE, 2011).

Gerenciamento de Configuração: Sommerville (2011) define o gerenciamento de configuração como o processo de gerenciar produtos de sistema durante a fase de desenvolvimento, ou seja, controlar as versões, mudanças, correção de defeitos, adição de recursos e outras alterações incluídas no código ou documentação do software.

Gestão de Riscos: Nogueira (2009) define o gerenciamento de riscos como um processo sistemático, objetivando minimizar os efeitos dos riscos através do tratamento dos fatores causadores, conseguindo com isso um produto de software com alto grau de qualidade, dentro do prazo e custo estimados.

Modelagem Visual: Software é abstrato e intangível. Por esta razão, recursos gráficos que expliquem de forma clara como o software funciona, como as partes estão relacionadas, incrementam as chances de se obter sucesso no projeto de software. Como Nogueira e Abe (2010) explicam, modelagem visual é o uso de notações gráficas para capturar o design do software. A UML é uma linguagem pensada para esta finalidade, obedecendo às melhores práticas da Engenharia de Software (LIMA, 2013).

Metodologias de Desenvolvimento: As metodologias são conjuntos muito bem estruturados de práticas que servem como guia durante o processo de desenvolvimento de software. Existem várias abordagens diferentes, cada uma com várias metodologias distintas. O RUP, o SCRUM, o XP e o ALM são algumas dessas metodologias (NOGUEIRA; ABE, 2010).

Normas e Modelos: As normas e modelos de software são documentos que fornecem diretrizes baseadas nas melhores práticas de Engenharia de Software, e que podem ser auditadas de modo que o nível de conformidade com determinada norma seja medido. Existem várias normas na área de Engenharia de Software, entre as quais podem ser citadas a CMMI, a NBR ISO/IEC 12207, ISO/IEC 15504 (NOGUEIRA; ABE, 2010).

Métricas: Segundo Nogueira e Abe (2010), "a adoção da prática de medição permite ao estimar de forma mais racional todo o processo de desenvolvimento de software.[...] As principais

métricas são: FPA – Pontos por Função, COCOMO – Constructive Cost Model, KLOC – Lines of Code e UCP – Pontos por Caso de Uso."

Cronogramação: Trata das características que podem influenciar na data de entrega do software, atrasando a finalização do projeto (NOGUEIRA; ABE, 2010).

Implementação: É a maneira como um software é implementado, e depende diretamente de como as diretrizes são seguidas: se as especificações foram seguidas, se as melhores práticas de programação orientada a objetos foram utilizadas, se o software foi testado, se os sistemas estão integrados, se foi usada gerência de configuração, etc (NOGUEIRA; ABE, 2010).

Testes: Os testes de software garantirão que o produto final tenha a maioria dos defeitos mitigados, aumentando o grau de qualidade e de conformidade com os requisitos (PRESSMAN, 2011).

CAPÍTULO 2: REDES NEURAIS ARTIFICIAIS

As RNAs (redes neurais artificiais) são uma das áreas de aplicação da IA. Entretanto, antes de explicar os detalhes das RNAs, é necessário falar sobre a parte do corpo humano que elas buscam imitar: as redes neuronais do cérebro.

O cérebro humano é o objeto mais complexo conhecido em todo o universo. Dentro dele, os neurônios criam conexões, que são responsáveis por transmitir os impulsos elétricos que carregam os pensamentos. Com cerca de 100 bilhões de neurônios formando cerca de 100 trilhões de conexões, suas capacidades computacionais exigiriam o equivalente a 4 usinas de Itaipu para alimentar um computador com as tecnologias atuais e processamento equivalente, caso este computador existisse (DE SANTI, 2012). E este é um processo extremamente caro, por conta do hardware, da mão de obra altamente especializada e das dificuldades em simular comportamentos que não são totalmente conhecidos. Um estudo nesta linha de pesquisa foi realizado pela Stanford University, que tenta criar um supercomputador com apenas 1 milhão de neurônios artificiais, com o projeto Brain Simulator (STANFORD UNIVERSITY, 2006).

O neurônio biológico é a unidade básica do sistema nervoso. É composto basicamente por:

- um corpo celular (também conhecido como soma), onde fica o núcleo da célula;
- dendritos, que são prolongamentos com ramos que recebem os sinais;
- axônio, que só possui ramificações na extremidade e conduz os sinais;
- sinapses, responsáveis por transmitir as informações de uma célula para outra célula;
- Os outros componentes do neurônio não tem importância significativa para este estudo.

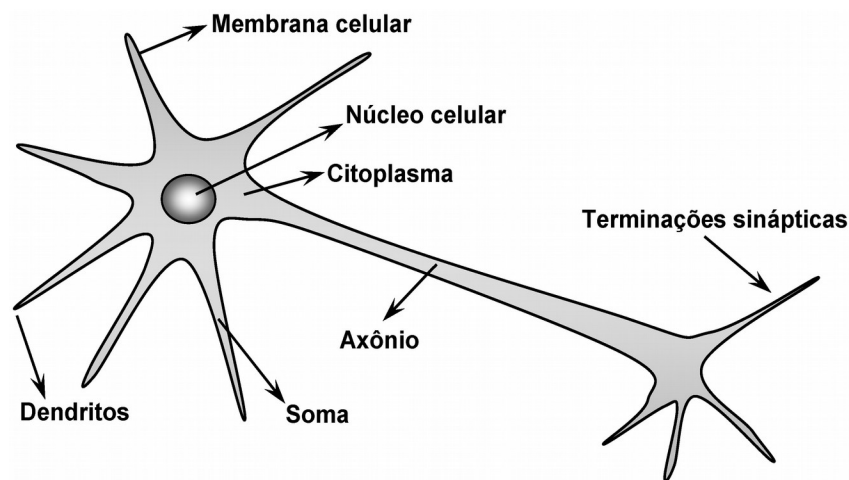


Figura 3: Representação do neurônio biológico. Fonte: (SILVA; SPATTI; FLAUZINO, 2010)

O neurônio recebe sinais de outros neurônios em seus dendritos, e caso estes sinais excedam um limiar, ocorre uma reação química que resulta em um pulso elétrico, conhecido como potencial de ação. Este pulso segue pelo axônio e é direcionado aos neurônios conectados no final dessa estrutura através das terminações sinápticas (COPPIN, 2012).

Inspirado no modo de funcionamento do neurônio biológico, McCulloch e Pitts criaram em 1943 um neurônio artificial.

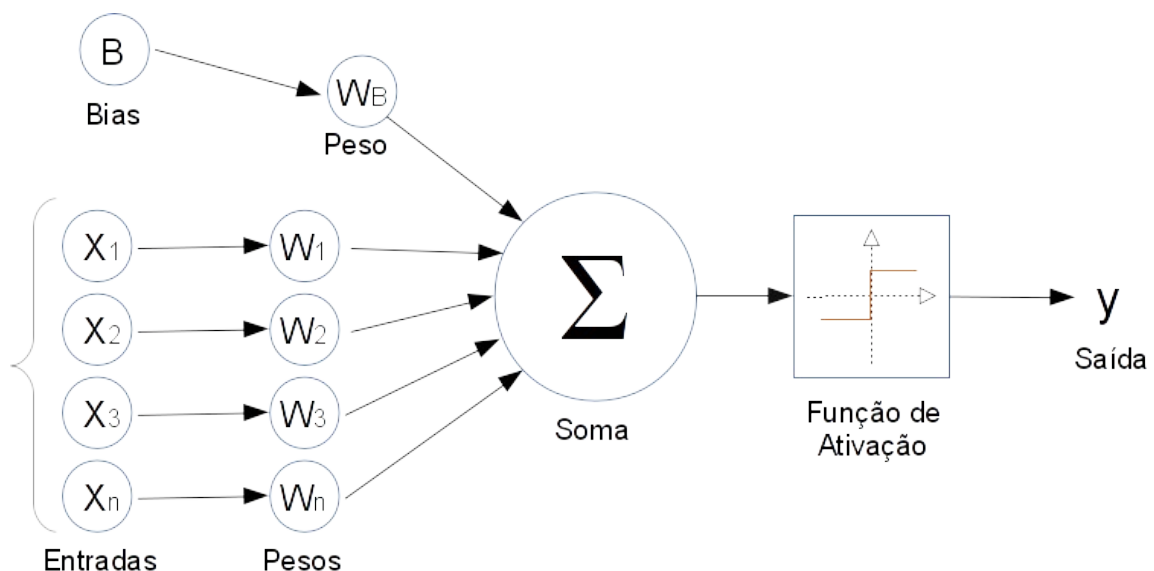


Figura 4: Representação do neurônio artificial. Fonte: (SILVA; SPATTI; FLAUZINO, 2010)

Cada neurônio artificial de McCulloch e Pitts tem uma série de entradas, onde são injetados valores. Além das entradas comuns, há uma entrada de viés, conhecida como bias. Bias é um valor

que contribui para a tendência dos resultados. Estes valores são multiplicados por pesos, somados e testados por uma função de ativação, que possui uma saída única. Quando a função linear é utilizada, se a soma das entradas multiplicadas por seus pesos for maior que o limiar, então diz-se que este neurônio foi ativado, e sua saída é +1. Caso contrário, o neurônio não foi ativado, e sua saída é 0.

$$X = \sum_{i=1}^n x_i \times w_i$$

Fórmula 1: Fórmula matemática do neurônio artificial. Fonte: (COPPIN, 2012).

A saída do processamento feito pela fórmula 1 é avaliada por uma função de ativação, que determinará a saída do perceptron. Este modelo de neurônio utiliza a função de ativação linear (COPPIN, 2012).

$$Y = \begin{cases} +1 & \text{se } X > \text{limiar} \\ 0 & \text{se } X \leq \text{limiar} \end{cases}$$

Fórmula 2: Função de ativação linear. Fonte: (COPPIN, 2012).

Inicialmente, é provável que o perceptron não encontre o resultado desejado. Para que isso aconteça, é necessário que ele passe por um processo de treinamento, chamado treinamento supervisionado. O treinamento de um perceptron consiste em fazer alterações nos pesos das entradas sinápticas, de modo que o resultado final se aproxime do resultado desejado. Seu algoritmo possui os seguintes passos:

1. Inicializar os pesos das entradas com valores aleatórios pequenos.
2. Apresentar um vetor de entradas ao neurônio.
3. Comparar a saída obtida com a saída desejada, calculando assim o erro.
4. Utilizar a fórmula de atualização de pesos para corrigir os pesos das entradas.
5. Executar novamente a partir do passo 2 até que a rede não apresente mais erros.

Para calcular o erro, utiliza-se a fórmula:

$$e_n = d_n - y_n$$

Fórmula 3: Cálculo do erro. Fonte: (COPPIN, 2012).

Onde:

e = erro

d = valor desejado

y = valor obtido na saída do neurônio

A seguir, a fórmula utilizada para atualização dos pesos sinápticos:

$$W_{\text{novo}} = W_n + Tx\text{Aprendizagem} \times \text{Erro} \times X_n$$

Fórmula 4: Atualização dos Pesos Sinápticos. Fonte: (COPPIN, 2012).

Onde:

W_{novo} = Peso ajustado

W_n = Peso atual

$Tx\text{Aprendizagem}$ = Constante entre 0 e 1 que define a velocidade dos ajustes.

Erro = Erro calculado na saída do neurônio

X_n = Valor inserido na entrada onde o peso está sendo ajustado.

Com esta configuração, o neurônio artificial é capaz de resolver uma gama de problemas matemáticos através do ajuste dos pesos. Entretanto, há problemas que não podem ser resolvidos com um único neurônio: como exemplo, podemos citar os problemas que não são linearmente separáveis. Para estes casos, o neurônio precisa ser associado com outros neurônios, criando uma RNA.

A figura a seguir representa os dois tipos de problemas:
O problema a é linearmente separável, ou seja, é possível separar as classes de objetos encontradas

no problema com apenas 1 reta. Entretanto, b não é linearmente separável.

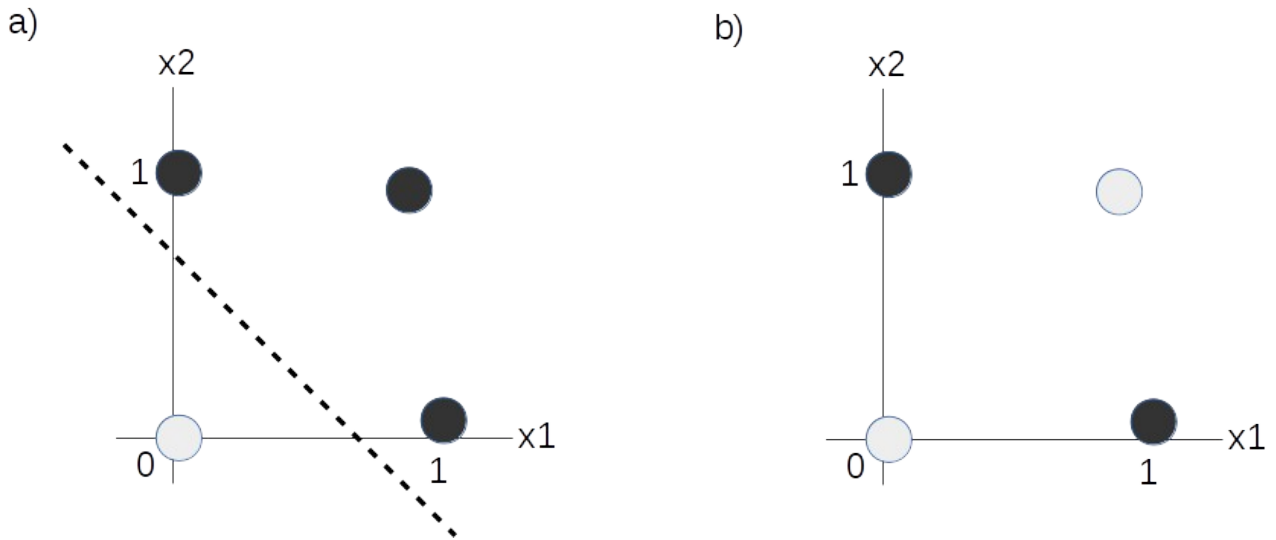


Figura 5: Separabilidade de problemas. Fonte: (COPPIN, 2012)

Para aumentar o poder do perceptron, é preciso conectá-lo a outros perceptrons, formando uma rede neural artificial perceptron multicamadas, capaz de lidar até mesmo com problemas não-linearmente separáveis (COPPIN, 2012)(RUSSEL; NORVIG, 2004).

CAPÍTULO 3: O MODELO PERCEPTRON MULTICAMADAS

Um perceptron isolado é considerado um perceptron de uma única camada. Interligar a saída de um perceptron nas entradas de perceptrons à frente é o modo prático de criar camadas. Com mais de uma camada, os perceptrons são capazes de aprender funções bem mais complexas que as linearmente separáveis. A figura a seguir demonstra um perceptron multicamadas, configurado em 3 camadas com alimentação adiante. A primeira camada, que recebe os dados, é chamada de camada de entrada. A camada final, que fornece os resultados, é chamada de camada de saída. Todas as camadas entre a de entrada e a de saída são conhecidas como camadas ocultas (RUSSEL; NORVIG, 2004).

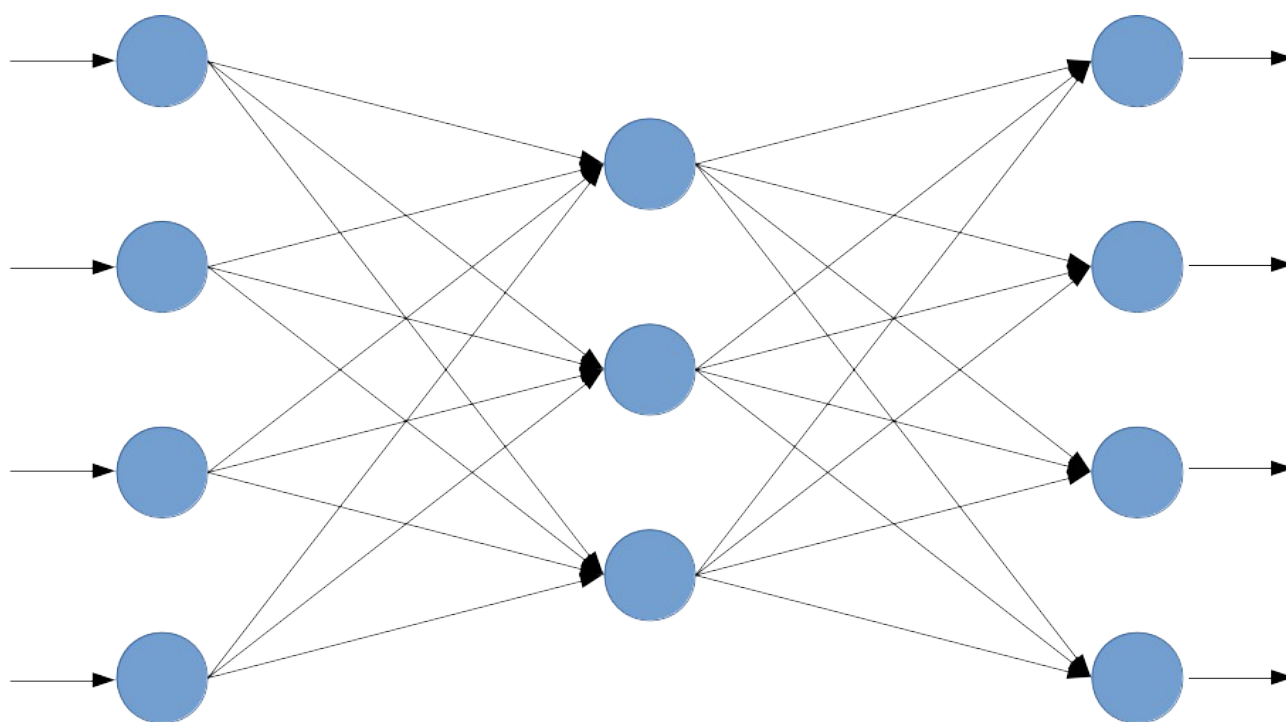


Figura 6: Rede perceptron 3 camadas adiante. Fonte: (COPPIN, 2012)

Conforme o tamanho da rede cresce, cresce também a dificuldade no processo de treinamento. O ajuste dos pesos em um único neurônio afeta toda a rede, já que os neurônios estão interligados. Deste modo, novas abordagens precisam ser utilizadas no momento de ajustar os pesos das conexões. A abordagem mais utilizada no treinamento do perceptron é a retropropagação (backpropagation), que é uma generalização da regra delta, utilizada em outros tipos de redes neurais artificiais como a Adaline, que foge do escopo deste trabalho (RUSSEL; NORVIG, 2004).

O algoritmo de retropropagação se assemelha ao algoritmo de treinamento do perceptron simples, porém o cálculo é feito para todos os neurônios, iterando entre as camadas. Passo a passo, o algoritmo de retropropagação funciona da seguinte forma:

1. Os valores de entrada são inseridos;
2. Verifica-se se os valores da saída são os esperados;
3. Caso não sejam, calcula-se o erro para cada saída;
4. Através do erro obtido, ajustar os pesos de cada neurônio da camada de saída;
5. Calcular o gradiente do erro para a camada anterior;
6. Ajustar o peso da camada anterior;
7. Calcular o gradiente do erro para a camada anterior;
8. Ajustar o peso da camada anterior;
9. Continuar assim até que todas as camadas tenham seus pesos ajustados.

O diferencial deste algoritmo é a capacidade de calcular a parcela de erro de cada neurônio, de modo que cada peso seja ajustado de acordo com sua parcela de participação no resultado final. O erro calculado na camada de saída é propagado inversamente até a camada de entrada, por isso o algoritmo recebe o nome “retropropagação”.

Para calcular o erro propagado da camada de saída para a camada intermediária, usa-se a fórmula:

$$Erro_i = Erro_s \times y' \times W_{is}$$

Fórmula 5: Cálculo do gradiente de erro na retropropagação. Fonte: (COPPIN, 2012).

Onde:

$Erro_i$ = Erro a ser propagado.

$Erro_s$ = Erro na saída.

y' = Derivada da função de ativação.

W_{is} = Peso entre os neurônios nas camadas intermediária e de saída.

Para que o algoritmo funcione, é necessário que a função de ativação seja derivável. As funções mais utilizadas são a tangente hiperbólica e a função logística. Abaixo as funções e suas derivadas.

$$f(x) = \frac{2}{1 + e^{-2x}}$$

Fórmula 6: Função Hiperbólica. Fonte: (COPPIN, 2012).

$$f'(x) = 1 - f(x)^2$$

Fórmula 7: Derivada da Função Hiperbólica. Fonte: (COPPIN, 2012).

$$f(x) = \frac{1}{1 + e^{-x}}$$

Fórmula 8: Função Logística. Fonte: (COPPIN, 2012).

$$f'(x) = f(x) \cdot (1 - f(x))$$

Fórmula 9: Derivada da Função Logística. Fonte: (COPPIN, 2012).

CAPÍTULO 4: PLANEJAMENTO DO FRAMEWORK

O framework foi planejado e desenvolvido levando em consideração os paradigmas citados anteriormente neste trabalho, quando aplicáveis..

4.1 – Engenharia de Requisitos

Para elaborar o framework, inicialmente, algumas questões foram utilizadas como direcionamentos para o início das atividades:

- Qual o objetivo do framework?
- Quais funcionalidades se espera?
- Como ele será utilizado?
- Por quem ele será utilizado?

Além dos questionamentos, outras técnicas foram utilizadas nesta fase:

- Verificação de softwares semelhantes
- Criação de cenários hipotéticos de aplicação do framework
- Prototipagem.

4.2 – Gerenciamento de Configuração

Tão logo o projeto foi iniciado, começaram a surgir anotações e documentos. São importantes marcos no decorrer do projeto, que serão alterados diversas vezes no decorrer do tempo, e precisam de um controle.

Para gerenciar todo o conteúdo do projeto, desde as versões dos softwares até a documentação, foi escolhido um SCV (sistema de controle de versão). A ferramenta escolhida foi o GIT, que é um software livre criado por Linux Torvalds, o criador do Linux. O GIT é um SCV totalmente gratuito, que funciona de forma distribuída. Foi criado um diretório no computador de desenvolvimento, onde um repositório GIT foi iniciado e passou a controlar os dados.

Na imagem abaixo, o comando git log foi utilizado para exibir as últimas atualizações no projeto.

```
[wesley@mobiledragon: ~/Projects/engesoftware-pmc]$ git log

commit db16e8003760ff3fd26bbd684df2aefe66a2d164
Author: Wesley Rodrigues da Silva <wesley.it@gmail.com>
Date:   Fri Apr 13 16:53:53 2014 -0300

    Refatoração do código e comentários.

commit bd29ea922e137a783817a86f2209d737d7dfc1e3
Author: Wesley Rodrigues da Silva <wesley.it@gmail.com>
Date:   Thu Mar 12 11:04:58 2014 -0300

    Correção de bug na rede neural.

commit 214f65c7e15b7d39e5e17b40c87c6a60d32cbd9
Author: Wesley Rodrigues da Silva <wesley.it@gmail.com>
Date:   Tue Feb 22 17:01:02 2014 -0300

    Otimização no treinamento da rede.

commit 2111e9628b163287e11103eccbe7b16510bd00a7
Author: Wesley Rodrigues da Silva <wesley.it@gmail.com>
Date:   Tue Feb 22 16:50:00 2014 -0300

    Organização na pasta de documentos do projeto.
...|
```

Figura 7: Saída do comando git log. Fonte: Autor

4.3 – Gestão de Riscos

Apesar de se tratar de um projeto acadêmico, existem riscos que podem inviabilizar a criação do framework. Seguindo os conceitos explicados por Nogueira (2009), os riscos do projeto foram conhecidos, documentados e mitigados. Os fatores causadores de riscos foram tratados, de modo que nenhuma surpresa negativa impactou no projeto.

4.4 – Modelagem Visual

Neste momento, o sistema, que já possuía regras de negócio definidas e requisitos funcionais e não-funcionais bem especificados, passou a se tornar algo mais concreto. Com a modelagem visual, tornou-se mais simples entender, planejar e conceber as partes do framework.

Através de um mapa mental, as ideias a respeito do framework foram organizadas em uma estrutura hierarquizada de fácil assimilação.

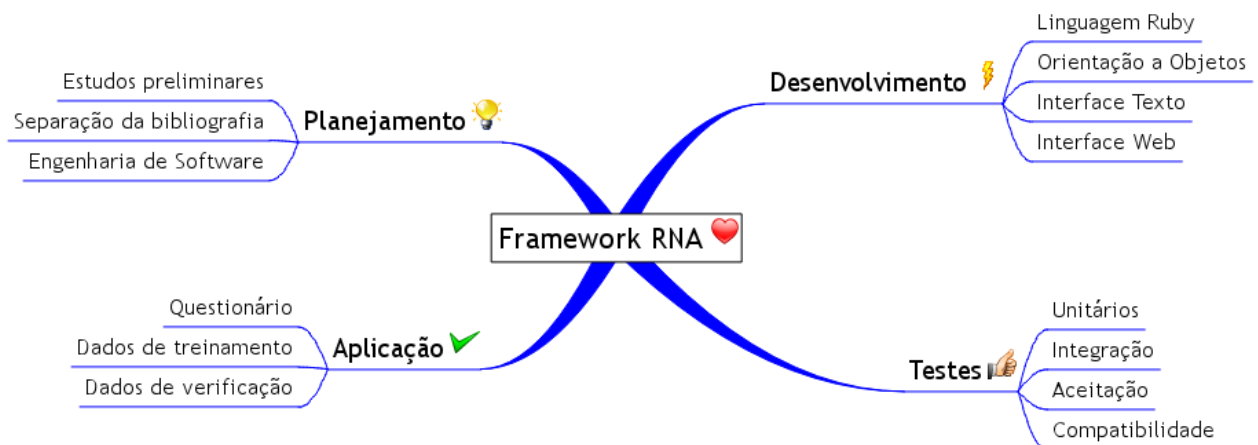


Figura 8: Mapa mental organizando as ideias. Fonte: Autor

Usando as informações obtidas com a engenharia de requisitos, foi construído um diagrama UML de casos de uso. Através deste diagrama, as funcionalidades do sistema ficam claramente explicadas.

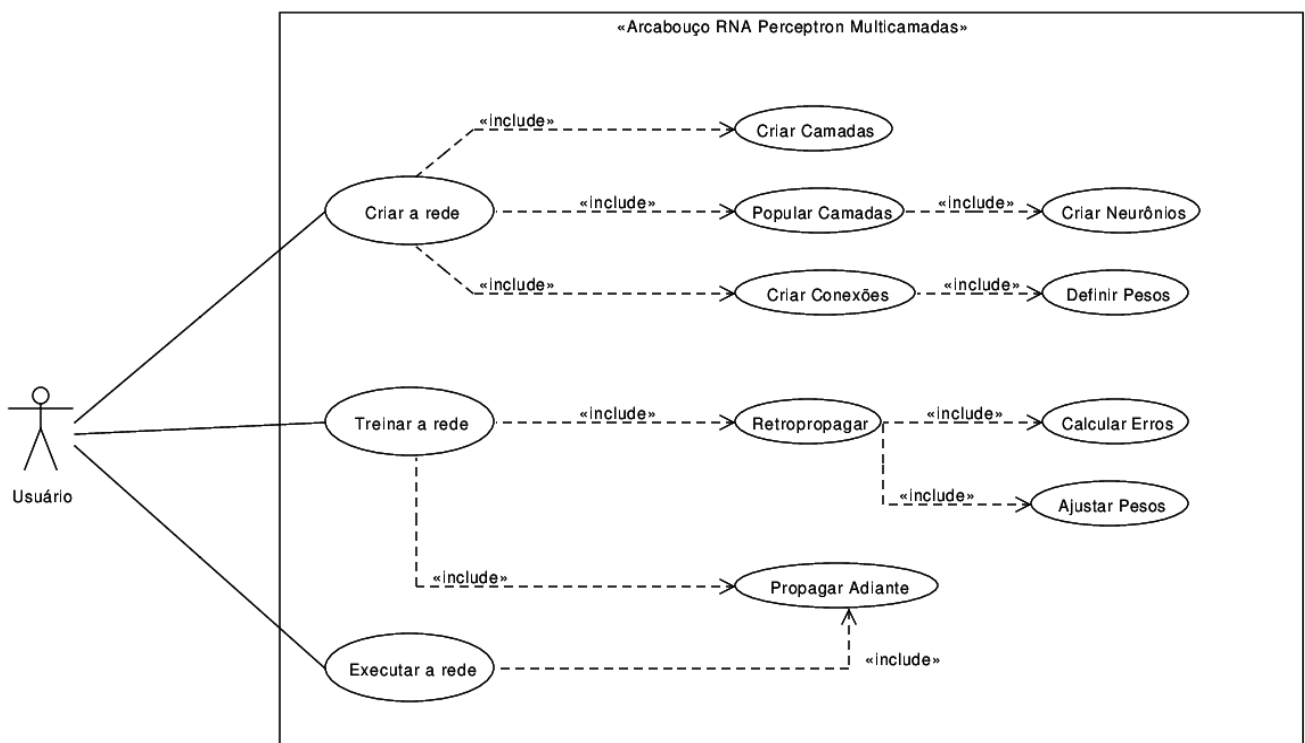


Figura 9: Diagrama de casos de uso. Fonte: Autor

Em seguida, visando entender todos os fluxos do sistema, foi elaborado um diagrama de atividades.

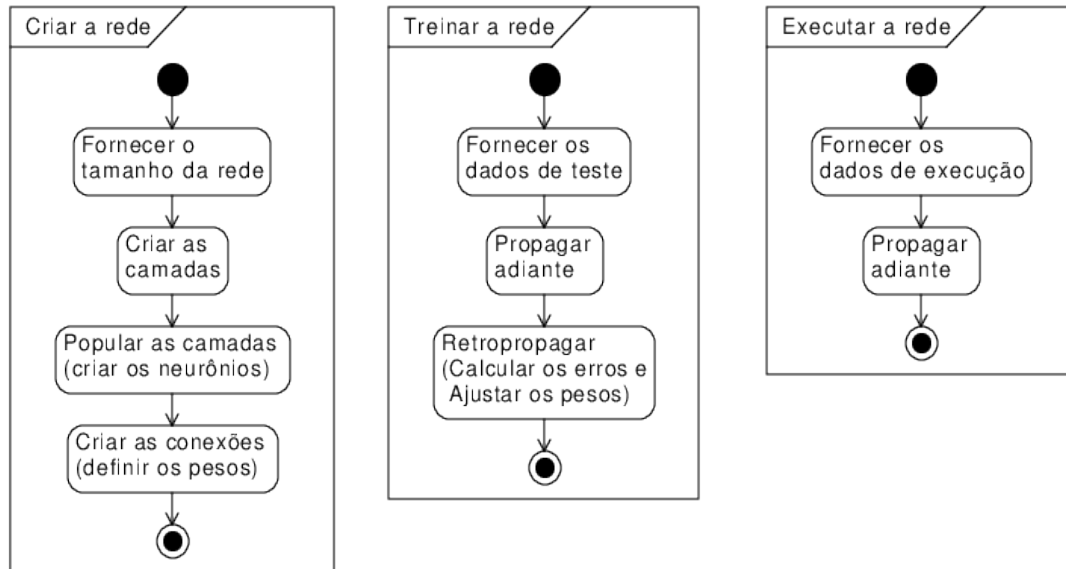


Figura 10: Diagrama de atividades. Fonte: Autor

Entender a sequência de utilização de um software é importante, principalmente para tarefas de manutenção ou modularização. O diagrama de sequência é uma ferramenta que cumpre este papel.

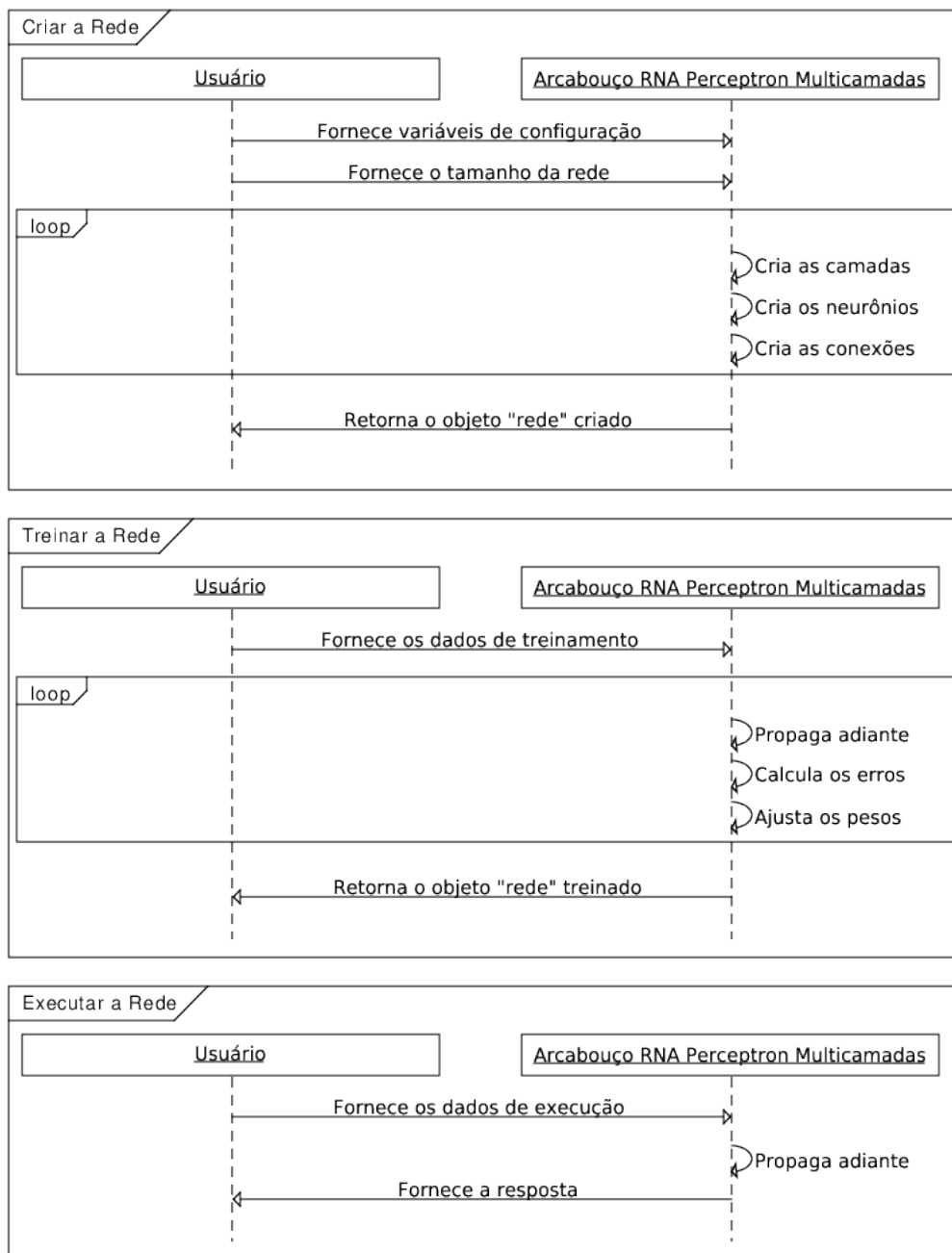


Figura 11: Diagrama de sequência. Fonte: Autor

Enfim, seguindo o paradigma da orientação a objetos, é necessário definir as classes do sistema. O diagrama de classes tem esta função.

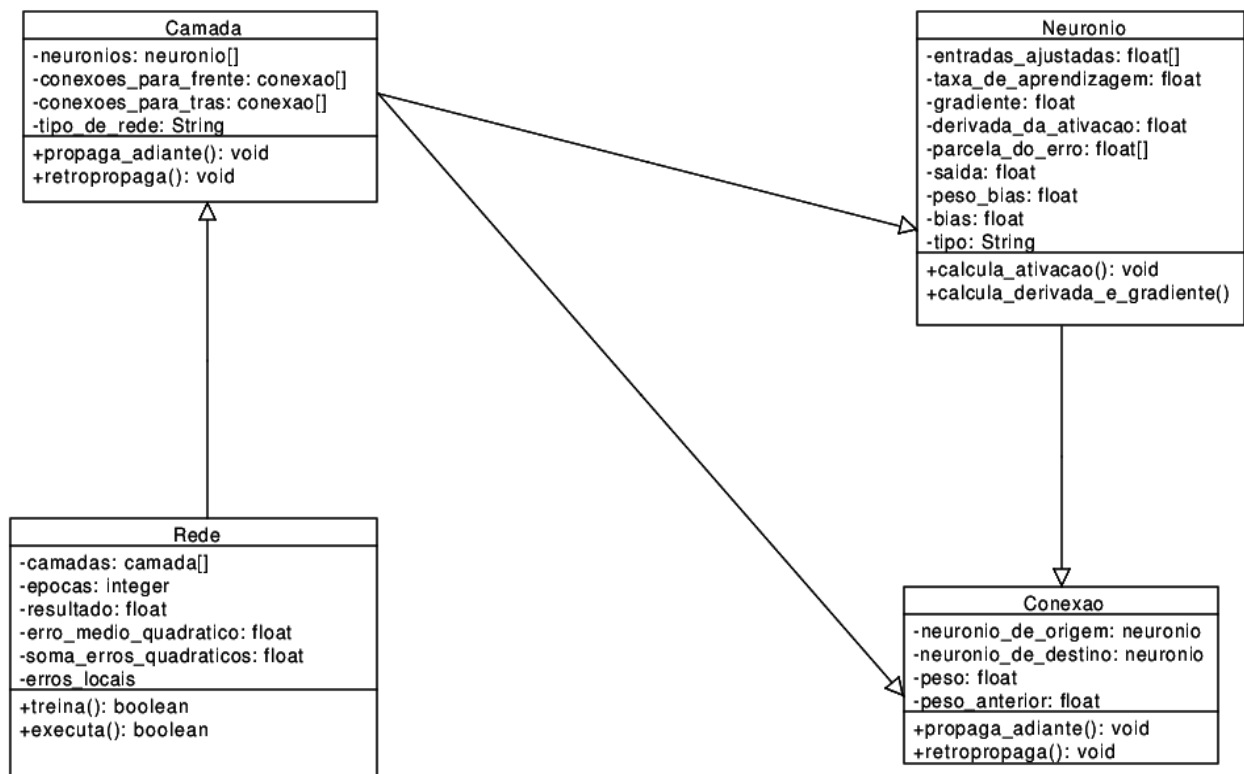


Figura 12: Esboço do Diagrama de classes. Fonte: Autor

4.5 – Metodologias de Desenvolvimento

O projeto foi desenvolvido levando em consideração os conceitos de metodologias ágeis. Foi um processo iterativo, com entregas regulares, gerenciamento do *backlog* e *sprints*. Entretanto, também foram utilizados alguns itens de metodologias prescritivas, como o RUP, para melhorar a compreensão do sistema de uma forma geral, bem como criar uma ampla documentação do processo.

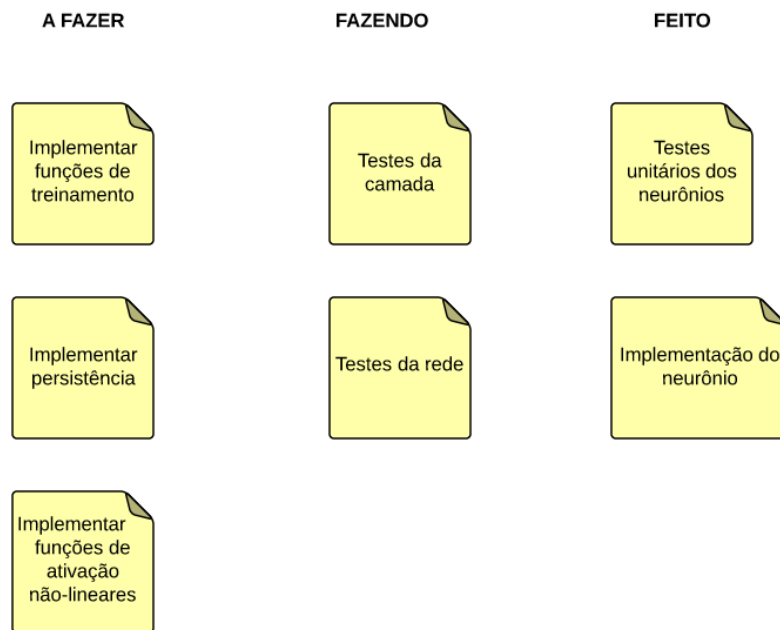


Figura 13: Fração do backlog de atividades. Fonte: Autor

4.6 – Normas e Modelos

Não foram utilizadas normas formais ou modelos prontos para este trabalho, entretanto, as melhores práticas ensinadas no curso de Engenharia de Software foram empregadas no projeto.

4.7 – Métricas

Um dos requisitos para utilização das métricas de software é conhecer a capacidade de desenvolvimento do time, ou possuir dados históricos. Como o projeto do framework foi desenvolvido apenas pelo autor, e não havia histórico de desenvolvimento deste tipo de aplicação, não foi possível estimar métricas.

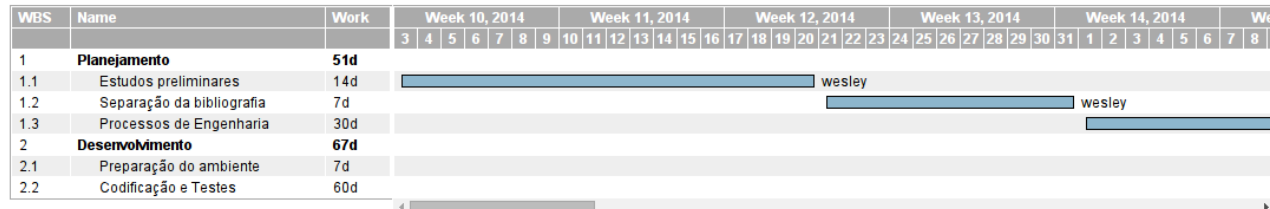
4.8 – Cronogramação

Seguindo as práticas recomendadas pelo PMBOK, um cronograma com duração estimada de cada tarefa foi estabelecido para o projeto do framework. A ferramenta escolhida foi o Planner, um software livre e de código aberto para criação de gráficos de Gantt.

Framework RNA

Start: March 2, 2014
Finish: August 13, 2014
Phase: Engenharia de Software
Report Date: September 5, 2014

Gantt Chart



Tasks

WBS	Name	Start	Finish	Work	Complete	Cost	Assigned to
1	Planejamento	Mar 3	May 12	51d			wesley
1.1	Estudos preliminares	Mar 3	Mar 20	14d	0%		wesley
1.2	Separação da bibliografia	Mar 21	Mar 31	7d	0%		wesley
1.3	Processos de Engenharia	Apr 1	May 12	30d	0%		wesley
2	Desenvolvimento	May 13	Aug 13	67d			wesley
2.1	Preparação do ambiente	May 13	May 21	7d	0%		wesley
2.2	Codificação e Testes	May 22	Aug 13	60d	0%		wesley

Resources

Name	Short name	Type	Group	Email	Cost
Wesley Rodrigues	wesley	Work		wesley.it@gmail.com	0

This file was generated by Planner

Figura 14: Gráfico Gantt. Fonte: Autor

4.9 – Implementação

Para a implementação do software, foi pensada uma arquitetura do tipo cliente-servidor, sendo que o servidor utiliza o protocolo HTTP. O software foi planejado para executar em ambiente Linux, porém, como a interface é HTTP, o sistema torna-se compatível com qualquer sistema operacional ou plataforma capaz de navegar na Internet através de um navegador de internet.

A linguagem de programação escolhida foi Ruby, por ser totalmente orientada a objetos, ter ampla documentação e uma comunidade ativa de desenvolvedores. Seguindo as boas práticas da linguagem, foi criado um arquivo para cada classe do projeto:

- `neuronio.rb`: classe que representa o neurônio e suas principais funções.
- `camada.rb`: classe que representa as camadas da rede neural, abriga os neurônios.
- `conexao.rb`: classe que simboliza as conexões sinápticas, responsável por transmitir os sinais ponderados pelos pesos.
- `rede.rb`: classe principal, já que agrupa e organiza as outras classes.
- `main.rb`: execução da rede em linha de comando.
- `main_web.rb`: execução da rede no navegador de internet.

4.10 – Testes

O TDD (Test Driven Development ou Desenvolvimento orientado a teste) é parte de uma metodologia ágil chamada XP (eXtreme Programming). Isso não restringe seu uso a esta metodologia. O que diferencia o TDD dos outros métodos é que se deve primeiro escrever os testes antes de codificar a lógica do sistema. Os testes são utilizados para facilitar no entendimento do projeto. Cada função ou componente do sistema é testada durante a codificação, garantindo que todo o sistema possua testes, aumentando o grau de qualidade (FREEMAN, 2012).

4.11 – Execução em linha de comando

Após passar por todas as iterações do ciclo de desenvolvimento, o software está pronto para ser executado. O framework suporta dois modos de aplicação: linha de comando ou interface web.

A sequência de imagens a seguir demonstra o framework em execução na linha de comando, aprendendo o padrão da tabela lógica XOR, lembrando que este não é um problema linearmente separável, exigindo mais de uma camada e/ou neurônios..

Inicialmente, é necessário instanciar a rede com o tamanho mínimo para resolver esta classe de problema, que exige ao menos 2 neurônios em uma camada interna e 1 na camada de saída.

```
## Fluxo de Execução *****
rede = Rede::new([2, 1])

@padrao = [ [[0, 0], [0]], [[0, 1], [1]], [[1, 0], [1]], [[1, 1], [0]] ]
rede.treina(@padrao)
puts "Salvando os pesos da rede"
rede.salva('/tmp/rede')
puts "Carregando os pesos da rede"
rede.carrega('/tmp/rede')

puts "\n--"
puts "Entrando 0, 0: #{rede.executa([0, 0])}"
puts "Entrando 0, 1: #{rede.executa([0, 1])}"
puts "Entrando 1, 0: #{rede.executa([1, 0])}"
puts "Entrando 1, 1: #{rede.executa([1, 1])}"
```

Figura 15: Código para execução de testes com tabela XOR. Fonte: Autor.

Depois de entrar com os parâmetros, basta executar o programa principal para que a rede seja treinada, e em seguida execute os dados.

```
[wesley@mobiledragon: ~/Projects/engesoftware-pmc]$ ./main.rb

-----
Rede treinada com sucesso na iteração 547!
-----

Camada 0
3.622729697962995 -3.6131358219796557 [-0.5145426586648859] [-1.4006124630875583]
Camada 1
[-1.4101014947846457]
--
Entrando 0, 0: [0.0]
Entrando 0, 1: [1.0]
Entrando 1, 0: [1.0]
Entrando 1, 1: [0.0]
```

Figura 16: Execução do framework em linha de comando. Fonte: Autor.

Conforme a imagem anterior, a rede demorou 547 ciclos ou iterações para aprender o padrão XOR. O número de ciclos variou de 5 à 900, de acordo com os números aleatórios inseridos nos pesos sinápticos.

4.12 – Execução em interface web

Assim como é possível executar o framework via interface de texto, também é possível executá-lo com outras interfaces, desde que as informações sejam corretamente enviadas à classe da rede neural artificial. A linguagem Ruby possui um módulo chamado Sinatra, que permite utilizar instruções HTTP nos códigos, implementando um servidor de aplicação web aderente ao protocolo HTTP 1.1.

A sequência de imagens a seguir demonstra o framework em execução na interface web, executando o mesmo teste feito no item anterior via linha de comando.

A principal diferença é que no caso da interface web, a linha de comando é utilizada apenas para inicializar o serviço Servidor HTTP, e depois todo o processo segue através do navegador.

```
[wesley@mobiledragon: ~/Projects/engesoftware-pmc]$ ./main_web.rb
[2014-09-08 19:25:39] INFO WEBrick 1.3.1
[2014-09-08 19:25:39] INFO ruby 2.1.2 (2014-05-08) [x86_64-linux-gnu]
== Sinatra/1.4.5 has taken the stage on 4567 for development with backup from WEBrick
[2014-09-08 19:25:39] INFO WEBrick::HTTPServer#start: pid=23563 port=4567
```

Figura 17: Inicialização do serviço HTTP. Fonte: Autor.

Depois de inicializar, basta acessar utilizando um navegador como o Mozilla Firefox ou Google Chrome, por exemplo.

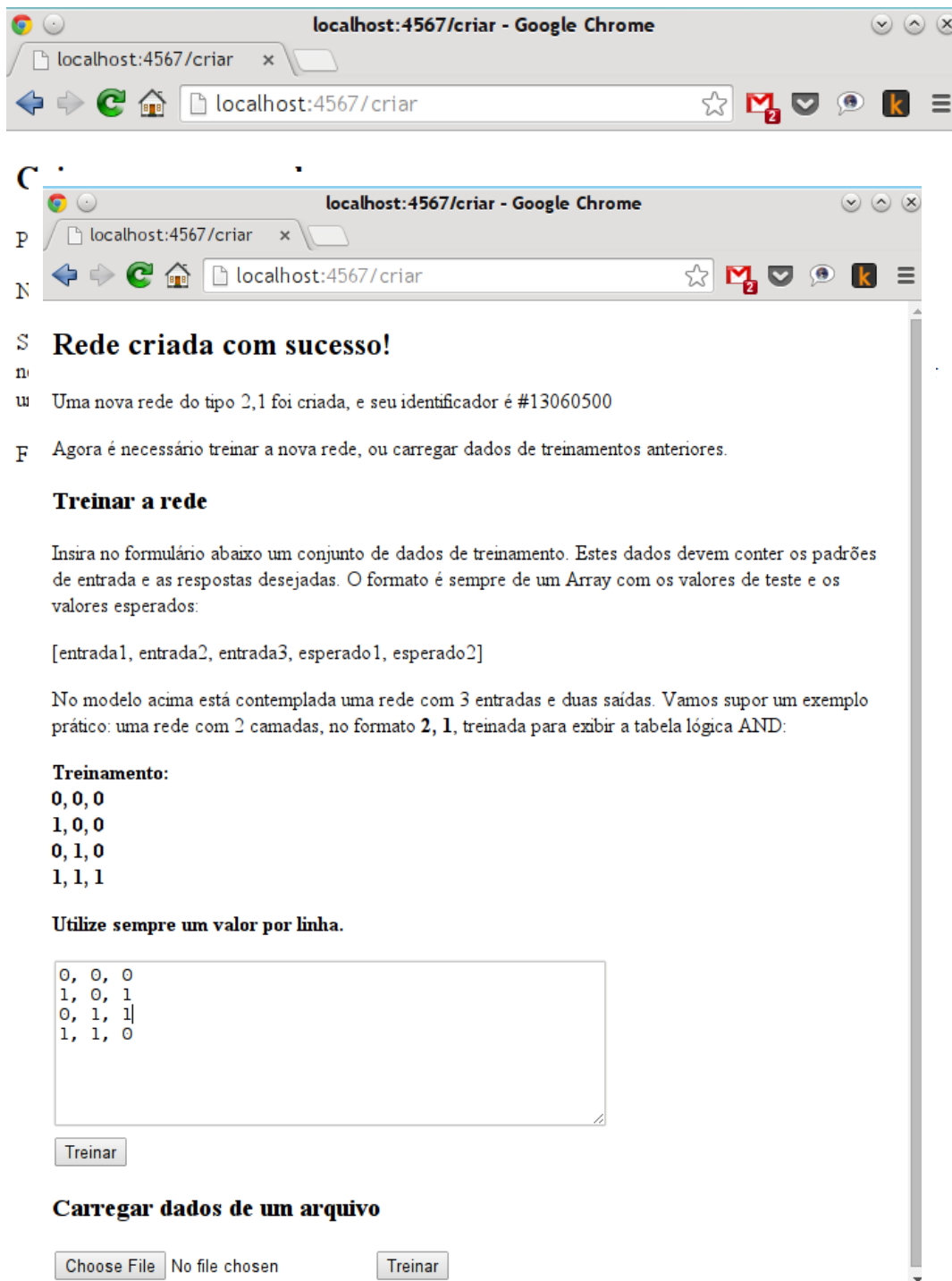


Figura 19: Inserção dos dados de treinamento. Fonte: Autor.

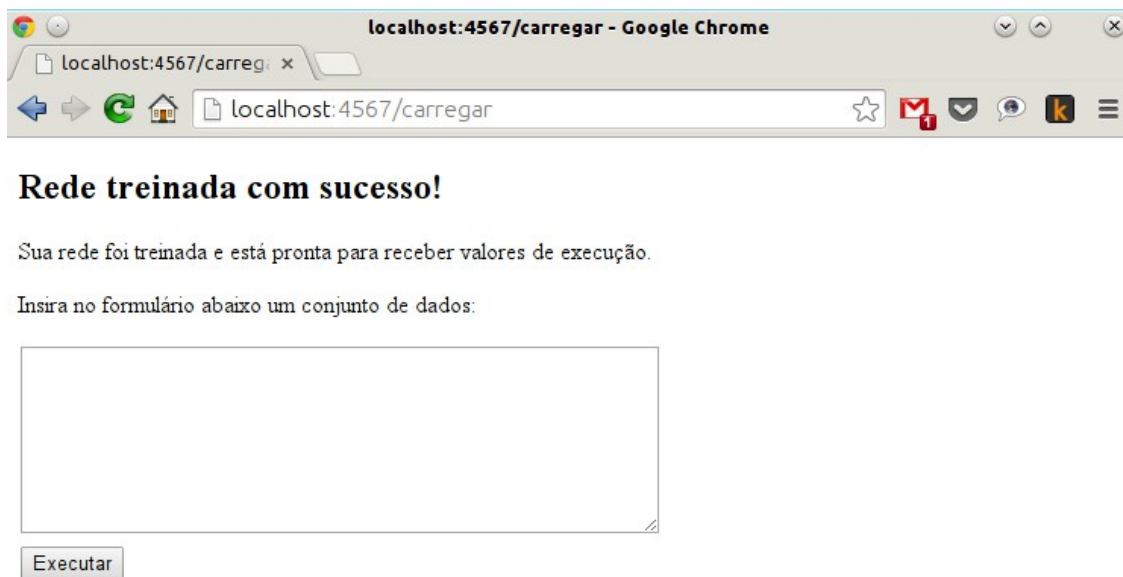


Figura 20: Rede treinada e pronta para execução. Fonte: Autor.

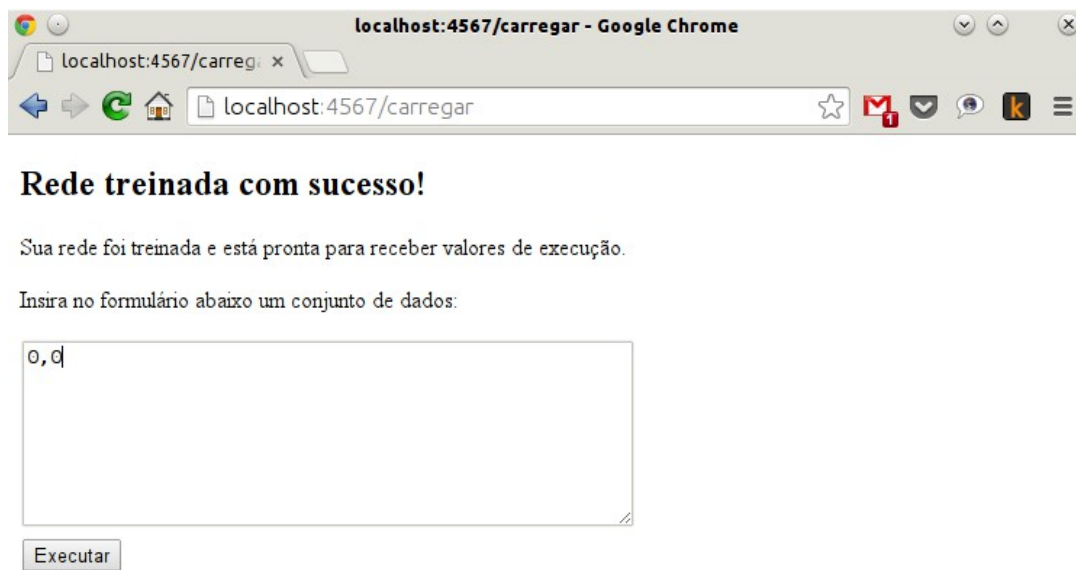


Figura 21: Execução de 0,0 na tabela XOR. Fonte: Autor.

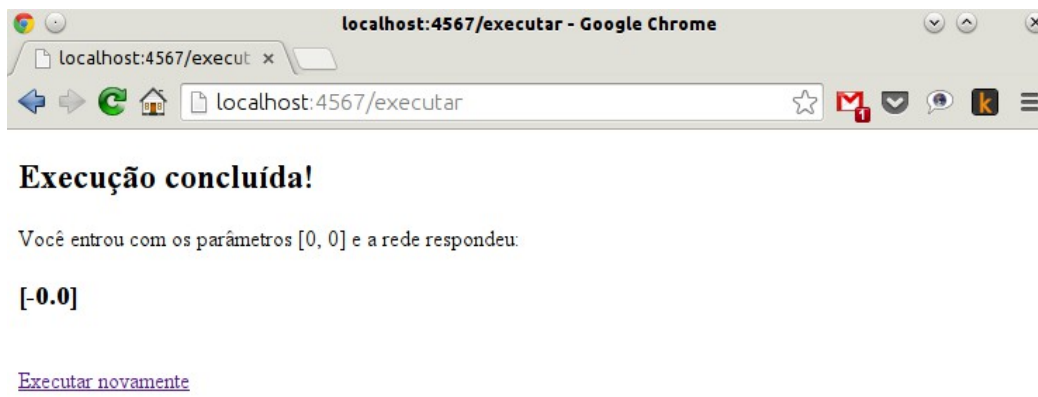


Figura 22: Resposta correta para 0,0. Fonte: Autor.



The image shows a web browser window with the title "localhost:4567/nova_execucao - Google Chrome". The address bar shows "localhost:4567/nova_execucao". The page content includes the heading "Entre com novos valores", a message "Sua rede foi treinada e está pronta para receber valores de execução.", and a prompt "Insira no formulário abaixo um conjunto de dados:". Below this is a text input field containing "1,0" and a button labeled "Executar".

Entre com novos valores

Sua rede foi treinada e está pronta para receber valores de execução.

Insira no formulário abaixo um conjunto de dados:

1,0

Executar

Figura 23: Execução de 1,0 na tabela XOR. Fonte: Autor.

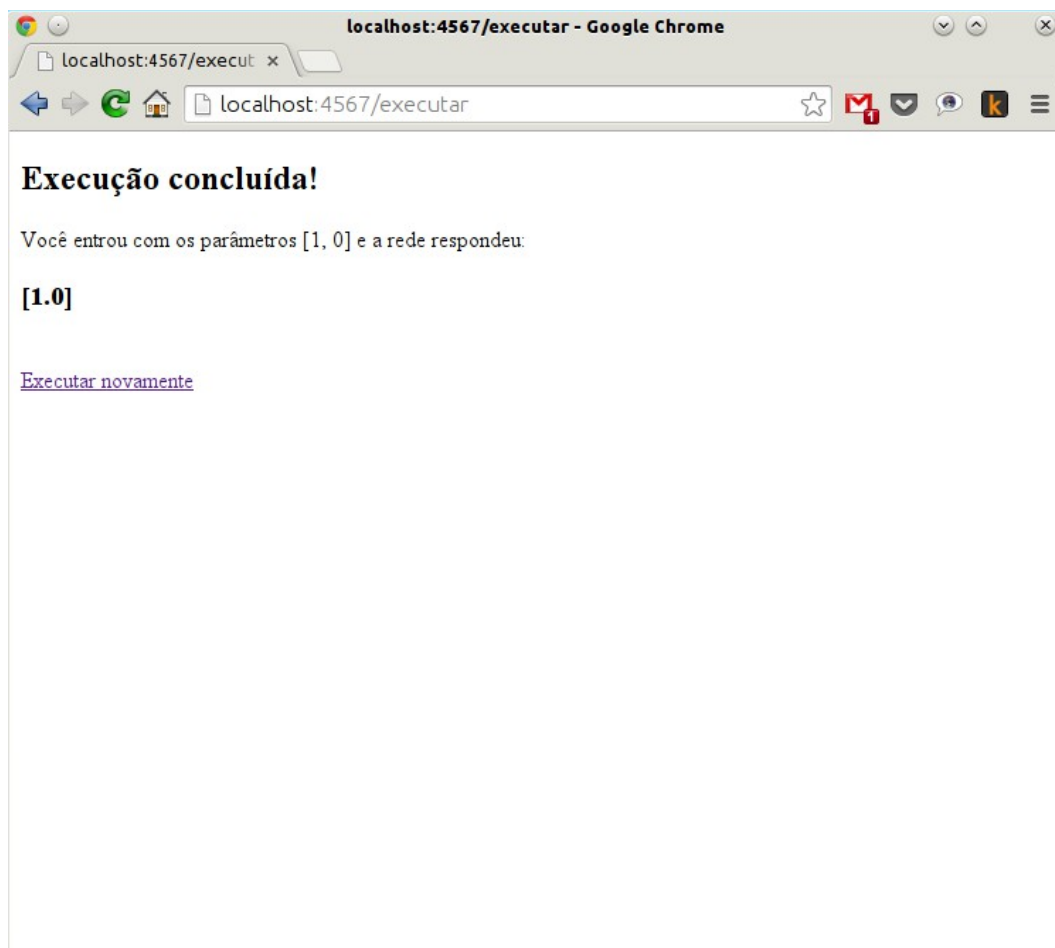


Figura 24: Resposta correta para 1,0. Fonte: Autor.

CONCLUSÃO DA PARTE I

Nesta parte do trabalho foi demonstrado que seguir as orientações dos paradigmas da engenharia de software é uma maneira de conseguir:

- organização no projeto, agilizando o entendimento e o acesso às informações
- redução de riscos, conhecendo e tratando os fatores
- aumento no grau de qualidade, seguindo normas e padrões e controlando todos os aspectos do projeto
- diminuição no tempo e esforço de desenvolvimento, reutilizando os componentes e acessando rapidamente as informações

A avaliação do processo foi positiva, os ganhos obtidos com a adoção do processo são significativos.

PARTÉ 2 – APLICAÇÃO DO FRAMEWORK

CAPÍTULO 5: CARACTERÍSTICAS DO AUTISMO

Autismo é um termo geral usado para descrever um grupo de transtornos de desenvolvimento do cérebro, conhecido como TEA (Transtornos do Espectro Autista). São manifestações que afetam o funcionamento social, a capacidade de comunicação (AUTISMO E REALIDADE, 2014), implicam um padrão restrito de comportamento e geralmente vem acompanhado de deficiência intelectual (DSM-IV, 1994).

5.1 – TEA

Além do Autismo, são considerados transtornos do espectro autista a síndrome de Asperger e o transtorno global do desenvolvimento sem outra especificação. Podemos localizar os TEA nos manuais de classificação dentro do capítulo dos transtornos globais do desenvolvimento (TGD), que inclui além dos TEA, a síndrome de Rett e o transtorno desintegrativo (AUTISMO E REALIDADE, 2014). Basicamente, 3 características são marcantes na criança autista:

- Inabilidade para interagir socialmente;
- Dificuldade no domínio da linguagem para comunicar-se ou lidar com jogos simbólicos;
- Padrão de comportamento restritivo e repetitivo.

As características também podem acontecer em diferentes níveis. Em casos mais leves, como a síndrome de Asperger, não há comprometimentos mais sérios; a criança apresenta dificuldade de socialização, interesse repetitivo por um mesmo assunto, porém, é comum ver portadores da síndrome de Asperger com QI acima da média (ALBERT EINSTEIN, 2013). Mas existem casos mais severos, onde o paciente é incapaz de manter contato social, além de apresentar comportamento agressivo e retardo mental (VARELLA, 2014).

Segundo o Dr. Dráuzio Varella (2013):

"O autismo acomete pessoas de todas as classes sociais e etnias, mais os meninos do que as meninas. Os sintomas podem aparecer nos primeiros meses de vida, mas dificilmente são identificados precocemente. O mais comum é os sinais ficarem evidentes antes de a criança completar três anos. De acordo com o quadro clínico, eles podem ser divididos em 3 grupos:

- 1) ausência completa de qualquer contato interpessoal, incapacidade de aprender a falar,

incidência de movimentos estereotipados e repetitivos, deficiência mental;

2) o portador é voltado para si mesmo, não estabelece contato visual com as pessoas nem com o ambiente; consegue falar, mas não usa a fala como ferramenta de comunicação (chega a repetir frases inteiras fora do contexto) e tem comprometimento da compreensão;

3) domínio da linguagem, inteligência normal ou até superior, menor dificuldade de interação social que permite aos portadores levar vida próxima do normal."

Autistas de bom rendimento, como os portadores de Asperger, podem apresentar alto desempenho em determinadas áreas do conhecimento (VARELLA, 2014).

Apesar das suspeitas de vacinas, dietas, contaminação por mercúrio ou fatores genéticos, a causa do Autismo ainda é desconhecida. Sabe-se que fatores genéticos têm influência na doença: em casos de autismo em gêmeos, é muito mais comum que idênticos nasçam ambos com autismo do que frateros (CDC, 2014).

5.2 – Algumas estatísticas sobre TEA

Um dos maiores produtores de pesquisas sobre TEA no mundo é o CDC (Center for Disease Control and Prevention — Centro de Controle e Prevenção de Doenças), que possui uma rede chamada ADDM (Autism and Developmental Disabilities Monitoring Network — Rede de Monitoramento de Autismo e Transtornos do Desenvolvimento). Segundo o CDC ADDM:

- Uma pesquisa feita nos EUA mostra que 12-13% das crianças com TEA nasceram em partos prematuros, muito pequenas ou em cesarianas (CDC Key Findings).
- Nos EUA, em 2010, 1 em cada 68 crianças com até 8 anos de idade foi diagnosticada com TEA. Em 2012, a quantidade era de 1 em 88, em 2009 era de 1 em 110 e em 2007 era de 1 em 150. Estes dados demonstram um crescimento de 29% nos casos de TEA.
- Crianças do sexo masculino apresentaram 5 vezes mais casos de TEA que do sexo feminino. Os transtornos também parecem ocorrer mais em crianças brancas do que em hispânicas ou negras.
- 46% das crianças com TEA nos EUA possuem média ou baixa capacidade intelectual ($QI < 85$).
- Menos da metade (44%) das crianças com TEA foram diagnosticadas antes dos 3 anos de idade.

- Cerca de 20% das crianças diagnosticadas com TEA nos EUA não receberam tratamento especial na escola ou clínica.

No Brasil, onde a população tem condições menos favorecidas que nos EUA, o governo federal vem tomando medidas para auxiliar a população portadora de TEA e seus familiares. Em dezembro de 2013 foi sancionada e publicada a lei nº 12.764, apelidada de “Lei Berenice Piana” em homenagem à mãe de uma pessoa com TEA que lutou pela conquista dos direitos. Esta lei institui a Política Nacional de Proteção dos Direitos da Pessoa com Transtorno do Espectro Autista, que assegura, entre outras conquistas, o acesso a ações e serviços de saúde para este público, incluindo o diagnóstico precoce, o atendimento multiprofissional, a nutrição adequada e a terapia nutricional, os medicamentos e as informações que auxiliem no diagnóstico e no tratamento (PLANALTO, 2013).

CAPÍTULO 6: DIAGNÓSTICO DA DOENÇA

Em medicina, alguns livros são referências importantes no diagnóstico de doenças. Segundo o Departamento de Diagnóstico e Classificação em Psiquiatria da Associação Brasileira de Psiquiatria, dois dos principais livros utilizados nesta área são o CID-10 e o DSM-IV (ABP, 2014).

6.1 – O CID-10

Em 1893 teve início um documento conhecido como Lista Internacional de Causas de Morte. Este documento evoluiu até se tornar o CID-10 (Classificação Internacional de Doenças e de Problemas Relacionados a Saúde, décima revisão). No CID-10, pode-se encontrar TEA em “F00-F99 – Transtornos Mentais e do Comportamento”, mais especificamente em “F84 – Transtornos globais do desenvolvimento” (OMS, 2008).

6.2 – O DSM

Sendo um dos mais importantes livros da medicina na área da psiquiatria, o DSM-IV, da Associação Americana de Psiquiatria, é utilizado como base no diagnóstico de TEA. Publicado desde 1952, possui cinco edições, entretanto, por ser recente, o DSM-V ainda não é tão utilizado quanto o DSM-IV (APA, 2014).

6.3 – Diagnóstico de TEA

O diagnóstico de TEA é obtido através de observação clínica e pelo histórico observado pelos responsáveis. Não existem marcadores biológicos que definam o quadro de TEA. Certos exames laboratoriais podem permitir a compreensão de fatores associados a TEA, mas, ainda assim, o diagnóstico do autismo é clínico e isso é um consenso entre os médicos, os editores do DSM-IV e do CID-10. Nestes manuais, há uma série de questões que, quando respondidas, podem dar indícios de TEA se determinadas condições forem satisfeitas. São questões que se referem ao relacionamento da criança com os pais, com os amigos, sobre o comportamento da criança, etc. (VARELLA, 2014).

Por se tratar de um questionário, diversos pesquisadores realizaram estudos tentando viabilizar ferramentas para apoiar os pais no diagnóstico prematuro de TEA. Uma busca por *Autism*

Screening Tests em bases de dados como Scielo, Teses USP ou Google Academics retorna dezenas de teses de doutorado e mestrado relatando experiências positivas nesse sentido. Se os dados consistem em um formulário, se existem regras que podem determinar através das respostas, se o indivíduo possui o transtorno, então um sistema que analise o questionário e verifique se as regras se aplicam será capaz, ao menos em teoria, de detectar o problema. Este raciocínio lógico levou o trabalho a seguir por este caminho.

CAPÍTULO 7: O QUESTIONÁRIO DE DIAGNÓSTICO

Como já existem pesquisas sérias em português sobre o assunto, foi escolhido um questionário já utilizado e validado para o teste deste estudo:

SATO, Fábio Pinato. Validação da versão em português de um questionário para avaliação de autismo infantil, 2008, Tradução para o português: Cristiane Silvestre Paula, Amanda Viviam dos Santos, Maria Clara Pacifico Mercadante, Clizeide da Costa Aguiar, Maria Eloisa B. D'Antino, José S. Schwartzman, Decio Brunoni, Marcos Tomanik Mercadante.

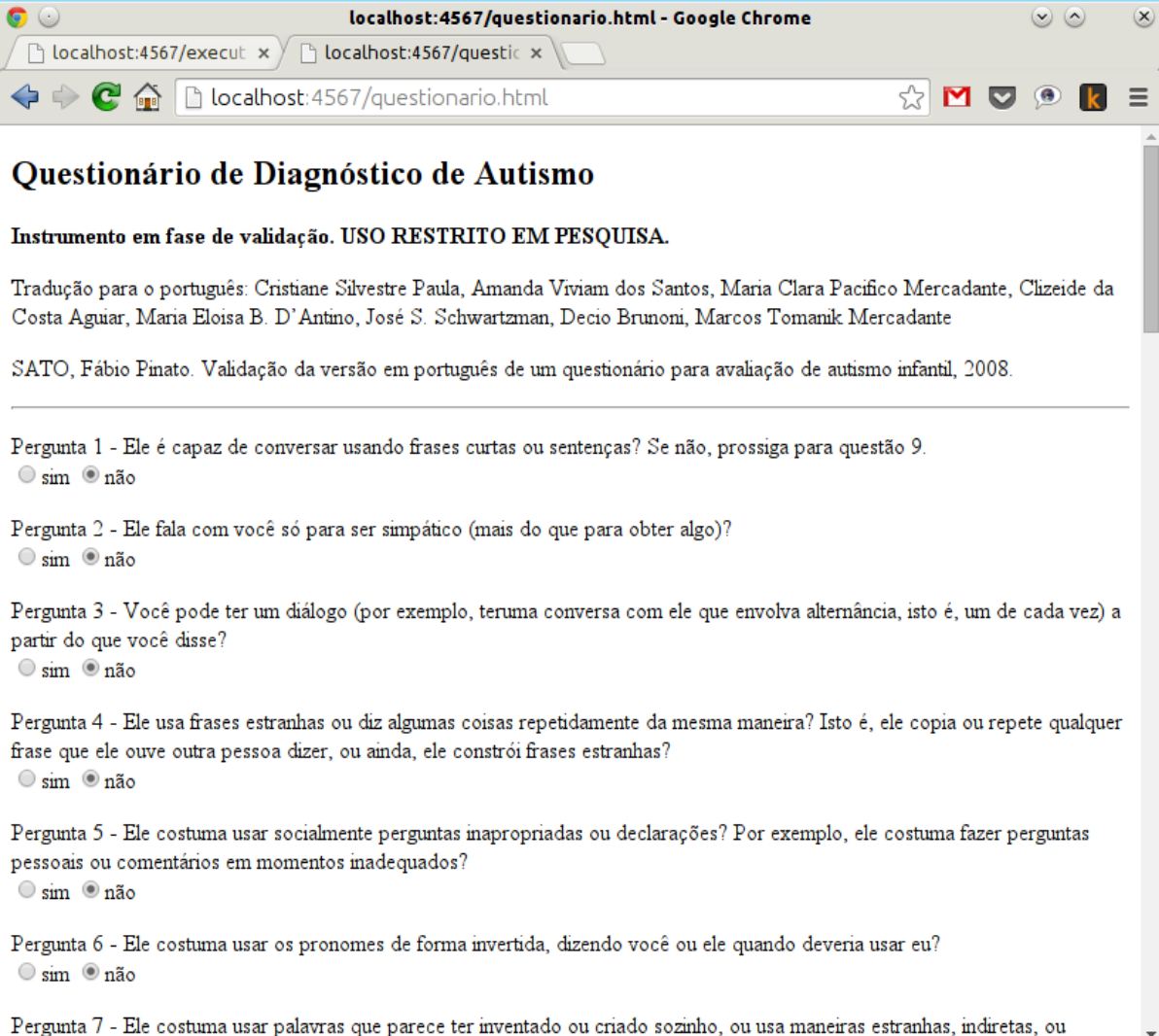
O questionário do Dr. Sato é composto de 40 perguntas, e segundo sua pesquisa, os transtornos são identificados quando a pontuação ultrapassa 15 respostas positivas. Entretanto, a análise direta é feita através de média, somando os pontos das questões e dividindo pela quantidade total de perguntas. A rede neural pode apresentar vantagens em relação ao método de questionário simples, pois ela pode aprender quais questões têm maior participação e incidência no caso de transtornos, aumentando assim seu peso na ponderação do resultado. Isso exigiria uma bateria de testes com pessoas portadoras de TEA, o que pode dificultar o processo de obtenção dos dados de treinamento.

CAPÍTULO 8: APLICAÇÃO DA RNA

O questionário possui 40 questões, por isso uma rede do tipo $40 \times 10 \times 3 \times 1$ foi criada, sendo que a entrada 1 corresponde à resposta sim e a entrada 0 corresponde à resposta não. O mesmo é válido para a saída, sendo que 1 significa alta probabilidade de apresentar autismo, enquanto 0 significa baixa probabilidade.

Para treinar a rede, 20 questionários foram respondidos com características que segundo o trabalho de Sato (2008), teriam altas chances de possuir TEA. Do mesmo modo, 20 questionários foram respondidos com características que estariam praticamente excluídas do diagnóstico de TEA.

Após a rede estar treinada, os pesos foram salvos em um arquivo, e o formulário foi apresentado em uma página. Depois de respondida, a página foi encaminhada à rede, que calculou a resposta e exibiu na página de resultados. Nesta página também foi exibido a média aritmética simples, fator usado por Sato (2008) para definir a chance do paciente portar algum TEA.



The image is a screenshot of a Google Chrome browser window. The address bar shows 'localhost:4567/questionario.html'. The page title is 'Questionário de Diagnóstico de Autismo'. The content includes a subtitle 'Instrumento em fase de validação. USO RESTRITO EM PESQUISA.', a list of translators, a citation for SATO (2008), and seven multiple-choice questions about autism symptoms. Each question has two radio button options: 'sim' and 'não'.

Questionário de Diagnóstico de Autismo

Instrumento em fase de validação. USO RESTRITO EM PESQUISA.

Tradução para o português: Cristiane Silvestre Paula, Amanda Viviam dos Santos, Maria Clara Pacifico Mercadante, Clizeide da Costa Aguiar, Maria Eloisa B. D'Antino, José S. Schwartzman, Decio Brunoni, Marcos Tomanik Mercadante

SATO, Fábio Pinato. Validação da versão em português de um questionário para avaliação de autismo infantil, 2008.

Pergunta 1 - Ele é capaz de conversar usando frases curtas ou sentenças? Se não, prossiga para questão 9.
☐ sim ☒ não

Pergunta 2 - Ele fala com você só para ser simpático (mais do que para obter algo)?
☐ sim ☒ não

Pergunta 3 - Você pode ter um diálogo (por exemplo, ter uma conversa com ele que envolva alternância, isto é, um de cada vez) a partir do que você disse?
☐ sim ☒ não

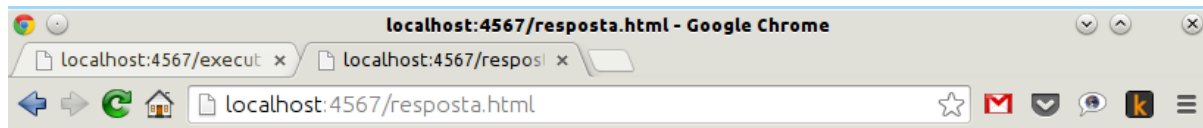
Pergunta 4 - Ele usa frases estranhas ou diz algumas coisas repetidamente da mesma maneira? Isto é, ele copia ou repete qualquer frase que ele ouve outra pessoa dizer, ou ainda, ele constrói frases estranhas?
☐ sim ☒ não

Pergunta 5 - Ele costuma usar socialmente perguntas inapropriadas ou declarações? Por exemplo, ele costuma fazer perguntas pessoais ou comentários em momentos inadequados?
☐ sim ☒ não

Pergunta 6 - Ele costuma usar os pronomes de forma invertida, dizendo você ou ele quando deveria usar eu?
☐ sim ☒ não

Pergunta 7 - Ele costuma usar palavras que parece ter inventado ou criado sozinho, ou usa maneiras estranhas, indiretas, ou

Figura 25: Página com formulário HTML do questionário. Fonte: Autor.



Respostas para o Questionário de Diagnóstico de Autismo

Instrumento em fase de validação. USO RESTRITO EM PESQUISA.

Tradução para o português: Cristiane Silvestre Paula, Amanda Viviam dos Santos, Maria Clara Pacifico Mercadante, Clizeide da Costa Aguiar, Maria Eloisa B. D'Antino, José S. Schwartzman, Decio Brunoni, Marcos Tomanik Mercadante

SATO, Fábio Pinato. Validação da versão em português de um questionário para avaliação de autismo infantil, 2008.

De um total de 40 perguntas, as respostas foram:

- 27 sim
- 13 não

Média aritmética: 67.5% sim, POSITIVO PARA TEA

Rede Neural: 1.0, POSITIVO PARA TEA

Figura 26: Resposta da rede versus resposta da média aritmética. Fonte: Autor.

CONCLUSÃO DA PARTE II

Nesta parte do trabalho foi demonstrado que existe uma grande variedade de materiais sobre autismo, e que eles podem ser adaptados para funcionar como mecanismos de seleção de dados para sistemas classificadores.

Ficou claro que é possível utilizar as redes neurais artificiais como instrumento de verificação para os dados coletados nos questionários de autismo.

Com o uso da interface HTML no navegador de internet, a utilização torna-se amigável, e pode ser distribuída via rede, possibilitando a coleta de informações até mesmo de pessoas em outras localidades.

PARTÉ 3 – CONSIDERAÇÕES FINAIS

CAPÍTULO 9: RESULTADOS DA PESQUISA

A intenção do autor com este trabalho era mostrar que a Engenharia de Software é uma poderosa ferramenta, que agrupa décadas de experiência em desenvolvimento de software e coloca nas mãos de quem a aplica toda a vivência dos profissionais e teóricos que contribuíram com a disciplina. Grandes feitos podem resultar de sua aplicação.

A construção do framework foi possível graças à aplicação da Engenharia de Software, o código fonte principal da rede neural encontra-se nos anexos deste trabalho, sob a licença GPL, também inclusa nos anexos.

O desenvolvimento do trabalho mostrou que o framework foi capaz de executar a tarefa de diagnóstico, atuando como um classificador. Isso abre inúmeras possibilidades de utilização do framework nas mais diversas áreas, já que não é necessário possuir conhecimentos avançados para utilizá-lo.

Finalmente, o trabalho mostrou que as engenharias podem servir como apoio à outras ciências, como psicologia e medicina, ajudando os profissionais e a população como um todo, o que a torna uma disciplina extremamente gratificante de se aplicar do ponto de vista humanístico.

CAPÍTULO 10: POSSÍVEIS APLICAÇÕES DO FRAMEWORK

O framework mostrou-se capaz de aprender padrões com uma quantidade significativa de entradas. Neste trabalho ele resolveu as tabelas lógicas AND, OR e XOR, além do questionário diagnóstico. Existem diversas áreas onde redes neurais podem ser aplicadas. Silva, Spatti e Flauzino (2010) citam algumas das áreas onde as redes costumam ser aplicadas e trazer benefícios em relação aos métodos tradicionais de programação:

- Avaliação de imagens
- Classificação de fala e escrita
- Reconhecimento de rostos
- Controle de meios de transporte
- Previsão de comportamentos
- Diagnósticos médicos
- Controle de transações financeiras
- Controle de aparelhos domésticos
- etc.

Enfim, há muitas áreas para aplicar os diferentes modelos de redes neurais, e com a ajuda da Engenharia de Software, o framework pode disponibilizar uma ferramenta simples para médicos, engenheiros agrônomos, economistas, e outros profissionais não especializados em desenvolvimento de software, para que eles possam dar foco ao seu próprio objeto de estudo, reaproveitando a rede neural como um serviço.

CAPÍTULO 11: CONFRONTAMENTO DE TECNOLOGIAS (LÓGICAS FUZZY E PARACONSISTENTE)

Além das RNAs do tipo perceptron, utilizadas neste trabalho, existem muitos outros tipos de redes, apoiadas por outros modelos teóricos. Duas redes bastante utilizadas em classificadores são as redes apoiadas pelas lógicas Fuzzy e Paraconsistente.

A lógica Fuzzy, estudada pelo Raciocínio Nebuloso, é uma lógica não clássica que não se limita a dois valores. Em uma lógica clássica, os valores podem ser verdadeiro ou falso, geralmente expressados como 0 e 1 por redes neurais. Uma rede neural Fuzzy apresenta um valor entre 0.0 e 1.0, de acordo com a plausibilidade do resultado (COPIN, 2012).

Outra lógica não clássica utilizada como base para redes neurais é a lógica Paraconsistente. Assim como na lógica Fuzzy, os valores não se limitam à 0 ou 1, mas podem ser expressados em um reticulado, com valores intermediários entre verdadeiro, falso, incompleto ou inconsistente.

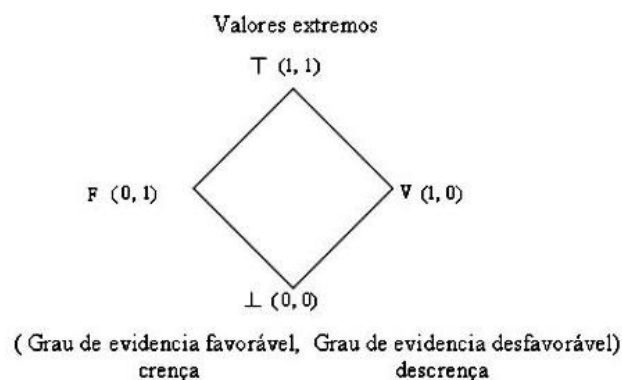


Figura 27: Reticulado representativo da Lógica Paraconsistente Anotada. Fonte: (INACIO, 2014)

Trabalhos futuros poderiam explorar estas duas variações de redes neurais para criar frameworks com estas lógicas, inclusive comparando os resultados e indicando quais seriam as escolhas mais adequadas para cada domínio de problema, já que estas lógicas lidam de maneiras diferentes com questões como exatidão e incertezas.

ANEXO I – QUESTIONÁRIO DE DIAGNÓSTICO DE AUTISMO

Instrumento em fase de validação. USO RESTRITO EM PESQUISA.

SATO, Fábio Pinato. Validação da versão em português de um questionário para avaliação de autismo infantil, 2008. Tradução para o português: Cristiane Silvestre Paula, Amanda Viviam dos Santos, Maria Clara Pacifico Mercadante, Clizeide da Costa Aguiar, Maria Eloisa B. D’Antino, José S. Schwartzman, Decio Brunoni, Marcos Tomanik Mercadante

Pergunta 1 - Ele é capaz de conversar usando frases curtas ou sentenças? Se não, prossiga para questão 9.

Pergunta 2 - Ele fala com você só para ser simpático (mais do que para obter algo)?

Pergunta 3 - Você pode ter um diálogo (por exemplo, ter uma conversa com ele que envolva alternância, isto é, um de cada vez) a partir do que você disse?

Pergunta 4 - Ele usa frases estranhas ou diz algumas coisas repetidamente da mesma maneira? Isto é, ele copia ou repete qualquer frase que ele ouve outra pessoa dizer, ou ainda, ele constrói frases estranhas?

Pergunta 5 - Ele costuma usar socialmente perguntas inapropriadas ou declarações? Por exemplo, ele costuma fazer perguntas pessoais ou comentários em momentos inadequados?

Pergunta 6 - Ele costuma usar os pronomes de forma invertida, dizendo você ou ele quando deveria usar eu?

Pergunta 7 - Ele costuma usar palavras que parece ter inventado ou criado sozinho, ou usa maneiras estranhas, indiretas, ou metafóricas para dizer coisas? Por exemplo, diz “chuva quente” ao invés de vapor.

Pergunta 8 - Ele costuma dizer a mesma coisa repetidamente, exatamente da mesma maneira, ou insiste para você dizer as mesmas coisas muitas vezes?

Pergunta 9 - Existem coisas que são feitas por ele de maneira muito particular ou em determinada ordem, ou seguindo rituais que ele te obriga fazer?

Pergunta 10 - Até onde você percebe, a expressão facial dele geralmente parece apropriada à situação particular?

Pergunta 11 - Ele alguma vez usou a tua mão como uma ferramenta, ou como se fosse parte do próprio corpo dele (por exemplo, apontando com seu dedo, pondo a sua mão numa maçaneta para abrir a porta)?

Pergunta 12 - Ele costuma ter interesses especiais que parecem esquisitos a outras pessoas (e.g., semáforos, ralos de pia, ou itinerários de ônibus)?

Pergunta 13 - Ele costuma se interessar mais por partes de um objeto ou brinquedo (e.g., girar as rodas de um carro), mais do que usá-lo com sua função original?

Pergunta 14 - Ele costuma ter interesses específicos, apropriados para sua idade e para seu grupo de colegas, porém estranhos pela intensidade do interesse (por exemplo, conhecer todos os tipos de trens, conhecer muitos detalhes sobre dinossauros)?

Pergunta 15 - Ele costuma de maneira estranha olhar, sentir/examinar, escutar, provar ou cheirar coisas ou pessoas?

Pergunta 16 - Ele costuma ter maneirismos ou jeitos estranhos de mover suas mãos ou dedos, tal como “um bater de asas” (flapping), ou mover seus dedos na frente dos seus olhos?

Pergunta 17 - Ele costuma fazer movimentos complexos (e esquisitos) com o corpo inteiro, tal como girar, pular ou balançar repetidamente para frente e para trás?

Pergunta 18 - Ele costuma machucar-se de propósito, por exemplo, mordendo o braço ou batendo a cabeça?

Pergunta 19 - Ele tem algum objeto (que não um brinquedo macio ou cobertor) que ele carrega por toda parte?

Pergunta 20 - Ele tem algum amigo em particular ou um melhor amigo?

Pergunta 21 - Quando ele tinha 4-5 anos ele repetia ou imitava espontaneamente o que você fazia (ou a outras pessoas) (tal como passar o aspirador no chão, cuidar da casa, lavar pratos, jardinagem, consertar coisas)?

Pergunta 22 - Quando ele tinha 4-5 anos ele apontava as coisas ao redor espontaneamente apenas para mostrar coisas a você (e não porque ele as desejava)?

Pergunta 23 - Quando ele tinha 4-5 anos ele costumava usar gestos para mostrar o que ele queria (não considere se ele usava tua mão para apontar o que queria)?

Pergunta 24 - Quando ele tinha 4-5 anos usava a cabeça pra dizer sim?

Pergunta 25 - Quando ele tinha 4-5 anos sacudia a sua cabeça para dizer ‘não’?

Pergunta 26 - Quando ele tinha 4-5 anos ele habitualmente olhava você diretamente no rosto quando fazia coisas com você ou conversava com você?

Pergunta 27 - Quando ele tinha 4-5 anos sorria de volta se alguém sorrisse para ele?

Pergunta 28 - Quando ele tinha 4-5 anos ele costumava mostrar coisas de seu interesse para chamar a sua atenção?

Pergunta 29 - Quando ele tinha 4-5 anos ele costumava dividir coisas com você, além de alimentos?

Pergunta 30 - Quando ele tinha 4-5 anos ele costumava querer que você participasse de algo que o estava divertindo?

Pergunta 31 - Quando ele tinha 4-5 anos ele costumava tentar confortá-lo se você ficasse triste ou magoado?

Pergunta 32 - Entre as idades de 4 a 5 anos, quando queria algo ou alguma ajuda, costumava olhar para você e fazia uso de sons ou palavras para receber sua atenção?

Pergunta 33 - Entre as idades de 4 a 5 anos tinha expressões faciais normais, isto é, demonstrava suas emoções por expressões faciais?

Pergunta 34 - Quando ele estava com 4 ou 5 anos ele costumava participar espontaneamente e/ou tentava imitar ações em jogos sociais – tais como “Polícia e Ladrão” ou “Pega-Pega”?

Pergunta 35 - Quando ele estava com 4 ou 5 anos jogava jogos imaginários ou brincava de “faz de conta”?

Pergunta 36 - Quando ele estava com 4 ou 5 anos parecia interessado em outras crianças da mesma idade que ele não conhecia?

Pergunta 37 - Quando ele estava com 4 ou 5 anos reagia positivamente quando outra criança aproximava-se dele?

Pergunta 38 - Quando ele estava com 4 ou 5 anos, se você entrasse no quarto e iniciasse uma conversa com ele sem chamar seu nome, ele habitualmente te olhava e prestava atenção em você?

Pergunta 39 - Quando ele estava com 4 ou 5 anos ele costumava brincar de “faz de conta” com outra criança, de forma que você percebia que eles estavam entendendo ser uma brincadeira?

Pergunta 40 - Quando ele estava com 4 ou 5 anos ele brincava cooperativamente em jogos de grupo, tal como esconde-esconde e jogos com bola?

*ANEXO II – CÓDIGO FONTE E LICENÇA DE
UTILIZAÇÃO*

Este estudo foi elaborado em ambiente GNU/Linux livre, utilizando ferramentas gratuitas e de código aberto, de modo que qualquer interessado possa reproduzir e avançar nos trabalhos iniciados.

Apenas o código fonte do framework via modo texto serão apresentados, já que a parte web do framework ainda precisa de melhorias para torná-la mais amigável aos usuários que não dominam desenvolvimento de software.

Arquivo main.rb:

```
#!/usr/bin/env ruby
# Coding: utf-8
# *****
# Author: Wesley Rodrigues <wesley.it@gmail.com>
# Description: OOPMC - Perceptron Multicamadas Orientado a Objetos
# Version: 0.1
# *****
#
# Copyright 2014 Wesley Rodrigues da Silva <wesley.it@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.
#
#
require './rede.rb'
require './camada.rb'
require './neuronio.rb'
require './conexao.rb'

## Definições: Estes parâmetros serão acessíveis via formulário pelo
# Sinatra
$limite_de_epocas = 2000
$taxa_de_aprendizagem = 0.05
$erro_aceitavel = 0.0001
$taxa_de_momentum = 0.97
$usar_bias = true
$bias = 1
$range_peso_bias = (-0.5..0.5)
$range_peso_conexao = (-0.5..0.5)
$debug = false
$tipo_de_rede = :hiperbolica
```

```

$arredondar_saida = true
$rede_treinada = false
# *****

## Fluxo de Execução *****
rede = Rede::new([2, 1])

@padrao = [ [[0, 0], [0]], [[0, 1], [1]], [[1, 0], [1]], [[1, 1], [0]] ]
rede.treina(@padrao)
#puts "Salvando os pesos da rede"
#rede.salva('/tmp/rede')
#puts "Carregando os pesos da rede"
#rede.carrega('/tmp/rede')

puts "\n--"
puts "Entrando 0, 0: #{rede.executa([0, 0])}"
puts "Entrando 0, 1: #{rede.executa([0, 1])}"
puts "Entrando 1, 0: #{rede.executa([1, 0])}"
puts "Entrando 1, 1: #{rede.executa([1, 1])}"

```

Arquivo rede.rb:

```

#!/usr/bin/env ruby
# Coding: utf-8
# *****
# Author: Wesley Rodrigues <wesley.it@gmail.com>
# Description: OOPMC - Perceptron Multicamadas Orientado a Objetos
# Version: 0.1
# *****
#
# Copyright 2014 Wesley Rodrigues da Silva <wesley.it@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.
#
#

class Rede
  attr_accessor :camadas
  def initialize(formato)
    @camadas = []
    formato.each_with_index do |n, i|
      if i == (formato.length - 1)
        @ultima = true
      else

```

```

        @ultima = false
      end
      @camadas << Camada::new(@ultima, n)
    end
    @camadas.each_index do |i| ## cria as conexoes
      break if @camadas[i] == @camadas.last
      @camadas[i].neuronios.each do |nc1|
        @camadas[i + 1].neuronios.each do |nc2|
          @conexao = Conexao::new(nc1, nc2)
          @camadas[i].conexoes_para_frente << @conexao
          @camadas[i + 1].conexoes_para_tras << @conexao
        end
      end
    end
  end

end

def lista_pesos
  @camadas.each_with_index do |c, i|
    puts "\nCamada #{i}"
    c.conexoes_para_frente.each { |cn| print "#{cn.peso} " }
    c.neuronios.each { |n| print "[#{n.peso_bias}] " }
  end
end

def executa(padrao)
  @camadas.first.neuronios.each { |n| n.entradas_ajustadas = padrao }
  @camadas.each do |c|
    c.propaga_adiante
    break if c == @camadas.last
    c.conexoes_para_frente.each { |con| con.propaga_adiante }
  end
  @resultado = []
  @camadas.last.neuronios.each { |n| @resultado << n.saida }
  return @resultado
end

def treina(padrao_de_treino)
  @epocas = 0
  @erro_medio_quadrtico = 1000
  @soma_erros_quadrticos = []
  loop do
    puts "\nEpoca #{@epocas}" if $debug == true
    padrao_de_treino.each do |p|
      @resultados_obtidos = executa(p[0])
      @resultados_desejados = p[1]
      @erros_locais =
        @resultados_desejados.zip(@resultados_obtidos).map { |n1, n2| n1 - n2 }
      @erros_locais.each_index { |i|
        @camadas.last.neuronios[i].parcela_do_erro << @erros_locais[i]
        @erro_quadrtico = 0.5 * ((@erros_locais.map { |erro|
          erro ** 2 }).inject { |n1, n2| n1 + n2 })
        @soma_erros_quadrticos << @erro_quadrtico ## pega
        todos os erros quadrticos de uma época
        puts "X:#{p[0]}, d:#{@resultados_desejados},
Y:#{@resultados_obtidos}, Er:#{@erros_locais}, ErQ:#{@erro_quadrtico}" if $debug ==
true
      }
      @camadas.reverse_each do |c|
        c.retropropaga
        c.conexoes_para_tras.each { |con|

```

```

con.retropropaga}
                                end
                                end
                                @erro_medio_quadratco = (@soma_erros_quadratcos.inject{|n1,
n2| n1 + n2}) / @soma_erros_quadratcos.length
                                puts "EMQ = #{@erro_medio_quadratco}" if $debug == true
                                @soma_erros_quadratcos = []
                                @epocas += 1
                                break if @epocas > $limite_de_epocas
                                if @erro_medio_quadratco < $erro_aceitavel
                                    $rede_treinada = true;
                                    puts
                                "\n\n-----"
                                puts "Rede treinada com sucesso na iteração
#{@epocas}!"
                                puts
                                "-----"
                                break
                                end
                                end
                                end
                                lista_pesos
                                end

                                def salva(arquivo)
                                    File.open(arquivo, 'w') {|f| f.write(Marshal.dump(self)) }
                                end

                                def carrega(arquivo)
                                    @camadas = Marshal.load(File.read(arquivo)).camadas
                                end

                                end
end

```

Arquivo camada.rb:

```

#!/usr/bin/env ruby
# Coding: utf-8
# *****
# Author: Wesley Rodrigues <wesley.it@gmail.com>
# Description: OOPMC - Perceptron Multicamadas Orientado a Objetos
# Version: 0.1
# *****
#
# Copyright 2014 Wesley Rodrigues da Silva <wesley.it@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software

```

```

# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.
#
#

class Camada
  attr_accessor :neuronios
  attr_accessor :conexoes_para_frente
  attr_accessor :conexoes_para_tras

  def initialize(ultima, neuronios)
    @neuronios = []
    @conexoes_para_frente = []
    @conexoes_para_tras = []
    if $tipo_de_rede == :hiperbolica
      neuronios.times {@neuronios << Neuronio::new(:hiperbolica)}
    elsif $tipo_de_rede == :linear
      neuronios.times {@neuronios << Neuronio::new(:linear)}
    elsif $tipo_de_rede == :hiper_linear
      if ultima == true
        neuronios.times {@neuronios << Neuronio::new(:linear)}
      else
        neuronios.times {@neuronios <<
Neuronio::new(:hiperbolica)}
      end
    end

    end

    def propaga_adiante()
      @neuronios.each {|n| n.calcula_ativacao}
    end

    def retropropaga()
      @neuronios.each {|n| n.calcula_derivada_e_gradiente}
    end
  end
end

```

Arquivo neurônio.rb:

```

#!/usr/bin/env ruby
# Coding: utf-8
# *****
# Author: Wesley Rodrigues <wesley.it@gmail.com>
# Description: OOPMC - Perceptron Multicamadas Orientado a Objetos
# Version: 0.1
# *****
#
# Copyright 2014 Wesley Rodrigues da Silva <wesley.it@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```

```

# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.
#
#

class Neuronio
  attr_accessor :entradas_ajustadas
  attr_accessor :parcela_do_erro
  attr_accessor :gradiente
  attr_accessor :taxa_de_aprendizagem
  attr_accessor :derivada_da_ativacao
  attr_accessor :saida
  attr_accessor :peso_bias

  def initialize(tipo)
    @entradas_ajustadas = []
    @peso_bias = rand($range_peso_bias)
    @parcela_do_erro = []
    @taxa_de_aprendizagem = $taxa_de_aprendizagem
    @tipo = tipo
  end

  def calcula_ativacao()
    if @tipo == :linear
      @saida = (@entradas_ajustadas.inject {|n1, n2| n1 + n2} + $bias
* @peso_bias) if $usar_bias == true
      @saida = (@entradas_ajustadas.inject {|n1, n2| n1 + n2}) if
$usar_bias == false
    elsif @tipo == :hiperbolica
      @saida = Math.tanh(@entradas_ajustadas.inject {|n1, n2| n1 +
n2} + $bias * @peso_bias) if $usar_bias == true
      @saida = Math.tanh(@entradas_ajustadas.inject {|n1, n2| n1 +
n2}) if $usar_bias == false
    end
    @saida = @saida.round(1) if $arredondar_saida == true
  end

  def calcula_derivada_e_gradiente()
    @derivada_da_ativacao = 1 if @tipo == :linear
    @derivada_da_ativacao = 1 - (@saida ** 2) if @tipo == :hiperbolica
    @gradiente = @derivada_da_ativacao * @parcela_do_erro.inject {|n1, n2|
n1 + n2} ## calcula o gradiente somando as parcelas de erro
    @peso_bias = @peso_bias + @taxa_de_aprendizagem * @gradiente ##
atualiza o peso do bias
    @parcela_do_erro = [] ## limpa o erro depois de gerar o gradiente
  end
end

```

Arquivo conexao.rb:

```

#!/usr/bin/env ruby
# Coding: utf-8

```

```

# *****
# Author: Wesley Rodrigues <wesley.it@gmail.com>
# Description: OOPMC - Perceptron Multicamadas Orientado a Objetos
# Version: 0.1
# *****
#
# Copyright 2014 Wesley Rodrigues da Silva <wesley.it@gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.
#
#

class Conexao
  attr_accessor :peso
  def initialize(origem, destino)
    @peso = rand($range_peso_conexao)
    @peso_anterior = @peso
    @neuronio_de_origem = origem
    @neuronio_de_destino = destino
  end

  def propaga_adiante()
    @neuronio_de_destino.entradas_ajustadas << @neuronio_de_origem.saida *
@peso
  end

  def retropropaga()
    @delta_peso = ($taxa_de_momentum * (@peso - @peso_anterior)) +
(@neuronio_de_destino.taxa_de_aprendizagem * @neuronio_de_destino.gradiente *
@neuronio_de_origem.saida)
    @peso_anterior = @peso
    @peso = @peso + @delta_peso
    @neuronio_de_origem.parcela_do_erro << @neuronio_de_destino.gradiente *
@peso
  end
end
end

```


ANEXO III – ARTIGO PUBLICADO NO XX SIMPEP

Título: IMPLEMENTANDO OS PARADIGMAS DA ENGENHARIA DE SOFTWARE COM O APOIO DA UML E DE SOFTWARE LIVRE

Title: IMPLEMENTING THE SOFTWARE ENGINEERING PARADIGMS SUPPORTED BY UML AND FREE SOFTWARE

Resumo: Implementar uma metodologia, técnicas ou mesmo padrões de conduta em uma empresa, além de ser uma tarefa trabalhosa, também pode custar muito financeiramente. Neste trabalho são abordadas ferramentas e técnicas que podem ser utilizadas gratuitamente por estudantes, profissionais ou empresas para implementar os paradigmas da engenharia de software. Nas próximas linhas são apresentadas ferramentas para engenharia de requisitos, qualidade de software, gerência de configuração, cronogramação, gestão de riscos, testes, métricas e modelagem visual, utilizando como base uma estação de trabalho com o sistema operacional gnu/linux fedora, instalado com as ferramentas de desenvolvimento a partir de uma mídia oficial. Conclui-se que é viável a implementação de tal ambiente, tornando mais produtivas as rotinas das empresas que não tem recursos financeiros para pagar por ferramentas proprietárias.

Abstract: To implement a methodology, techniques or even standards of best practices in a company, besides being a laborious task, may also cost a lot of money. This paper discusses tools and techniques which can be used free of charge by students, professionals or companies to implement the paradigms of software engineering. In the next lines are given information about requirements engineering, software quality assurance, configuration management, chronograph, risk management, testing, metrics and visual modeling using a workstation running the operating system gnu / linux fedora, installed with the development tools, from the official media. The conclusion is that it is possible to implement this environment, improving the production process of companies that do not have financial resources to pay for proprietary tools.

Palavras- Ambiente de desenvolvimento; software livre; código aberto

chaves:

Keywords: Development environment; free software; open source

Área: 8 - gestão do conhecimento organizacional

Sub-área: 8.4 - gestão de projetos

1. Introdução

As empresas gastam cada vez mais recursos com softwares, que nem sempre resolvem os problemas por não terem sido adequadamente escolhidos. O software funciona bem como ferramenta, mas é necessário que haja uma metodologia para utilizá-lo de forma produtiva. Mesmo quando há tal metodologia, o investimento pode ser tão alto que a empresa demora alguns anos para começar a obter os lucros. Muitas empresas, e até mesmo governos, instituições de ensino e as forças armadas estão aderindo ao software livre (GOVERNO, 2013; CIETEC, 2013).

No capítulo **Preparação do Ambiente** é abordada uma forma de instalação de uma distribuição do sistema operacional GNU/Linux, que será a base para a implementação das ferramentas estudadas no artigo.

Em seguida, no capítulo **Aplicação dos Paradigmas da Engenharia de Software**, os paradigmas são apresentados, e uma ferramenta ou metodologia é mostrada, visando implementar na prática os conceitos de cada item.

O produto final (conseguido com a colocação deste estudo em prática) é uma estação de trabalho capaz de aplicar os Paradigmas da Engenharia de Software, utilizando os padrões da UML e ferramentas livres.

2. Preparação do Ambiente

A distribuição Linux mais utilizada atualmente é o Red Hat Enterprise Linux, mas como ela não é gratuita, não é aplicável para este estudo (REDHAT, 2013). Foi escolhida a distribuição Fedora, por ser totalmente gratuita, ter uma vasta quantidade de software em seus repositórios oficiais, ter suporte a diversos idiomas, ser compatível com uma imensa gama de dispositivos de hardware, e finalmente por ter uma comunidade de desenvolvedores ativa. O processo de instalação

do sistema operacional não é o foco deste estudo, mas pode ser encontrado na documentação oficial do projeto. As ferramentas de desenvolvimento podem ser instaladas no momento da seleção dos pacotes, conforme exibido na imagem a seguir (FEDORA, 2013).

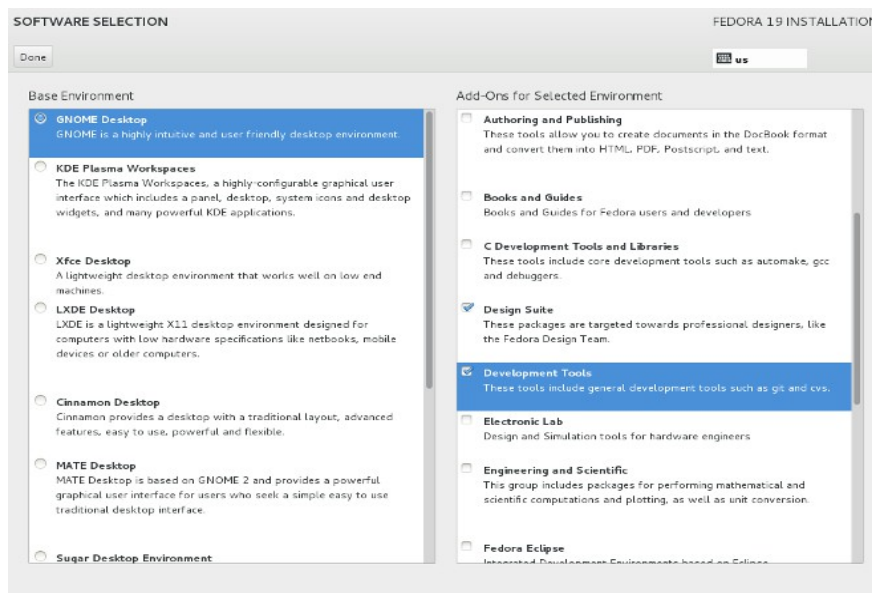


Figura 28 - Seleção de Software. Fonte: Reprodução de tela

3. Aplicação dos Paradigmas da Engenharia de Software

3.1 Engenharia de Requisitos

De acordo com Pressman (2006), o processo de elicitação de requisitos é a pedra fundamental do processo de desenvolvimento, pois ele garante que o produto final é exatamente o que foi solicitado pelo cliente. Os passos que definem a Engenharia de Requisitos são Elicitação de requisitos, Análise e negociação de requisitos, Especificação de requisitos, Modelagem do sistema, Validação de requisitos e Gestão de requisitos. Outros paradigmas, como a Qualidade, dependem diretamente da Engenharia de Requisitos.

A UML, através de diagramas, pode ser utilizada para expressar os requisitos, sejam funcionais ou não-funcionais. O Diagrama de Casos de Uso é uma excelente ferramenta para execução desta tarefa (LIMA, 2011).

Há várias ferramentas para confecção deste diagrama, e neste estudo será apresentado o Papyrus UML, que é um plugin para a IDE Eclipse, aqui utilizada como ferramenta de desenvolvimento (ECLIPSE, 2013).

O Papyrus está disponível no repositório oficial do Projeto Eclipse, para instalá-lo basta procurar pelo termo UML na janela de instalação de software, selecioná-lo e seguir com o processo

normal de instalação de plugins (PAPYRUS, 2013).

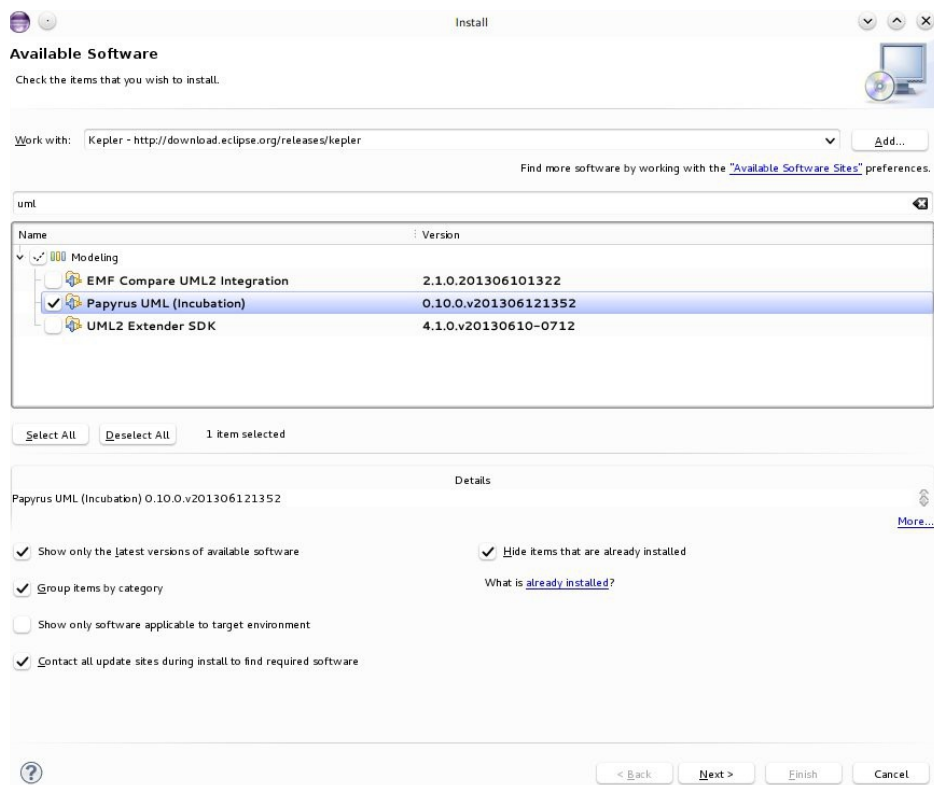


Figura 29 - Instalação do Papyrus. Fonte: Reprodução de tela

Depois de instalado, é possível iniciar um Diagrama de Casos de Uso através do processo de criação de um novo arquivo, seja pelo menu principal ou através do mouse, na pasta principal do projeto. O Diagrama de Casos de Uso elaborado no Papyrus pode ser exportado ou impresso, e a imagem a seguir ilustra sua exibição em tela.

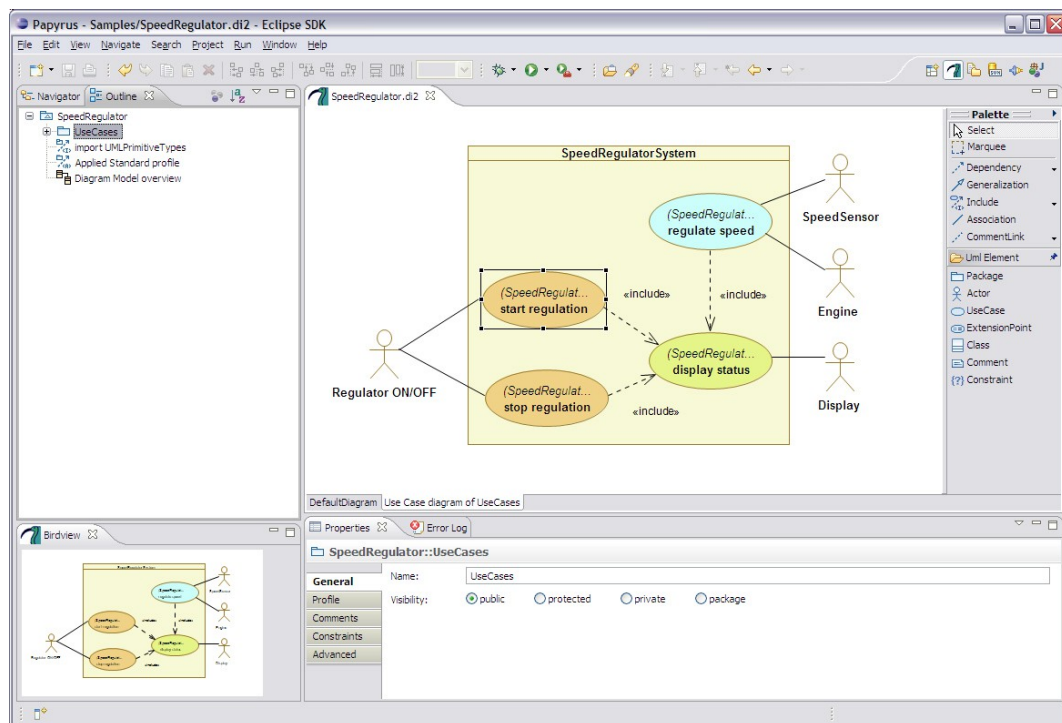


Figura 30 - Diagrama de Casos de Uso. Fonte: Reprodução de tela

3.2 Qualidade de Software

A prática de entregar o produto final com defeitos, e corrigi-los após o término do projeto não é mais aceita pelas empresas modernas, que buscam a qualidade, ou seja, receber aquilo que foi prometido que seria entregue (SOMMERVILLE, 2003).

Para garantir a qualidade no processo, a fase de elicitação dos requisitos é fundamental, pois a partir destes requisitos o acompanhamento do projeto será conferido e mostrará se a qualidade está se perdendo no decorrer do prazo (LIMA, 2011).

Além dos recursos encontrados na ferramenta Papyrus, vista no item anterior, anotações e atas de reuniões podem ser elaboradas com o LibreOffice, uma suíte de aplicativos gratuita, que pode ser instalada em sistemas Windows, Mac e Linux, e mantém um bom nível de compatibilidade com o MS-Office, ferramenta de escritórios mais utilizada na atualidade. Com o LibreOffice Writer, pode-se criar textos com uma rica formatação, utilizando recursos multimídia e de correção de ortografia. Há suporte para diversos idiomas, incluindo português, inglês, francês, espanhol e alemão (LIBREOFFICE, 2013).

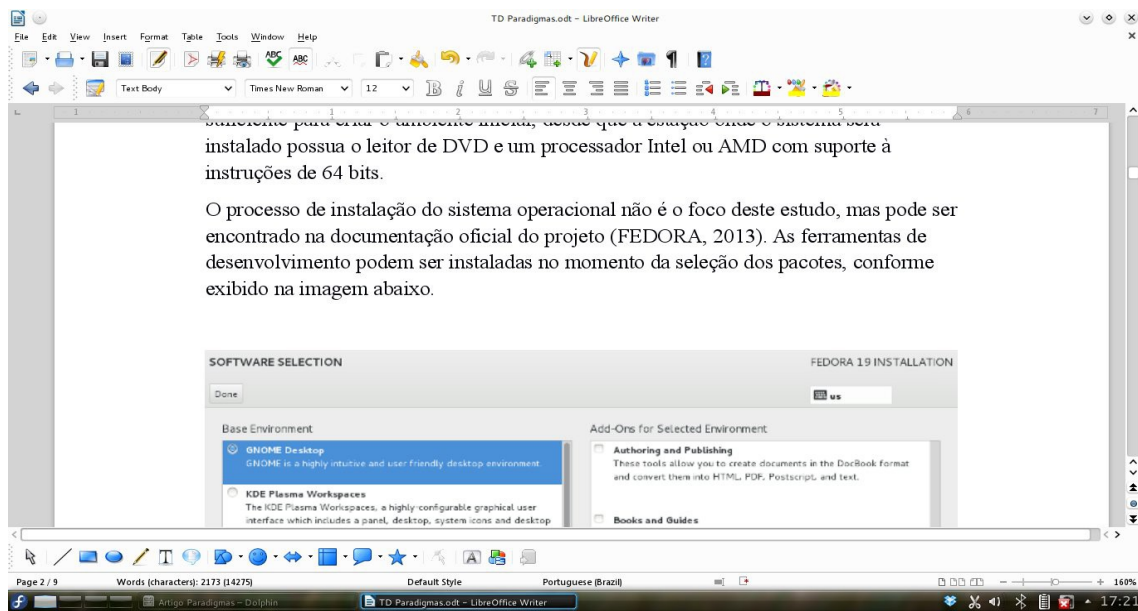


Figura 31 - Processador de Texto Writer. Fonte: Reprodução de tela

O LibreOffice Draw oferece recursos avançados para a criação de mapas, diagramas, ilustrações e trabalhos visuais (LIBREOFFICE, 2013).

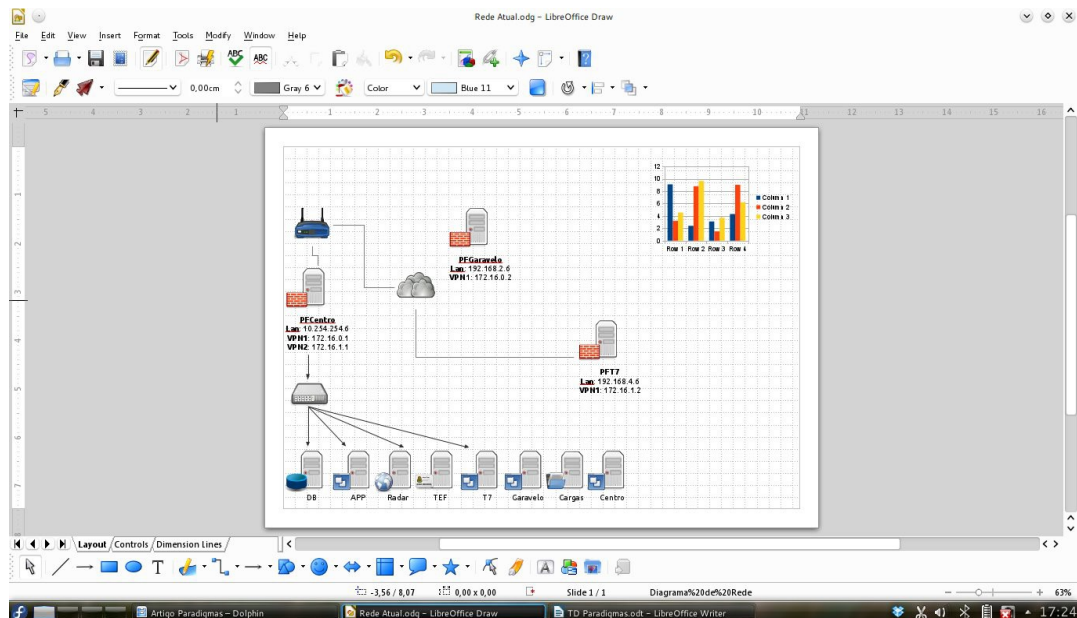


Figura 32 - Editor Gráfico Draw. Fonte: Reprodução de tela

3.3 Gestão da Configuração de Software

Durante o ciclo de desenvolvimento de um software, problemas são corrigidos, recursos são adicionados, mudanças são feitas, e é necessário manter o controle sobre estas alterações. Elas são

denominadas “versões”, e é muito comum ter que retomar alguma atividade que estava sendo executada em uma versão anterior do software, abandonando a versão atual. O VCS — sigla para Version Control System, ou Sistema de Controle de Versão — é o responsável por gerenciar as versões e suas operações mais importantes (SOMMERVILLE, 2003).

Em software livre é muito comum o desenvolvimento de aplicações por pessoas espalhadas por diferentes regiões, muitas vezes até em países diferentes. Este modelo colaborativo exige um controle de versões eficiente, rápido e distribuído. Linus Torvalds, criador do Linux, criou também uma ferramenta para controle de versões chamada GIT. O GIT é um SCV gratuito que acompanha a maioria das distribuições Linux, e pode ser utilizado em outros sistemas operacionais com a mesma eficiência (GIT, 2013; GITHUB, 2013).

Para facilitar o uso do GIT, empresas adotaram um modelo de desenvolvimento distribuído de projetos de código aberto, criando repositórios GIT na Internet, mediante um cadastro gratuito. Deste modo, os projetos são hospedados na infraestrutura destas empresas e podem ser acessados por qualquer usuário com o cliente GIT instalado. O mais utilizado atualmente chama-se GitHub. O acesso aos arquivos, versões e detalhes pode ser feito através de um navegador de Internet, sem dependência de plataformas.

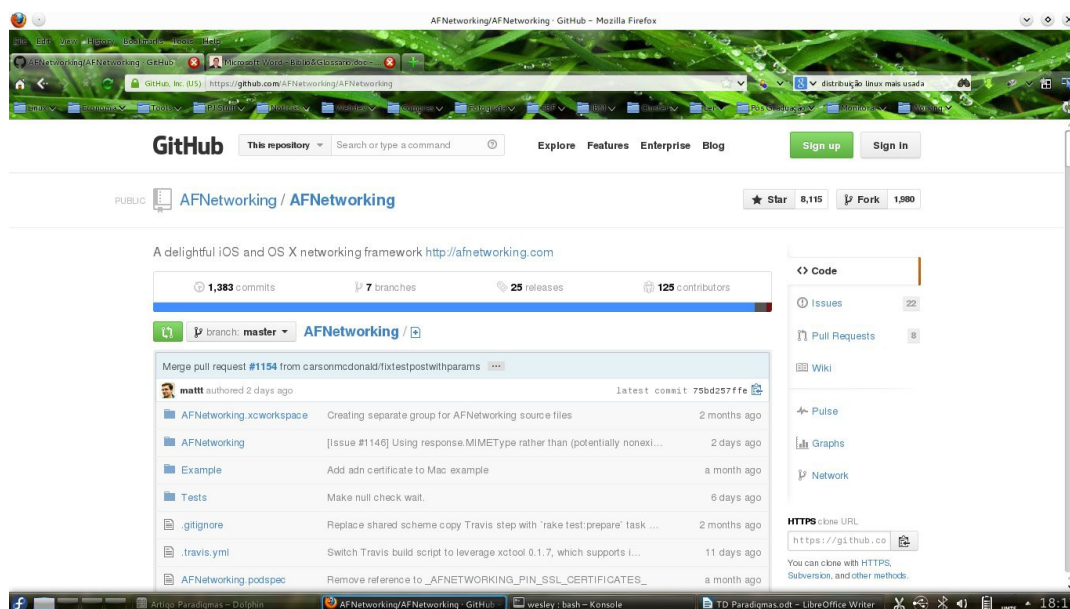


Figura 33 - Página de projeto no GitHub. Fonte: Reprodução de tela

3.4 Modelagem Visual

A modelagem visual é o processo de ilustrar através de modelos e imagens o processo intelectual do software, bem como seu processo de desenvolvimento. Através destes modelos, uma compreensão profunda do software em desenvolvimento pode ser atingida por pessoas em menos tempo se comparado a leitura de material impresso (NOGUEIRA, 2003).

A UML apoia a modelagem visual fornecendo uma padronização na representação dos processos (LIMA, 2013).

Existem diversas ferramentas para trabalhar com a modelagem visual, mas como neste estudo está sendo utilizada a IDE Eclipse, o foco será dado ao software Papyrus, utilizado anteriormente para elaboração do Diagrama de Casos de Uso.

O Papyrus oferece suporte para a criação dos diagramas mais comuns utilizados nos processos de Engenharia de Software.

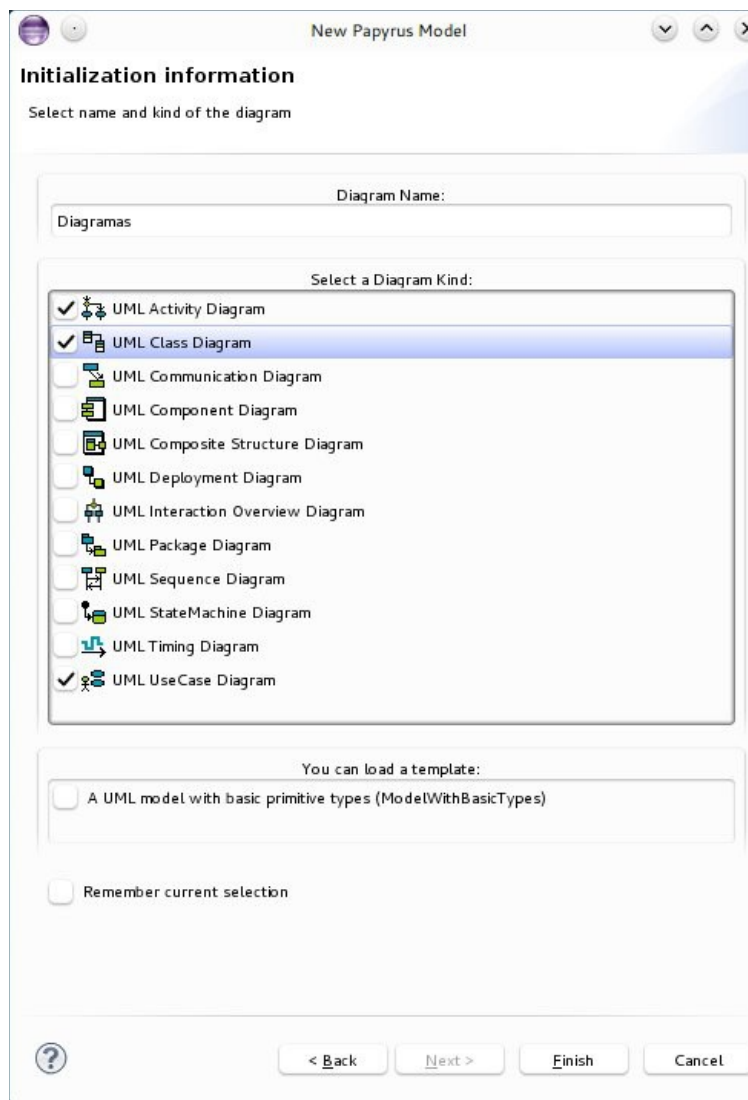


Figura 34 - Diagramas suportados. Fonte: Reprodução de tela

3.5 Métricas

É possível medir diretamente alguns fatores, como número de defeitos encontrados, número de correções aplicadas, número de alterações nos requisitos, número de linhas de código, e ainda é possível utilizar técnicas mais avançadas, como a análise de pontos de função, que através de cálculos específicos com pesos preestabelecidos, chega a uma quantidade de horas aproximadas a serem gastas no projeto (PRESSMAN, 2006).

Se houver um processo onde cada mudança, problema ou incidente seja registrado em uma ferramenta, um histórico rico será criado, e servirá para calcular diversas métricas. A ferramenta Mantis Bug Tracker atende a estes requisitos, além de ser totalmente livre. Os Bug Trackers — rastreadores de erros — são softwares utilizados primariamente para registro de solicitações de

correção, mas que podem ser utilizados para acompanhar o processo de desenvolvimento e como os erros são tratados.

Recursos como Roadmap, Changelog e exportação de relatórios para outros formatos estão disponíveis no Mantis, que pode ser visualizado na imagem abaixo (MANTIS, 2013).

ID	Version	Component	Severity	Status	Assignee	Date	Description
0018114		Other	major	assigned	(deepakhemani)	2013-07-24	Need to work on understanding DB architecture for JI Collaborative
0018113		Website	minor	assigned	(edarsaut)	2013-07-24	summary
0018106	1	GUI	minor	new		2013-07-24	Test bug report
0018102		GUI	minor	new		2013-07-24	asdfsadf
0018112	1	Other	minor	feedback	(mandce)	2013-07-24	test summary test
0018111	1	GUI	minor	assigned	(fariolol)	2013-07-24	dfg
0018104	3	GUI	minor	new		2013-07-24	login window is not looking good. this not user freindly.
0018109	1	GUI	minor	feedback	(bibi)	2013-07-24	Test
0018110		GUI	minor	new		2013-07-24	Test
0018108		GUI	minor	new		2013-07-24	Test bug report
0018107		GUI	minor	new		2013-07-24	Test bug report
0018105		GUI	minor	new		2013-07-24	Test bug report
0018099	4	Website	major	assigned	(smcheatham)	2013-07-24	Testing this issue
0018101		Other	major	assigned	(uloda)	2013-07-23	Just another bug report
0018098		GUI	minor	new		2013-07-23	this is a dummy summar
0018096	1	Other	minor	assigned	(manu1985)	2013-07-23	nothing is ok.
0018097		GUI	minor	new		2013-07-23	Teste mantis
0018094		Website	crash	confirmed	(Damandeep Singh)	2013-07-23	FAQ
0018095			trivial	assigned	(web3d)	2013-07-23	FEI

Figura 35 - Mantis exibindo uma lista de bugs. Fonte: Reprodução de tela

3.6 Cronogramação

A Cronogramação serve para estabelecer estimativas de tempo mais prováveis para tarefas individuais em um projeto, aplicando modelos estatísticos e determinando o caminho crítico. Ela calcula os limites de tempo para uma tarefa específica e as cadeias de tarefas que acarretam na duração total do projeto (PRESSMAN, 2006).

A cronogramação de projeto de software é semelhante à cronogramação de outras engenharias, e pode utilizar as já consolidadas técnicas Gantt, PERT e CPM.

O Planner é um software estável e de fácil utilização, que permite acompanhar individualmente as tarefas e ver seu gráfico Gantt. (PLANNER, 2013).

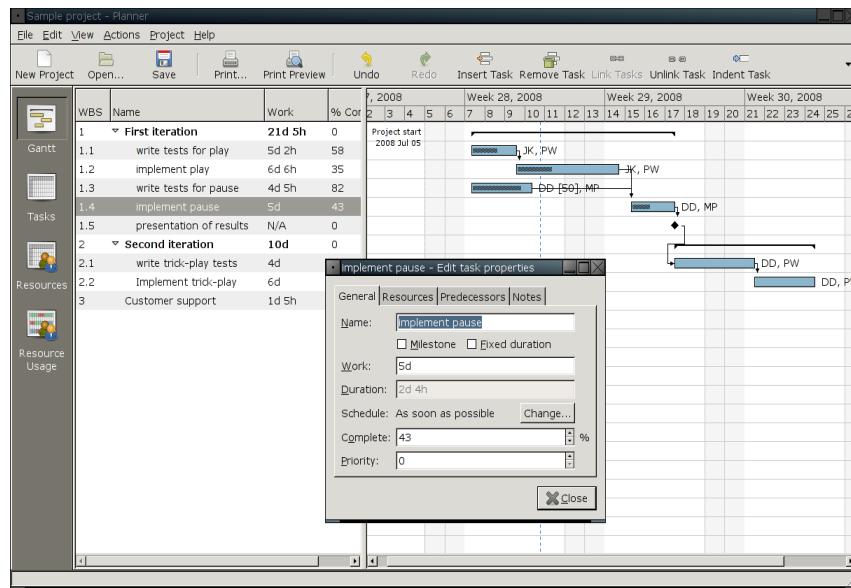


Figura 36 - Gráfico Gantt no Planner. Fonte: Reprodução de tela

Os gráficos PERT também podem ser implementados no ambiente proposto, utilizando o VYM. O Vym é uma ferramenta de mapas mentais gratuita, que acompanha as principais distribuições Linux, e pode ser utilizado na criação de gráficos PERT, além de ajudar na organização das ideias (VYM, 2013).

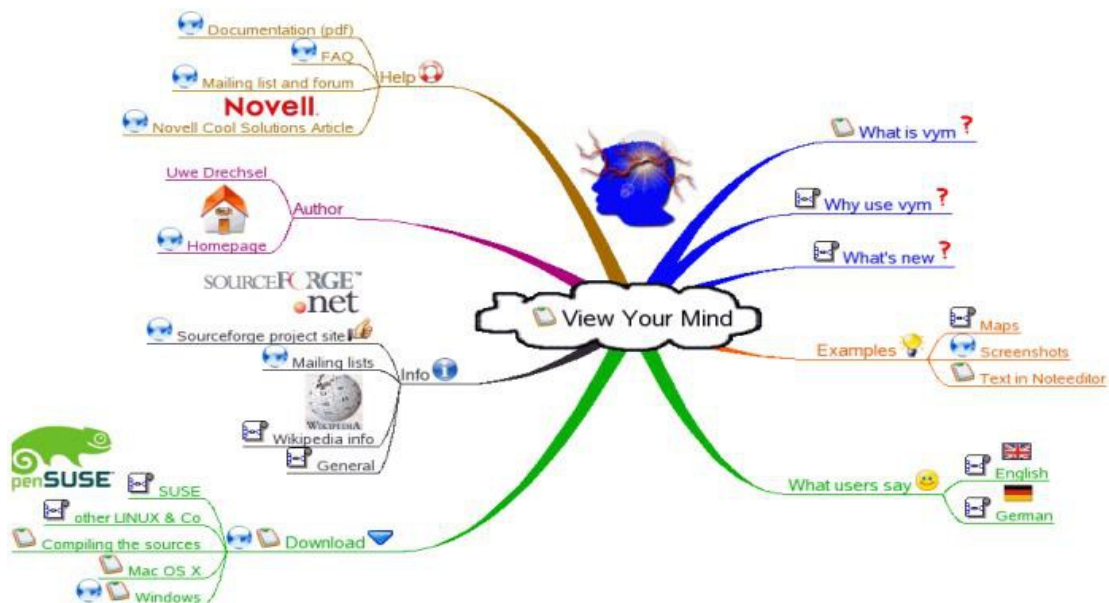


Figura 37: Mapa Mental no VYM. Fonte: Reprodução de tela

3.7 Testes

Uma fase muito importante no processo de desenvolvimento de software é a etapa de testes, na qual é analisado se o software cumpre as tarefas que foi designado a fazer, relatando sua qualidade no final do processo (PRESSMAN, 2006).

De acordo com a norma ISO 9126 (2004), a funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade são os atributos qualitativos para um software, que portanto devem fazer parte do cronograma de testes.

A realização de testes é, quase sempre, limitada por restrições de cronograma e orçamento; eles determinam quantos testes serão possíveis de executar. É importante que os testes sejam bem planejados e desenhados, para conseguir-se o melhor proveito possível dos recursos alocados para eles (FILHO, 2003).

Sendo uma etapa tão importante, é necessário que ferramentas apoiem as metodologias utilizadas pela equipe de qualidade, visando agilizar e profissionalizar o processo. Informações preciosas para gerar os planos de testes podem ser obtidas nos diagramas da UML, entre eles os requisitos, com o propósito de testar se todos eles foram cumpridos na implementação final do software.

Para a IDE estudada neste artigo, o JUnit é uma poderosa ferramenta para implementação de testes. Trata-se de um framework de código aberto com foco na criação de testes com apresentação de resultados. Com ele, é possível testar cada método de cada classe do software, de maneira isolada. Esta forma de testar os componentes é muito viável para descobrir pequenos erros no código.

Assim como o Eclipse, o JUnit é totalmente livre e pode ser encontrado em grande parte das distribuições Linux, bem como no site do projeto (JUNIT, 2013).



Figura 38 - Teste Unitário com JUnit. Fonte: Reprodução de tela

3.8 Gestão de Riscos

Existem riscos de diversas categorias, que a qualquer instante podem prejudicar ou mesmo interromper o projeto de software.

Barry W. Boehm representou os riscos em software sistematicamente através de um desenho em forma de espiral, que tem como princípio ser incremental. Por isso, uma das abordagens que ajudam na mitigação dos riscos é a de divisão em etapas pequenas a serem analisadas. Os software que vimos para Métricas e Cronogramação podem ser úteis no processo de break down das tarefas, já que é muito mais fácil analisar pequenas tarefas se comparadas ao projeto completo (PMI, 2004).

4. Conclusão

A análise mostra que é possível implementar processos razoáveis utilizando ferramentas que além de livres, em sua maioria tem código aberto, ou seja, permitem que o usuário adeque às suas necessidades.

Uma estação de trabalho Linux é um ponto de partida viável para projetos de desenvolvimento em java, pois conta com muitas ferramentas nativas para controle de processos e produtividade, automatização de tarefas e auxílio na produção de software de qualidade.

Os processos descritos neste estudo também oferecem aos estudantes a possibilidade de manter um laboratório de Engenharia de Software com possibilidades reais de aplicação prática, sem os custos que normalmente inviabilizam a experimentação de alguns aplicativos comerciais consolidados.

Referências

- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISO 9126: Modelo de Qualidade de Software*. Rio de Janeiro: ABNT, 2004.
- CIETEC. *Centro de Inovação, Empreendedorismo e Tecnologia*, 2013. Disponível em: <<http://www.cietec.org.br/noticia/13/ti-maior-tera-investimentos-de-r-500-milhoes>>. Acesso em: 18 jul 2013.
- ECLIPSE. *Eclipse IDE*, 2013. Disponível em: <<http://www.eclipse.org/>>. Acesso em: 18 jul 2013.
- FEDORA. *Projeto Fedora*, 2013. Disponível em: <<http://fedoraproject.org>>. Acesso em: 18 jul 2013
- FILHO, Wilson de Pádua Paula. *Engenharia de Software: fundamentos, métodos e padrões*. 2º ed. Rio de Janeiro: LTC, 2003.
- GIT. *Sistema de Controle de Versão Distribuído GIT*. Disponível em: <<http://git-scm.com/>>. Acesso em: 18 jul 2013.
- GITHUB. *Collaborative Project Management - GitHub*. Disponível em: <<https://github.com/>>. Acesso em: 18 jul 2013.
- GOVERNO. *Portal de Software Livre do Governo Federal*, 2013. Disponível em: <<http://www.softwarelivre.gov.br/comunidade-no-governo>>. Acesso em: 18 jul 2013
- JUNIT. *Junit: A programmer-oriented testing framework for Java*. Disponível em: <<http://junit.org/>>. Acesso em: 18 jul 2013.
- LIBREOFFICE. *LibreOffice – The Document Foundation*, 2013. Disponível em: <<http://pt-br.libreoffice.org/>>. Acesso em: 18 jul 2013.
- LIMA, Adilson. *UML 2.3 – Do Requisito à Solução*. São Paulo: Editora Érica, 2011.
- MANTIS. *Mantis: Gerenciador de Bugs de Software*. Disponível em: <www.mantisbt.org>. Acesso em: 18 jul 2013.
- NOGUEIRA, Marcelo. *Engenharia de Software: Um Framework para a Gestão de Riscos em Projetos de Software*. Rio

de Janeiro: Ed. Ciência Moderna, 2009.

PAPYRUS. *Papyrus UML Plugin*. Disponível em: <<http://www.papyrusuml.org/>>. Acesso em: 18 jul 2013.

PLANNER. *Planner Project Manager*, 2013. Disponível em: <<https://wiki.gnome.org/Planner>>. Acesso em: 18 jul 2013.

PMI. *A Guide to the Project Management Body of Knowledge - PMBOK*. Project Management Institute, 2004.

PRESSMAN, Roger S. *Engenharia de Software*. 5º ed. Rio de Janeiro: McGraw Hill, 2006.

REDHAT. *Red Hat Company*, 2013. Disponível em: <<http://br.redhat.com/>>. Acesso em: 18 jul 2013.

SOMMERVILLE, Ian. *Engenharia de Software*. 6º ed. São Paulo: Addison Wesley, 2003.

VYM. View Your Mind. 2013. Disponível em: <<http://sourceforge.net/projects/vym/>>. Acesso em: 18 jul 2013.

ANEXO IV – ARTIGO PUBLICADO NO WCSEIT 2013

COMO A EFICÁCIA DO PERSONAL SOFTWARE PROCESS PODE SER REFORÇADA E VALIDADA POR TEORIAS DA PSICOLOGIA E DA ADMINISTRAÇÃO DE EMPRESAS

Wesley Rodrigues da Silva¹, Marcelo Nogueira²

Abstract — *The Software Process Improvement are effective means to increase the productivity, quality and reduce development costs, especially the Personal Software Process, Team Software Process, and Capability Maturity Model. In this paper, the relationship between the Personal Software Process and known theories of Psychology and Business Administration are investigated, aiming for theoretical bases which explains and validates the process. Factors like resistance to change and organizational development are explained, confirming the applicability of the Personal Software Process and focusing the attention to the phases that may lead the implementation to failure.*

Keywords — *Software Process Improvement, resistance to changes, organizational development.*

INTRODUÇÃO

Muitas das técnicas e metodologias criadas em algumas áreas de estudo podem ser aplicadas em outras áreas com sucesso, se as devidas adaptações forem realizadas. Neste estudo, o risco de fracasso, os fatores de sucesso e os resultados da implantação do *Personal Software Process* em uma organização de desenvolvimento de software são explicados através de conceitos conhecidos e estudados há aproximadamente um século pela Administração de Empresas e pela Psicologia. Na seção **O CMM, o TSP e o PSP**, é feita uma

breve apresentação destes modelos, com um detalhamento maior do PSP, objeto de estudo deste artigo. Os objetivos e métodos de funcionamento são apresentados de forma resumida.

Em seguida, na seção **A generalização de conceitos da Administração e da Psicologia encontrados no PSP**, é apresentado um cenário básico de implementação de mudanças em organizações, de acordo com a Teoria Geral da Administração. As dificuldades enfrentadas nas fases iniciais do PSP também são abordadas nesta seção, com embasamento em conceitos da Psicologia. Ainda nesta seção, é apresentado o processo de mudança indicado pelo Desenvolvimento Organizacional, que é o estudo das interações dos indivíduos com a organização, do ponto de vista da Psicologia Empresarial.

O CMM, o TSP e o PSP

As empresas desenvolvedoras de software contam com diversos recursos para melhorar o controle de seus processos. O CMM — *Capability Maturity Model* — é um destes recursos, que apresenta as melhores práticas para diagnóstico e avaliação do controle de maturidade do desenvolvimento de software [1]. Porém, o CMM avalia a corporação de forma global, desconsiderando as características dos grupos e indivíduos. Para garantir que a corporação tenha um excelente processo de maturidade, é necessário garantir que as equipes e principalmente, que cada indivíduo tenha um alto nível de maturidade [1]. O TSP — *Team*

¹ Wesley Rodrigues da Silva, aluno do curso de pós-graduação em engenharia de software da Universidade Paulista, Rua Antonio de Macedo, 505 - Parque São Jorge - São Paulo - SP, Brazil, wesley.it@gmail.com

² Marcelo Nogueira, professor e coordenador do curso de pós-graduação em engenharia de software da Universidade Paulista, Rua Antonio de Macedo, 505 - Parque São Jorge - São Paulo - SP, Brazil, marcelo@noginfo.com.br

Software Process — tem a função de guiar as organizações na produção de software com elevado grau de qualidade, seguindo processos de maturidade e boas práticas para equipes, resultando em produtos finais confiáveis e com menor custo [2]. O PSP — *Personal Software Process* — é definido como um conjunto de práticas e métodos, que capacita os desenvolvedores de software a controlar sua rotina de trabalho, obtendo para o indivíduo os mesmos resultados que o TSP garante para equipes e o CMM para organizações [3].

Funcionamento do PSP

O PSP concentra-se em cada etapa do ciclo de desenvolvimento de software, visando:

- Entender realmente quanto tempo é gasto com o desenvolvimento, em cada etapa;
- Criar cronogramas mais realistas com base nas informações conhecidas;
- Impedir que o indivíduo se sobrecarregue com atividades impossíveis de realizar;
- Mostrar para cada indivíduo onde é possível aplicar melhorias.

Com o crescimento do indivíduo, as equipes passam a trabalhar de modo mais integrado, melhorando os resultados globais como consequência [3],[4].

Após implementar com sucesso o PSP, é possível perceber:

- Diminuição no número de defeitos: Ao analisar o histórico, o desenvolvedor consegue descobrir as causas e diminuir as ocorrências dos defeitos mais frequentes;
- Aumento no cumprimento das agendas: Sabendo quanto tempo leva cada etapa, prazos mais realistas podem ser projetados;
- Aumento no grau de qualidade do produto final: O desenvolvedor consegue isolar as distrações e problemas, atendendo os requisitos do software [3],[4].

A aplicação do PSP baseia-se na análise de informações coletadas através de formulários. Nestes formulários são registradas informações relacionadas a erros de programação e interrupções no trabalho, entre outros eventos.

Quando os resultados são analisados, a pessoa avaliada passa a conhecer sua situação no contexto do projeto, e com isso pode planejar uma estratégia visando aumentar a produtividade e corrigir os problemas detectados [3].

De fato, o PSP e o TSP podem ser considerados facilitadores para alcançar o CMM, que é um objetivo corporativo de larga escala e exige esforço coletivo [1],[2],[4].

No início da utilização do PSP percebe-se uma diminuição geral na produtividade, período no qual o indivíduo está absorvendo o processo. Este curto período cede lugar a um acentuado aumento na performance, que marca o domínio do processo pelos utilizadores [5].

Em contrapartida, não há um aumento nos erros encontrados no código; imediatamente após a adoção do PSP a taxa de erros inicia uma queda abrupta.

O PSP, criado em 1993, possui 20 anos de existência, e suas diretivas mostraram ótimos resultados nas empresas que implementaram os processos corretamente. Mas seriam os conceitos que servem como base para este modelo tão recentes quanto o próprio PSP? Será que há alguma explicação para as altas taxas de fracasso durante a implementação dos processos de melhoria de software [5]?

A GENERALIZAÇÃO DE CONCEITOS DA ADMINISTRAÇÃO E DA PSICOLOGIA ENCONTRADOS NO PSP

Frederick Taylor e os métodos científicos

A Administração Científica, criada no início do século XX por Frederick Winslow Taylor (1856-1915), revolucionou a gestão das empresas, melhorando diretamente a produtividade e lucratividade. Taylor foi o primeiro estudioso a analisar a empresa completamente, desde o funcionário da linha de produção ao presidente, criando padrões e rotinas para cada indivíduo [6].

Com a implementação dos métodos de Taylor, observou-se:

- Eliminação do desperdício de esforço humano e dos movimentos inúteis;
- Racionalização da seleção e adaptação dos operários à tarefa;
- Facilidade no treinamento dos

operários e melhoria da eficiência e rendimento da produção pela especialização das atividades;

- Distribuição uniforme do trabalho para que não haja períodos de falta ou excesso de trabalho;
- Definição de métodos e estabelecimento de normas para a execução do trabalho;
- Estabelecer uma base uniforme para salários equitativos e prêmios de produção [6].

É possível encontrar grandes semelhanças entre o PSP e os métodos propostos por Taylor, pois ambos se baseiam em medir a situação atual e tomar ações planejadas para melhorar os resultados [3],[6].

Desenvolvimento Organizacional

Na década de 1960, com as grandes mudanças econômicas relacionadas ao crescimento industrial, as organizações perceberam alterações no comportamento de seus funcionários. Os trabalhadores estavam preocupados com seu desempenho dentro da empresa, mas também com a realização de tarefas que realizassem seus anseios profissionais.

Estes fatos sociais forçaram os pesquisadores a desenvolver um novo modelo de gestão de empresas, chamado DO — Desenvolvimento Organizacional, que se tornou parte integrante da prática e da teoria da administração de empresas.

A palavra chave no DO é *mudança*. Com a imprevisibilidade dos ambientes corporativos, o DO passou a perceber que cada indivíduo é diferente e único, com especialidades e necessidades diferentes, criando planos de ação diferenciados levando estes fatos em consideração [7], [8],[9].

Uma organização contém três aspectos que definem toda a sua atividade: estrutura, tecnologia e comportamento. Qualquer um destes aspectos quando alterados, causam mudanças na organização em geral.

A estrutura envolve a hierarquia administrativa, o fluxo de comunicação e a definição da missão, objetivos e políticas.

A tecnologia compreende os sistemas

operacionais adotados, equipamentos, engenharia do processo e do produto, pesquisa e métodos de trabalho, etc.

O comportamento se refere aos processos de administração das pessoas.

Na Psicologia, a área responsável pelo comportamento ficou conhecida como Behaviorismo, ou seja, o conjunto de teorias que apontam o comportamento como objeto principal de estudo da Psicologia [8],[9].

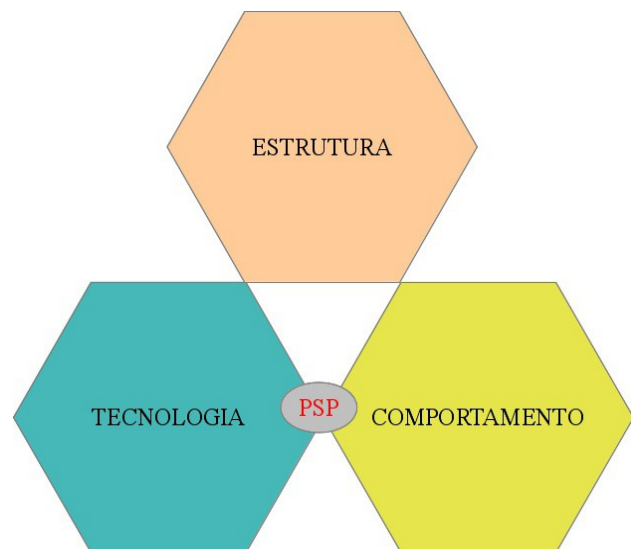


FIGURA. 1

ASPECTOS QUE DEFINEM AS ATIVIDADES DAS ORGANIZAÇÕES [13]

O PSP encontra-se entre a tecnologia e o comportamento, porque se trata de uma tecnologia e de um processo de trabalho, que ao mesmo tempo aumenta a produtividade e trata o comportamento pessoal, resultando em um aumento da disciplina pessoal e controle de tempo [3].

Resistência ao PSP e outros modelos de melhoria do processo de software

No Brasil e em outros países do mundo, foram realizados trabalhos de campo e pesquisas no intuito de entender as dificuldades mais comuns durante a implementação de processos de melhorias no desenvolvimento de software [10], [11],[12].

Os fatores relacionados aos funcionários (fatores humanos) são as causas mais frequentes atribuídas ao fracasso na implementação de projetos de melhoria de software, com destaque para:

- falta de adesão e participação por

parte do corpo operacional;

- falta de treinamento adequado à nova metodologia ou modelo;
- falta de comprometimento e liderança por parte dos líderes [10].

Brietzke e Rabelo [10] fizeram um estudo em 2006 buscando fatores de resistência durante as fases de implementação de processos de melhoria de software. Este estudo demonstra uma pesquisa de campo feita em diversas empresas da Europa.

As tabelas a seguir demonstram uma parte deste estudo, onde questionários foram respondidos pelos funcionários de áreas correlatas ao gerenciamento de projetos de software.

TABELA I

FATORES DESMOTIVADORES PARA DESENVOLVEDORES DE SOFTWARE [14]

Desmotivadores	Ocorrência no grupo de 21 pessoas	
	Frequência	(%)
Restrições de tempo	7	44
Falta de evidência de benefícios	6	38
Falta de recursos	5	31
Pressões comerciais	4	25
Inércia	4	25
Baixa prioridade	4	25
Processos pesados	2	13
Comunicação inadequada	2	13
Falta de apoio geral	2	13
Experiências negativas	2	13
Demissão da equipe	2	13
Incêndios	1	6
Imposição	1	6
Métricas inadequadas	1	6
Objetivos irrelevantes	1	6
Programas de larga escala	1	6

TABELA II

FATORES DESMOTIVADORES PARA GERENTES DE PROJETOS [15]

Desmotivadores	Ocorrência no grupo de 21 pessoas	
	Frequência	(%)
Restrições de tempo	13	62
Inércia	9	43
Restrições no orçamento	5	24
Processos pesados	5	24
Pressões comerciais	4	19
Falta de compromisso	4	19
Imposição	3	14
Carga de trabalho	3	14
Falta de padrões	2	10
Falta de feedbacks	2	10
Conflitos de personalidade	2	10
Comunicação inadequada	1	5
Clientes	1	5
Equipe sem experiência	1	5
Boas práticas isoladas	1	5
Falta de apoio geral	1	5
Experiências negativas	1	5
Falta de conhecimento técnico	1	5
Falta de criatividade	1	5

TABELA III

FATORES DESMOTIVADORES PARA GERENTES DE EQUIPES [16]

Desmotivadores	Ocorrência no grupo de 21 pessoas	
	Frequência	(%)
Falta de recursos	8	67
Restrições de tempo	7	58
Inércia	6	50
Falta de apoio geral	6	50
Experiências negativas	4	33
Falta de habilidades com SPI	4	33
Restrições no orçamento	3	25
Pressões comerciais	3	25
Equipe sem experiência	3	25
Comunicação inadequada	2	17
Processos pesados	1	8
Falta de evidência de benefícios	1	8
Mudanças organizacionais	1	8
Conflitos de personalidade	1	8

Das tabelas exibidas, a inércia e as experiências ruins somadas resultam em mais da metade dos problemas relacionados às barreiras de implementação, mostrando que há uma tendência inerente à resistência a mudanças por

parte dos indivíduos [10],[11],[12].

A resistência às mudanças como fenômeno da Psicologia e Administração de Empresas

O processo de mudança que ocorre com a implementação do PSP pode ser generalizado e explicado pela Teoria Geral da Administração, na área conhecida como Evolução das Teorias Administrativas.

A mudança comportamental ocorre em 3 fases: descongelamento, implantação e recongelamento.

No descongelamento, a necessidade de mudança é tornada tão aparente que já não é mais aceitável não efetuar a mudança. Em paralelo, no PSP, esta é a fase de anotação dos defeitos e interrupções, analisando o processo atual e chegando-se à conclusão que é necessário implementar melhorias.

A fase de implantação é onde as mudanças de comportamento são inseridas com base na análise dos dados da fase anterior, e ocorre do mesmo modo no PSP.

Finalmente, na fase de congelamento, o novo padrão de comportamento é consolidado, depois de comparar com os resultados anteriores e realizar novas e contínuas avaliações.

Das fases citadas, a que oferece as maiores dificuldades é o descongelamento. A insegurança e a ameaça, (entre outros itens exibidos na Figura 2), são alguns dos fatores que provocam resistência no indivíduo, que sente-se prejudicado com a mudança de ambiente por perceber que pode perder sua posição ou benefícios conquistados na organização [7],[11].



FIGURA. 2

FATORES QUE PROVOCAM A RESISTÊNCIA INDIVIDUAL [17]

A Psicologia estuda o fenômeno conhecido como resistência individual às mudanças, no qual a pessoa sente-se desconfortável com uma possível nova situação e cria barreiras visando permanecer numa situação que julga ser mais segura. É um processo intrínseco do ser humano, que ocorre com maior intensidade em alguns indivíduos que em outros [10],[11].

O Instituto Tavistock, localizado na Inglaterra, produziu com uma equipe de antropólogos, psicólogos e psiquiatras diversos programas em seleção de pessoal, tratamento de neuroses de guerra e reabilitação de prisioneiros. Estas experiências originaram um método conhecido como pesquisa-ação, que se baseia na colaboração dos integrantes de uma mesma organização objetivando resolver seus próprios problemas — resolvendo os problemas da organização de maneira indireta [7].

A pesquisa-ação requer envolvimento do pesquisador no processo de ação, e é diferente das outras variedades de pesquisa, pois fornece uma abordagem e um processo de geração e uso de informações que originam o programa de ação a ser adotado pela organização. Métodos como a pesquisa-ação podem ser utilizados para implementar mudanças, como os processos de melhoria de software [7].

CONCLUSÃO

Apesar de ser um modelo com relativamente poucos anos de existência, os estudos que levaram à criação do PSP se iniciaram há quase

um século, com Taylor e outros estudiosos da Administração, e graças ao desenvolvimento da Psicologia, entendemos as dificuldades encontradas na implantação do processo.

Assim como os outros processos e metodologias, para que o PSP seja corretamente implementado, os passos necessários para que haja total aderência por parte dos funcionários devem ser pontos críticos e pré-requisitos.

As barreiras impostas às mudanças podem ser transpassadas com comunicação e um processo transparente de implementação.

No primeiro instante, o PSP pode assemelhar-se com uma ferramenta de aferição de performance para fins comparativos e classificatórios de funcionários, e esta característica pode assustar os envolvidos no processo. É necessário deixar claro que não se trata de uma ferramenta de classificação e punição, e sim de uma ferramenta de percepção, análise e correção dos mais variados comportamentos.

Vencidas estas barreiras, a empresa passa a apresentar melhores resultados, com custos reduzidos e satisfação por parte dos empregados.

REFERÊNCIAS

- Capítulo 1. Pressman, R. "Software Process Improvement", In: *Software Engineering: A Practitioner's Approach*, 7. ed., São Paulo: McGraw-Hill Higher Education, 2009. Cap. 30, p. 786-795.
- Capítulo 2. Humphrey, W.; Chick, T.; Nichols, W.; Pomeroy-Huff, M. *Team Software Process (TSP): Body of Knowledge (BOK)*, Carnegie Mellon University: Software Engineering Institute, 2010.
- Capítulo 3. Pomeroy-Huff, M.; Cannon, R.; Chick, T.; Mullaney, J.; Nichols, W. *The Personal Software Process (PSP): Body of Knowledge (BOK), Version 2.0*, Carnegie Mellon University: Software Engineering Institute, 2009.
- Capítulo 4. Pressman, R. "Process Models", In: *Software Engineering: A Practitioner's Approach*, 7. ed. São Paulo: McGraw-Hill Higher Education, 2009. Cap. 2, p. 53-62.
- Capítulo 5. Baddoo, N.; Hall, T. "De-motivators for software process improvement: an analysis of practitioners views", *The Journal of Systems and Software*, v. 66, 2003, p. 23-33.
- Capítulo 6. Chiavenato, I. "Administração Científica: Arrumando o Chão da Fábrica", In: *Introdução à teoria geral da administração: uma visão abrangente da moderna administração das organizações*, 7. ed. rev. e atual. Rio de Janeiro: Elsevier, 2003. Cap. 3, p. 53-77.
- Capítulo 7. Ferreira, A.; Reis, A.; Pereira, M. "Desenvolvimento Organizacional", In: *Gestão Empresarial: de Taylor aos nossos dias*, São Paulo: Pioneira Thomson Learning, 2002. Cap 7, p. 66-82.
- Capítulo 8. Ferreira, A.; Reis, A.; Pereira, M. "Behaviorismo", In: *Gestão Empresarial: de Taylor aos nossos dias*, São Paulo: Pioneira Thomson Learning, 2002. Cap. 4, p. 40-46.
- Capítulo 9. Pinheiro, T. *Comportamento Organizacional*, São Paulo: Universidade Nove de Julho, 2008.
- Capítulo 10. Brietzke, J.; Rabelo, A. "Resistance factors in software processes improvement", *Clei Electronic Journal*, Canoas, v. 9, n. 1, p. 4, jun. 2006.
- Capítulo 11. Hernandez, J.; Caldas, M. "Resistência à mudança: uma visão crítica", *Revista de Administração de Empresas FGV*, São Paulo, v. 41, n. 2, p. 31-45, jun. 2001.
- Capítulo 12. Virtanen, P.; Pekkola, S.; Paivarinta, T. "Why SPI Initiative Failed: Contextual Factors and Changing Software Development Environment", *System Sciences (HICSS)*, 2013 46th Hawaii International Conference, p. 4606-4615.
- Capítulo 13. Figura 1 - "Aspectos que definem as atividades das organizações", Elaborada pelos autores com base nas informações de Ferreira, A.; Reis, A.; Pereira, M. "Desenvolvimento Organizacional", In: *Gestão Empresarial: de Taylor aos nossos dias*, São Paulo: Pioneira Thomson Learning, 2002. Cap 7, p. 66-82.
- Capítulo 14. Tabela I, "Fatores desmotivadores para desenvolvedores de software", Brietzke, J.; Rabelo, A. "Resistance factors in software processes improvement", *Clei Electronic Journal*, Canoas, v. 9, n. 1, p. 4, jun. 2006.
- Capítulo 15. Tabela II, "Fatores desmotivadores para gerentes de projetos", Brietzke, J.; Rabelo, A. "Resistance factors in software processes improvement", *Clei Electronic Journal*, Canoas, v. 9, n. 1, p. 4, jun. 2006.
- Capítulo 16. Tabela III, "Fatores desmotivadores para gerentes de equipes", Brietzke, J.; Rabelo, A. "Resistance factors in software processes improvement", *Clei Electronic Journal*, Canoas, v. 9, n. 1, p. 4, jun. 2006.
- Capítulo 17. Figura 2, "Fatores que provocam a resistência individual", Ferreira, A.; Reis, A.; Pereira, M. "Desenvolvimento Organizacional", In: *Gestão Empresarial: de Taylor aos nossos dias*, São Paulo: Pioneira Thomson Learning, 2002. Cap 7, p. 66-82.

REFERÊNCIAS

- AUTISMO E REALIDADE. **O que é o autismo**. Organização Autismo e Realidade. Disponível em: <<http://autismoerealidade.org/informe-se/sobre-o-autismo/o-que-e-autismo/>>. Acesso em: 05 mai. 2014.
- BROWN UNIVERSITY. **A History of Computer Programming Languages**. Brown University Website. Disponível em: <http://cs.brown.edu/~adf/programming_languages.html>. Acesso em: 10 abr. 2014.
- VARELLA, Dráuzio. **Autismo Infantil**. Site do médico Dráuzio Varella. Disponível em: <<http://drauziovarella.com.br/crianca-2/autismo/>>. Acesso em: 05 mai. 2014.
- ABP. **Classificações Diagnósticas**. Associação Brasileira de Psiquiatria. Disponível em: <http://www.abpbrasil.org.br/boletim/exibBoletim/imprimir.php?boltex_id=27&bol_id=7>. Acesso em: 07 mai. 2014.
- ALBERT EINSTEIN. **A Síndrome de Asperger**. Hospital Albert Einstein. Disponível em: <<http://www.einstein.br/einstein-saude/pagina-einstein/Paginas/entendendo-a-sindrome-de-asperger.aspx>>. Acesso em: 6 mai. 2014.
- APA - American Psychiatric Association. **DSM-IV**: Diagnosis and Statistical Manual of Mental Disorders. 4. ed. Washington DC: APA, 2000
- APA American Psychiatric Association. **DSM-IV**. Disponível em: <<http://www.psych.org/practice/dsm>>. Acesso em: 07 mai. 2014.
- ARTERO, Almir Olivette. **Inteligência Artificial: Teórica e Prática**. São Paulo: Livraria da Física, 2009.
- CARDOSO, Ana Patrícia Simões. **Síndrome de Asperger: Qualidade de Vida e Rendimento Escolar na Adolescência**. Dissertação para obtenção do Grau de Mestre em Medicina - Universidade da Beira Interior. Covilhã - Portugal, Outubro de 2011.
- CDC KF. **Key Findings**: Population Attributable Fractions for Three Perinatal Risk Factors for Autism Spectrum Disorders, 2002 and 2008 Autism and Developmental Disabilities Monitoring Network. Disponível em: <<http://www.cdc.gov/ncbddd/autism/features/keyfindings-risk-factors.html>>. Acesso em: 06 mai. 2014.

- CDC: **Autism Numbers and Facts:** Autism in United States. Disponível em: <http://www.cdc.gov/ncbddd/autism/index.html>>. Acesso em: 06 mai. 2014.
- COPPIN, Ben. **Inteligência Artificial.** Rio de Janeiro: LTC, 2012.
- DE SANTI, Alexandre. **As mentiras do seu cérebro.** Super Interessante, São Paulo: Abril. n. 315. p. 54, jun. 2012.
- FREEMAN, Steve. **Desenvolvimento de Software Orientado a objetos:** Guiado por Testes. Rio de Janeiro: Alta Books, 2012.
- INÁCIO, João. Lógica Paraconsistente. Site do Professor João Inácio. Disponível em: <http://paralogike.com.br/site/links/ver/28>>. Acesso em: 07 mai. 2014.
- LIMA, Adilson da Silva. **UML 2.3:** Do Requisito à Solução. 1. ed. São Paulo: Érica, 2013.
- NOGUEIRA, Marcelo; ABE, Jair. **Paradigmas da Engenharia de Software Como Fatores Críticos de Sucesso Para a Gestão de Projetos no Contexto Brasileiro.** In: SIMPEP, n. 17, 2010, Bauru.
- NOGUEIRA, Marcelo. **Engenharia de Software:** Um Framework para a Gestão de Riscos em Projetos de Software. Rio de Janeiro: Ciência Moderna, 2009.
- OMS CID-10. **Centro Colaborador da OMS para a Classificação de Doenças em Português - 2008.** Disponível em: <http://www.datasus.gov.br/cid10/V2008/cid10.htm>>. Acesso em: 07 mai. 2014.
- PLANALTO. **Acesso à informação do Planalto Federal.** Disponível em: <http://www2.planalto.gov.br/excluir-historico-nao-sera-migrado/lei-reconhece-direito-das-pessoas-com-autismo-a-educacao-e-ao-ensino-profissionalizante-entre-outras-conquistas>>. Acesso em: 06 mai. 2014.
- PRESSMAN, Roger S. **Engenharia de Software:** Uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.
- RUSSELL, Stuart; NORVIG, Peter. **Inteligência Artificial.** 2ª. ed. Rio de Janeiro: Campus, 2004.
- SILVA, Ivan Nunes; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. **Redes Neurais Artificiais:** para engenharias e ciências aplicadas. São Paulo: Artliber, 2010.CDC
- SOMMERVILLE, Ian. **Engenharia de Software.** 9. ed. São Paulo: Pearson, 2011.

- STANFORD UNIVERSITY. **The Challenge:** Simulating a million neurons. Disponível em: <<http://www.stanford.edu/group/brainsinsilicon/challenge.html>>. Acesso em: 13 abr. 2014.
- WHO - World Health Organization. **The ICD-10 classification of mental and behavioural disorders:** diagnostic criteria for research. Geneva: WHO, 1993.

LEITURA ADICIONAL UTILIZADA

- BRANDÃO, Marcus Lira. **As bases biológicas do comportamento:** Introdução à neurociência. São Paulo: Pedagógica Universitária, 2004.
- CASTRO, Leandro Nunes. **Computação Natural:** uma jornada ilustrada. São Paulo: Livraria da Física, 2010.
- DAWKINS, Richard. **O Relojoeiro Cego:** A teoria da evolução contra o desígnio divino. São Paulo: Companhia das Letras, 2001.
- DIVERIO, Tirajú Asmuz; MENEZES, Paulo Blauth. **Teoria da Computação:** máquinas universais e computabilidade. 3. ed. Porto Alegre: Bookman, 2011.
- KOJIMA, Hiroyuki; TOGAMI, Shin. **Guia mangá de cálculo:** diferencial e integral. São Paulo: Novatec, 2010.
- LUDWIG, Oswaldo; MONTGOMERY, Eduard. **Redes Neurais:** Fundamentos e Aplicações com Programas em C. Rio de Janeiro: Ciência Moderna, 2007.
- NUNES, Dalto. **Introdução à abstração de dados.** Porto Alegre: Bookman, 2012.
- TOSCANI, Laira Vieira; VELOSO, Paulo. **Complexidade de Algoritmos.** 3. ed. Porto Alegre: Bookman, 2012.