**ChatListen**

**DETECÇÃO DE ASSEDIADORES SEXUAIS NO CHAT DO FACEBOOK**

**Plano de Gerência de Configuração**

São José dos Campos, 12 de setembro de 2015.

Plano de Gerenciamento de Configuração

|  |  |
| --- | --- |
| **Equipe** | **E-mail** |
| Luiz Carlos Farias da Silva | Luiz.silva116@fatec.sp.gov.br |
|  |  |

**Índice**

Plano de Gerenciamento de Configuração

1. Introdução

O Plano de Gerenciamento de Configuração descreve as atividades do Gerenciamento de Controle de Configuração e Mudança que serão executadas durante o ciclo de vida do produto. As atividades se resumem em identificar a configuração do software, manter a integridade durante o andamento do projeto e controlar mudanças.

* 1. **Finalidade**

Este documento tem como finalidade criar um padrão para que seja seguido por todos os membros da equipe e tem como meta garantir um maior controle do produto durante o projeto. Serão detalhados os recursos necessários, as responsabilidades e o cronograma de atividades.

O projeto desenvolvido foi o ChatListen – um sistema para detecção de assediadores sexuais no chat do Facebook. O referido sistema tem a finalidade de promover meios para a detecção de assediadores sexuais que usam o chat do Facebook como meio de comunicação.

* 1. **Escopo**

O ChatListen é destinado aos pais que tenham a intenção de monitorar as conversas de seus filhos em relação à possível ocorrência de assediadores sexuais ou pedófilos sem que seja necessário ter acesso ao conteúdo das conversas.

Outra abrangência seria a própria investigação criminal, ou seja, a monitoração permite verificar a presença ou não de assediadores sexuais ou pedófilos usando o chat do Facebook quando já se tem, por alguma outra forma, suspeitas sobre o a possibilidade da ocorrência do assédios ou da pedofilia.

1. Modelo de Desenvolvimento

O projeto foi baseado no modelo SCRUM por se tratar de um método reconhecidamente eficaz no que tange ao gerenciamento das mudanças e às definições de prioridades, sempre com vistas ao valor de negócio para o cliente. O controle de versões do software foi realizado com o sistema GitHub (repositório/cliente git).

O model SCRUM facilita muito o desenvolvimento por não ser hierarquizado e por ser flexível com relação às atribuições das atividades. Tem-se também como foco a qualidade do projeto sempre.

1. Gerenciamento de Configuração de Software

### Papéis na Gerencia de Configuração.

|  |  |  |
| --- | --- | --- |
| **Papéis** | **Responsabilidade** | **Perfil** |
| 1 SCRUM Master | * Criar as **baselines** do projeto; * Assegurar que as tarefas exigidas sejam cumpridas; * Mediar a comunicação do time de desenvolvimento com o Product Owner - PO. * O SCRUM master tem a importante função de interfaceamento também, protegendo a equipe contra as adversidades que surgem no decorrer do projeto * Criar a documentação do projeto; * Assegurar o bom funcionamento do Scrum; * Promover melhorias técnicas no ambiente. | * Experiência em Métodos Ágeis como Scrum, XP, etc. * Superior completo na área de TI (ciências/engenharia da computação); * Certificação em SCRUM ; |
| 1 Product Owner | * Controlar Backlog **–** as user stories * Verificar junto ao cliente os requisitos funcionais do sistema; * Definir prioridades para os requisitos; * Testes de aceitação junto ao cliente; * Deixar o cliente ciente do que esta ocorrendo no projeto; * Analise de custos; | * Experiência em engenharia de Software, documentação de Sistemas; * Superior completo ou cursando ciências/engenharia da computação ou sistemas de informação; * Conhecimento na ferramenta starUml; * Facilidade e desenvoltura no relacionamento com clientes |
| 1 DBA | * Modelagem do banco de dados; * Implementação do banco de dados; * Manter a integridades das informações do banco de dados; * Backup e restore; | * Formação superior em Banco de Dados; * Experiência como DBA; * Certificação em MySQL \ PLSQL |
| 2 Desenvolvedores | * Desenvolvimento de diagramas de classe; * Desenvolvimento de diagramas de sequência; * Desenvolvimento do código fonte; * Desenvolvimento de testes unitários em conjunto com responsável pelos testes; * Deployment | * Experiência em Java, SQL, JSP, HTML&CSS, Bootstrap, Git, * Conhecimento na ferramenta Eclipse Luna; * Conhecimento em padrões de projeto. |
| 1 Tester | * Realização de testes unitários; * Criação de testes automatizados; | * Desenvolvedor Pleno Java; * Experiência com framework JUnit; * Experiência em integração contínua. * Experiência em Java |

**Tabela 1: Responsáveis e suas responsabilidades.**

### Ferramentas, ambientes e infraestrutura.

### Ferramentas para gerenciamento

|  |  |  |  |
| --- | --- | --- | --- |
| **Ferramenta** | **Descrição** | **Versão** | **Licença** |
| GitHub | Sistema de Controle de Versão  Repositório Público Gratuito  github.com/fariaslcfs/ChatListen | 2.5.2 | Free para respositório público |
| BitBucket | Sistema de Controle de Versão  Repositório Particular Gratuito  bitbucket.org/fariaslcfs/chatlisten | web | free |
| Trello | Gerenciador de projeto  Quadro organizador de tarefas  trello.com/timepgctg | 2015 | Free |

### Tabela 2: Ferramentas para gerenciamento.

### Ferramentas para o ambiente de desenvolvimento:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo** | **Ferramenta** | **Versão** | **Licença** |
| Sistema Operacional | Windows 10 | 10.0 | OEM |
| Banco de Dados | MySQL | 5.6.24-log MySQL Community Server | GPL |
| BrModelo | 3.0 | Free |
| Controle de Versão | Git | 2.4.0 | Free |
| GitHub | web | Free repositório gratuito |
| BitBucket | web | Free repositório  particular e público |
| Plataforma de Desenvolvimento | IDE: Eclipse Luna | 4.4 | Free |
| Gerência de Build e ependência  Maven | 3.2.5 | Free |
| Linguagem backend: Java 8 | 1.8.0\_45 | Free |
| ORM: EclipseLink | 4.3.8 | Free |
| Testes unitários: JUnit | 4.12 | Free |
| Comunicação | whatsapp | 5.2.0.6 | 2.12.250 |
| Processador de texto | Microsoft Office | 2010 | Professional |
| Diagramação do sistema | Star UML | 5.0 | Free |
| Gerenciamento de tarefas | Trello | web | Free |

**Tabela 3: Ferramentas para desenvolvimento.**

### Estrutura do ambiente

|  |  |  |
| --- | --- | --- |
| **Ambiente** | **Descrição** | **Transição** |
| Banco de Dados | É o ambiente onde conterá o Banco de dados. | Transição se dará com a configuração total da base, incluindo as inserções necessárias para o funcionamento dos módulos subsequentes. |
| Desenvolvimento | Ambiente para o desenvolvimento do Sistema. | A transição ocorrerá assim que os requisitos forem cumpridos e os testes forem aceitos pelos desenvolvedores e pelo Tester. |
| Homologação | Ambiente suporte para os testes de integração. | A transição ocorrerá quando a integração dos módulos for bem sucedida. |

**Tabela 4: Estrutura do ambiente.**

### Configuração dos computadores dos ambientes

|  |  |  |  |
| --- | --- | --- | --- |
| **Qtd** | **Ambientes** | **Configuração Hardware** | **Configuração Software** |
| 2 | Desenvolvedor | Processador Intel Core i5 3ª G n  Memória RAM 6GB  HD 500GB | Windows 7  Eclipse Luna  Java JDK 8  Java JRE 8  Git/eGit-Eclipse  StarUml 5.0  Microsoft Office 2010 Professional  Windows Defender |
| 1 | DBA | Processador Intel Core i7 3ª G  Memória RAM 8GB  HD 1TB | Windows 7  Eclipse Luna  Java JDK 8  Java JRE 8  Git  StarUml 5.0  Microsoft Office 2010 Professional  Windows Defender |
| 1 | SCRUM Master | Processador Intel Core i5 3ª G  Memória RAM 4GB  HD 500GB | Windows 7  Eclipse Luna  Java JDK 8  Java JRE 8  Git  StarUml 5.0  Microsoft Office 2010 Professional  Windows Defender  Trello 2015 |
| 1 | Tester | Processador Intel Core i5 3ª G  Memória RAM 4GB  HD 500GB | Windows 7  Eclipse Luna  Java JDK 8  Java JRE 8  Git/eGit-Eclipse  StarUml 5.0  Microsoft Office 2010 Professional  Windows Defender  Plugin Selenium Expert (Selenium IDE) 0.25.1-signed |
| 1 | Product Owner | Processador Intel Core i5 3ª G  Memória RAM 4GB  HD 500GB | Windows 7  Eclipse Luna  Java JDK 8  Java JRE 8  Git  StarUml 5.0  Microsoft Office 2010 Professional  Windows Defender  Trello 2015 |

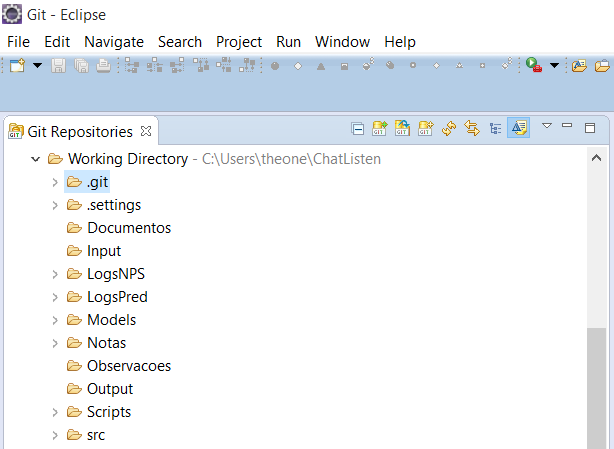
**Tabela 5: Configuração das maquinas dos ambientes.**

1. Plano de Configuração

### Versionamento

### O controle de versionamento será efetuado por meio do sistema git (cliente / repositório) tendo o *github.com* e o *bitbucket.com* com diretórios remotos, sendo o github como o origin e o bitbucket como diretório secundário com serviço gratuito de diretório particular até 5 usuários.

### Estrutura de diretórios

A estrutura de diretórios será a do próprio projeto Eclipse compartilhada com toda a equipe por meio do cliente git (shell) e eGit no Eclipse. A estrutura é apresentada na figura abaixo.

Os diretórios Input, LogsNPS, LogsPred, Models, Scripts, Notas e Observacoes foram criados para, respectivamente: armazenar arquivos gerais de entrada no projeto que ainda não estão categorizados; armazezar os logs de não predadores categorizados por faixa etária; armazenar os logs dos predadores baixados de perverted-justice.com e já tratados; armazenar os modelos de PLN usados no projeto; armazenar os scripts criados como ferramentas para auxílio no projeto; armazenar notas úteis aos membros da equipe; e armazenar observações que devem ser analizadas.

Serão excluidos, por meio do .gitignore, a pasta target, os arquivos .class e os arquivos .exe. Os arquivos constantes do diretório *Notas* serão nomeados da seguinte forma: **Nota-<dd-mm-aaaa>.txt** e os arquivos do diretório *Observacoes* serão nomeados da seguinte forma: **Observação-<dd-mm-aaaa>.txt**. Os demais diretórios são os usuais da estrutura Maven para source e resources, os de configuração do git e os de configuração do projeto Eclipse.

### Nomenclatura das versões

O processo de nomeação das versões obedecerá o seguinte formato e critério:

versão **N1.N2.N3.N4**, onde:

**N1** = número da versão maior, ou seja, é o número da versão de **mudanças grandes** e, por isso, são versões que mudam com menor frequência;  
**N2** = número de versão menor. É o número das mudanças menos significativas, como exemplo pode-se citar as atualizações comuns;

**N3** = número da liberaçã ou ***release.***

**N4** = número da ***Build***, a versão da compilação do programa, ou seja, o número de vezes que ele foi compilado até que se chegue a versão atual.

### Política de branch

As branches criadas serão a master (padrão do sistema) que será a branch de homologação e será criada a branch desenvolvimento. Neste ramo será efetuado o desenvolvimento do sistema. Serão criadas novas branches para investigação de bugs encontrados depois de pontos de commit. Essas branches serão nomeadas por **bugfix-commit-<nome do último commit antes da ocorrência do bug>** e serão fundidas à branch desenvolvimento assim que o bug for resolvido.

### Política de commit

### Política de Merge

### Baselines

Master: Contém a estrutura do item já definida. Tem como finalidade receber todos os itens. A equipe armazena nesta pasta todas as versões dos documentos e códigos.

Branch: Armazena documentos de uma versão que esta sofrendo uma mudança diferente da linha normal de desenvolvimento.

Tag: Armazena as configurações bases do projeto. Estes itens de configuração representam versões base e não sofrerão mudanças.

**Tag – funciona como um marca paginas.**

### Baselines do projeto

As baselines do projeto serão criadas quando as funcionalidades do plano release () forem atendidas. A baseline deste projeto será baseada em itens do Backlog, o conjunto de requisitos do sistema.

Uma *baseline* serve como uma fotografia capaz de descrever um projeto em um determinado instante de sua execução. Esta fotografia, para funcionar como um marco ou um snapshot, deve conter algumas informações. Como exemplo, pode-se citar: arquivo de Código, controle de tarefas, planilha de teste, relatório de acompanhamento de Projeto e cronograma.

**Release – cada release é uma versão do projeto.**

**Baseline – é um marco (um check) para caso queira voltar em um ponto especifico do projeto.**

**Commit – salvo o projeto.**

### Politica de Branch

Serão criadas novas Branchs para o projeto quando for necessário realizar correção de erros do sistema, ou alterar funcionalidades de releases anteriores.

**Branch – são as tarefas a serem executadas no dia. Ex. Hoje faremos a tela de cadastro, amanhã faremos a interface gráfica.**

**Branch – eu crio um novo ramo com uma copia do que já tenho, este ramo não influencia em nada na branch principal, só quando eu faço o merge porque dai ele une.**

**Vou criar uma nova branch, por exemplo, no terceiro commit um botão não funciona, então, eu crio uma nova branch (um novo ramo) para corrigir este erro no botão, após corrigir eu realizo o merge para unir as tarefas (aplico as correções na branch principal, para que todos visualizem a correção).**

### Politica de Merge

Os responsáveis pela realização do merge serão os desenvolvedores responsáveis por trabalhar nas branchs em conjunto com os desenvolvedores que trabalharam em outra parte do código.

**Merge: Vou unir duas branchs (unir duas tarefas). Ex. Quando surge um erro no sistema, preciso criar uma nova branch para esta correção, a após a correção eu realizo o merge para unir a branch principal e a correção realizada. Eu coloco a correção na branch máster, assim todos podem visualizar esta correção.**

### Politica de Commit

Para que seja realizado um commit em um baseline será necessário constar o id do commit, o código referente às tarefas relacionadas a baseline e a versão da aplicação, conforme abaixo:

<Id commit> <código> <versão> comentário.

**Identifico com o ID do commit para ficar mais fácil à visualização (uma padronização do projeto), assim em casos de erros ou insatisfação do cliente eu sei em qual commit voltar para realizar as devidas alterações.**

**A PARTIR DAQUI VAMOS TER QUE UPAR NOSSO PROJETO NA INTERNET E TIRAR PRINTS**

**0**