ifsp – INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

SABRINA FONTES CARVALHO

1462253

Personal Software Process for Engineers: Part I

program 2

Trabalho apresentado ao Curso de Análise e Desenvolvimento de Sistemas, IFSP, como requisito parcial para a matéria de Metodologia de Análise de Sistemas Orientada a Objetos em 2016, sob orientação dos professores:

MSc Carlos Henrique Veríssimo Pereira

MSc Francisco Supino Marcondes e

MSc João Vianei Tamanini.

SÃO PAULO

2016

Sumário

LISTA DE ILUSTRAÇÕES 4

1 RELATÓRIO TÉCNICO-CIENTÍFICO 5

1.1 INTRODUÇÃO 5

1.2 DESENVOLVIMENTO 5

1.2.1 OBJETIVOS 5

1.2.1.1 GERAL 5

1.2.1.2 ESPECÍFICO 6

1.2.2 METODOLOGIA 6

1.2.2.1 PSP0.1 6

1.2.2.1.1 PLANNING 7

1.2.2.1.2 DEVELOPMENT 7

1.2.2.1.3 POSTMORTEM 9

1.2.2.2 PROCEDIMENTOS EXPERIMENTAIS 10

1.2.2.2.1 PROGRAM REQUIREMENTS 10

1.2.2.2.2 DIAGRAMAS 11

1.2.2.2.2.1 DIAGRAMA DE CASO DE USO 11

1.2.2.2.2.2 DIAGRAMA DE CLASSE 12

1.2.2.2.2.3 DIAGRAMA DE OBJETOS 13

1.2.2.2.2.4 DIAGRAMA DE COLABORAÇÃO 14

1.2.2.2.2.5 DIAGRAMA DE SEQUÊNCIA 15

1.2.2.2.2.6 DIAGRAMA DE ATIVIDADES 16

1.2.2.2.2.7 DIAGRAMA DE MÁQUINA DE ESTADO 17

1.2.2.2.2.8 DIAGRAMA DE COMPONENTES 18

1.2.2.2.2.9 DIAGRAMA DE DEPLOYMENT 19

1.2.2.2.3 PROJECT PLAN SUMMARY 20

1.2.2.2.4 PROCESS IMPROVEMENT PROPOSAL (PIP) 21

1.2.2.2.5 TIME RECORDING LOG 22

1.2.2.2.6 DEFECT RECORDING LOG 22

1.2.2.3 RESULTADOS 23

1.2.2.3.1 PLANNING 23

1.2.2.3.2 DEVELOPMENT 23

1.2.2.3.3 POSTMORTEM 23

1.3 CONCLUSÕES 24

1.4 ELEMENTOS PÓS-TEXTUAIS 24

1.4.1 REFERÊNCIAS 24

1.4.2 APÊNDICES 25

1.4.2.1 JAVA CODING STANDARD 25

1.4.2.2 JAVA SIZE COUNTING STANDARD 28

1.4.2.3 CÓDIGO 29

1.4.2.4 EVIDÊNCIAS DE TESTE 32

LISTA DE ILUSTRAÇÕES

FIGURA 1 - modelo de visão 4+1 da arquitetura 5

TABELA 1 - psp0.1 process script 6

TABELA 2 - psp0.1 planning script 7

TABELA 3 - psp0.1 development script 8

TABELA 4 - psp0.1 postmortem script 10

FIGURA 2 - Diagrama de caso de uso 12

FIGURA 3 - Diagrama de CLASSE 13

FIGURA 4 - Diagrama de OBJETOS 14

FIGURA 5 - Diagrama de colaboração 15

FIGURA 6 - Diagrama de SEQUÊNCIA 16

FIGURA 7 - Diagrama de atividades 17

FIGURA 8 - Diagrama de máquina de estado 18

FIGURA 9 - Diagrama de componentes 19

FIGURA 10 - Diagrama de deployment 19

Quadro 1 - PSP0.1 Project Plan Summary 20

Quadro 2 - process improvement proposal 21

Quadro 3 - time recording log 22

Quadro 4 - defect recoding log 22

Quadro 5 - coding satandard 25

Quadro 6 - size couting standard 28

FIGURA 11 - CÓDIGO DA CLASSE PRINCIPAL – PARTE 1 29

FIGURA 12 - CÓDIGO DA CLASSE PRINCIPAL – PARTE 2 29

FIGURA 13 - CÓDIGO DA CLASSE FILEMANAGER 30

FIGURA 14 - CÓDIGO DA CLASSE CONTADOR – PARTE 1 31

FIGURA 15 - CÓDIGO DA CLASSE CONTADOR – PARTE 2 31

FIGURA 16 - contagem de elementos do prog1 32

FIGURA 17 - CONTAGEM DE ELEMENTOS DO PROG2 32

# RELATÓRIO TÉCNICO-CIENTÍFICO

## INTRODUÇÃO

Este relatório tem o objetivo de apresentar resultados do ciclo de desenvolvimento de uma aplicação que realiza a contagem total e dos elementos de um programa utilizando a metodologia *Personal Software Process* (PSP), que é um processo de desenvolvimento de software que visa melhorar as estimativas de tempo e esforço, através de planejamento detalhado e acompanhamento de dados obtidos em programas anteriores.

Além dessa técnica, a arquitetura 4+1 está sendo utilizada para modelagem, conforme pode ser visto na Figura 1 e será desenvolvida na fase de *Design*.

1. modelo de visão 4+1 da arquitetura

Designn

Componente

Processo

Deploy

CDU

## DESENVOLVIMENTO

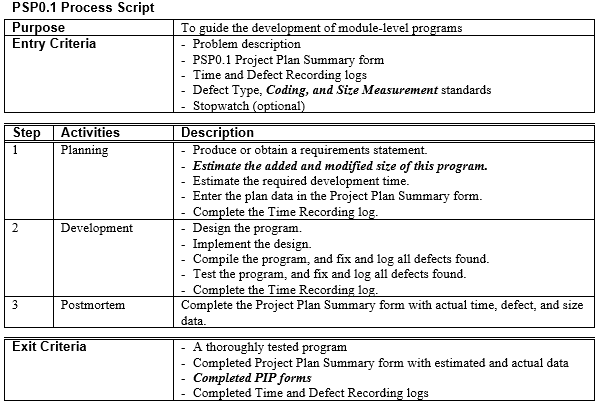
No decorrer deste relatório será desenvolvido o *Program 2* utilizando o PSP0.1 *Process Script*, conforme a Tabela 1.

### OBJETIVOS

#### GERAL

O objetivo geral desse relatório é relatar através de demonstrações, os resultados obtidos durante todas as fases de planejamento, desenvolvimento e post-mortem do PSP0.1, conforme pode ser visto na Tabela 1, a fim de demonstrar como essa metodologia impacta de maneira positiva no desenvolvimento de projetos.

1. psp0.1 process script



#### ESPECÍFICO

O objetivo específico desse relatório é o desenvolvimento do Programa 2 que, por si só, possui o objetivo de medir e estimar o tamanho dos programas produzidos, a fim de tornar possível o planejamento e gestão do trabalho de maneira eficaz.

### METODOLOGIA

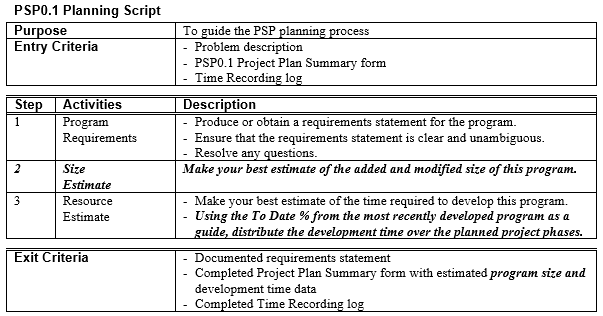
#### PSP0.1

Conforme pode ser observado na Tabela 1, no PSP0.1, além dos processos contidos no PSP0, dois processos novos são implementados, o *PIP* *Form,* que deverá conter informações para aprimorar o processo, e o *Size Counting and Coding Standards,* que são padrões da própria linguagem e a contagem do tamanho do programa que servem para aumentar a qualidade do código final. O *Project Plan Summary* ganhou uma nova seção chamada *Program Size Summary* e o tempo da fase de planejamento agora será calculado baseado no histórico do programa anterior.

##### PLANNING

Na fase de *Planning*, são necessários para se dar início ao planejamento a descrição do problema a ser resolvido, o *Project Plan Summary form* e o *Time Recording log.* Essa fase possui três subfases, conforme pode ser visto na Tabela 2: Requisitos do Programa, Estimativa do Tamanho do Programa e Estimativas dos Recursos. Espera-se que ao final da fase de planejamento os requisitos do programa estejam documentados, o *Project Plan Summary form* esteja preenchido com a estimativa do tamanho do programa e tempo de desenvolvimento e o *Time Recording log* também esteja preenchido.

1. psp0.1 planning script

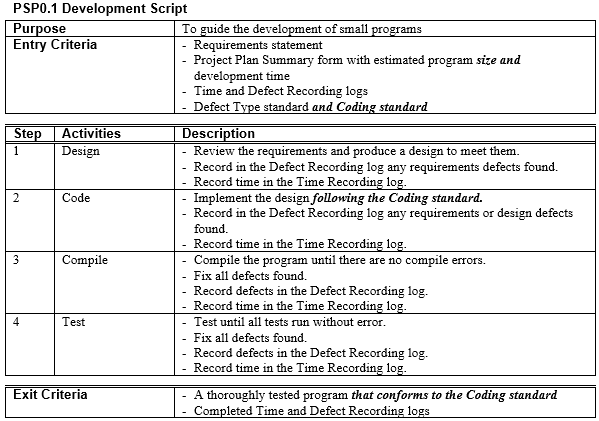


##### DEVELOPMENT

Na fase de *Development* é realizado o design do programa, a implementação do design, a compilação do programa, o teste do programa, correção de eventuais erros encontrados e o preenchimento do *Time Recording log*. Essa fase é subdividada em quatro outras subfases: *Design*, *Code*, *Compile* e *Test*, conforme pode ser visto na Tabela 3.

Para início dessa fase, devem estar prontos a documentação dos requisitos, o *Project Plan Summary form* com as estimativas preenchidas, o *Time Recording log* e o *Defect Recording log* para preenchimento e o *Defect Type Standard* e *Coding Standard* também para preenchimento. Espera-se que ao final dessa fase se tenha o programa totalmente testado e que esteja conforme o *Coding Standard* e que os *Time Recording log* e *Defect Recording log* estejam preenchidos.

1. psp0.1 development script



Na fase *Design* é feita a revisão dos requisitos produzidos na fase *Planning* e, baseados nos requisitos, são feitos os diagramas que auxiliam na construção e implementação do programa. Os diagramas assim como suas relações e explicações, podem ser encontrados no item 1.2.2.2.2 deste relatório.

Na fase *Code* é realizada a implementação do *design* feito na fase anterior. O código do programa 2 pode ser encontrado no Apêndice 1.4.2.3 desse documento.

Na fase *Compile*, o programa que foi implementado na fase *Code* é compilado e todos os defeitos que forem encontrados, são corrigidos.

Na fase *Test*, todos os testes são executados sem nenhum erro, caso algum erro seja encontrado, deve ser corrigido. Nessa fase são geradas as evidências de teste que estão no Apêndice 1.4.2.4 desse documento.

Ao término década uma das subfases, é preenchido o *Defect Recording log*, encontrado no item 1.2.2.2.4, afim de registrar eventuais erros encontrados, e a linha correspondente a essa subfase no *Time Recording log*, que pode ser encontrado no item 1.2.2.2.3.

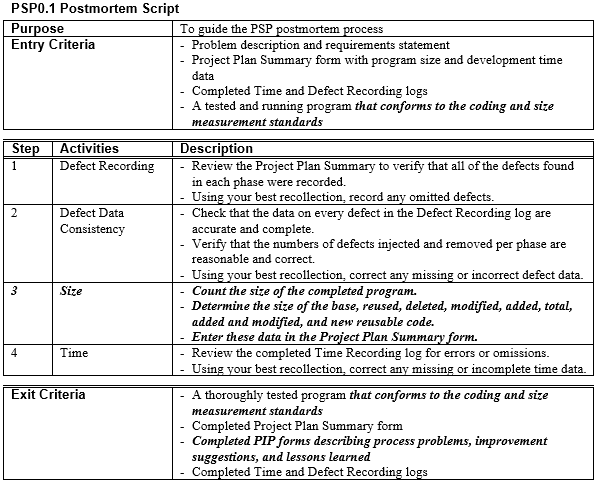
##### POSTMORTEM

A fase *Postmortem* é a última fase do PSP0.1. Nela o *Project Plan Summary,* encontrado no item 1.2.2.2.2, é completado com o tempo real gasto em cada atividade, assim como os defeitos e dados e é feita a contagem do pograma completo, determinando o tamanho do código base, reusado, deletado, modificado, adicionado e total.

Além disso, os outros formulários são revisados em busca de algum erro em cada fase ali registrada. O *Defect Recording log,* que está no item 1.2.2.2.4, também passa por essa revisão, verificando se os defeitos encontrados foram removidos e se cada um está registrado corretamente.

Por fim, *o Time Recording log,* que está no item 1.2.2.2.3, é revisto para assegurar sua veracidade.

1. psp0.1 postmortem script



#### PROCEDIMENTOS EXPERIMENTAIS

##### PROGRAM REQUIREMENTS

Para o programa 2, foram identificados os seguintes requisitos:

* O programa deverá receber um arquivo \*.java;
* O programa deverá apresentar mensagem de erro, caso o arquivo não esteja no formato correto;
* O programa deverá realizar a contagem de classes, métodos e linhas de código contidos no arquivo, de acordo com o artefato Countig Standard;
* O programa deverá apresentar na tela o resultado da contagem.

A partir dos requisitos levantados, foi realizado o diagrama de Caso de Uso, Figura 1, que demonstra como o sistema devrá realizar a contagem, dizendo funcionalmente o que deverá ocorrer a partir do momento que o programa for iniciado.

##### DIAGRAMAS

Os diagramas feitos na fase *Development*, subfase de *Design* são Diagrama de Caso de Uso (DCU), Diagrama de Objetos (DOB), Diagrama de Sequência (DSQ), Diagrama de Comportamento de Objeto (DCO), Diagrama de Classes (DCL), Diagrama de Máquina de Estado (DME) e Diagrama de Atividades (DAT).

Ainda, existem o Diagrama de Componentes e o Diagrama de Deploy, também conhecido como topologia de servidores.

###### DIAGRAMA DE CASO DE USO

Após o levantamento de requisitos, foi pensado na forma em que o programa deverá funcionar. Para auxiliar na organização das ideias de forma a transformar os requisitos em um documento funcional, foi feito o diagrama de caso de uso, conforme Figura 1.

Nele são apresentados os fluxos principal, alternativo e de exceção, com previsão de todas as funcionalidades que o programa 2 deve possuir desde o momento que se inicia, até o momento em que se encerra, incluindo eventuais erros que podem ocorrer.

1. Diagrama de caso de uso

Exibindo Diagrama de Caso de Uso.png

###### DIAGRAMA DE CLASSE

No diagrama de classe encontrado na Figura 2, três classes estão representadas, após a análise e levantamento de requisitos.

Através dos requisitos e o diagrama de caso de uso, foi necessário realizar uma pesquisa a respeito de classes e métodos de Java que realizam a leitura de arquivos. O escolhido foi o *FileManager*, porque ele possui um método específico que permite obter todos os arquivos de um diretório passado como parâmetro, enquanto o *FileReader* trata de arquivos individualmente, mas também foi utilizado para se obter o caminho do diretório dos arquivos.

Além disso, foi identificada a necessidade de se ter uma classe que realize a contagem das linhas de código, assim como os métodos, além da classe principal.

1. Diagrama de CLASSE

C:\Users\IBM_ADMIN\Downloads\Diagrama de Classes.png

###### DIAGRAMA DE OBJETOS

O diagrama de objetos foi feito com base no diagrama de classes, ele retrata os objetos das classes instanciados e os valores de suas variáveis. No diagrama retratado na Figura 3, retrata o início do programa, onde o diretório já foi inserido, a classe Contador já foi instanciada, porém a contagem ainda não foi iniciada, portanto os valores das variáveis estão zerados.

1. Diagrama de OBJETOS

C:\Users\IBM_ADMIN\Downloads\Diagrama de Objetos.png

###### DIAGRAMA DE COLABORAÇÃO

A partir do diagrama de objetos, percebeu-se a necessidade de demonstrar as interações entre os objetos e seus métodos, para isso, foi feita essa representação através do diagrama de colaboração, conforme Figura 4.

Nele pode ser visto a ordem com que os métodos são chamados assim como seus parâmetros passados. No caso do Programa 2, todos os métodos são chamados pela classe Principal que acaba funcionando como uma controladora de todas as operações do programa 2 e também como uma apresentadora, uma vez que é responsabilidade dessa classe demonstrar o resultado da contagem na tela.

1. Diagrama de colaboração

C:\Users\IBM_ADMIN\Downloads\Diagrama de Colaboração.png

###### DIAGRAMA DE SEQUÊNCIA

Após a construção do diagrama de colaboração, indetificou-se outra necessidade, a de demonstrar através dos fluxos do diagrama de sequência, a ordem com que cada método é chamado, assim cmom suas dependências.

Como pode ser visto na Figura 5, os objetos são interdependentes, o início se dá no momento em que o usuário insere o diret’roio onde estão os arquivos a serem contados, a classe Principal solicita que a classe FileManager verifique se esse diretório é válido ou se possui algum arquivo dentro dele, após isso, os caminhos dos arquivos são solicitados para essa mesma classe que retorna o resultado em forma de *array.*

1. Diagrama de SEQUÊNCIA

C:\Users\IBM_ADMIN\Downloads\Diagrama de Sequência.png

###### DIAGRAMA DE ATIVIDADES

O diagrama de atividades teve por objetivo demonstrar a sequência em que as atividades acontecem, demonstrando a interação do usuário com o sistema.

Conforme a Figura 6, podemos ver claramente o papel do usuário e o papel do sistema no programa 2.

1. Diagrama de atividades

C:\Users\IBM_ADMIN\Downloads\Diagrama de Atividades (1).png

###### DIAGRAMA DE MÁQUINA DE ESTADO

O diagrama de máquina de estado, conforme Figura 7, demonstra os momentos em que o programa 2 está ativo, ou seja, o momento em que está ocorrendo algum processamento. No caso do programa 2, foram identificados 3 momentos em que isso ocorre, o momento em que o programa está buscando o diretório, buscando os caminhos dos arquivos e fazendo a contagem de métodos e linhas de código.

1. Diagrama de máquina de estado

C:\Users\IBM_ADMIN\Downloads\Diagrama de Máquina de Estado.png

###### DIAGRAMA DE COMPONENTES

No diagrama de componentes, representado na Figura 8, são demonstrados todos os componentes que são essenciais para a execução e funcionamento do programa. Pode ser observado o próprio programa 2, em sua forma executável, a interface com o usuário e as bibliotecas do Java que são essenciais para a construção do programa.

1. Diagrama de componentes

C:\Users\IBM_ADMIN\Downloads\Diagrama de Componentes.png

###### DIAGRAMA DE DEPLOYMENT

O diagrama de *deployment* é também conhecido como diagrama de implantação (ou instalação) ele representa as partes físicas envolvidas na execução do programa. No programa 2, por se tratar de um programa pequeno, o único *hardware* envolvido é o terminal do usuário, onde a aplicação será executada, conforme pode ser visto na Figura 9.

1. Diagrama de deployment

C:\Users\IBM_ADMIN\Downloads\Diagrama de Deployment.png

##### PROJECT PLAN SUMMARY

1. PSP0.1 Project Plan Summary

**PSP0.1 Project Plan Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 15/10/2016 |
| Program | Contagem de linhas e elementos de um programa | Program # | 2 |
| Instructor | Carlos / Francisco / João | Language | Java |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Program Size*** | ***Plan*** | | |  | ***Actual*** | | |  | ***To Date*** | | |
| ***Base (B)*** |  | | |  | 45 | | |  |  | | |
|  |  | | |  | ***(Measured)*** | | |  |  | | |
| ***Deleted (D)*** |  | | |  | 0 | | |  |  | | |
|  |  | | |  | ***(Counted)*** | | |  |  | | |
| ***Modified (M)*** |  | | |  | 0 | | |  |  | | |
|  |  | | |  | ***(Counted)*** | | |  |  | | |
| ***Added (A)*** |  | | |  | 87 | | |  |  | | |
|  |  | | |  | ***(T − B + D − R)*** | | |  |  | | |
| ***Reused (R)*** |  | | |  | 56 | | |  | 56 | | |
|  |  | | |  | ***(Counted)*** | | |  |  | | |
| ***Added and Modified (A+M)*** | 300 | | |  | 87 | | |  | 132 | | |
|  |  | | |  | ***(A + M)*** | | |  |  | | |
| ***Total Size (T)*** |  | | |  | 87 | | |  | 132 | | |
|  |  | | |  | ***(Measured)*** | | |  |  | | |
| ***Total New Reusable*** |  | | |  | 87 | | |  | 132 | | |
|  |  | | |  |  | | |  |  | | |
| **Time in Phase (min.)** | ***Plan*** |  | **Actual** | | |  | **To Date** | | |  | **To Date %** |
| Planning | 60 |  | 43 | | |  | 66 | | |  | 11,61 |
| Design | 180 |  | 165 | | |  | 300 | | |  | 52,81 |
| Code | 100 |  | 78 | | |  | 120 | | |  | 21,12 |
| Compile | 30 |  | 16 | | |  | 19 | | |  | 3,34 |
| Test | 30 |  | 6 | | |  | 13 | | |  | 2,28 |
| Postmortem | 60 |  | 42 | | |  | 50 | | |  | 8,80 |
| Total | 460 |  | 350 | | |  | 568 | | |  | 100 |
|  |  |  |  | | |  |  | | |  |  |
| **Defects Injected** |  |  | **Actual** | | |  | **To Date** | | |  | **To Date %** |
| Planning |  |  | 0 | | |  | 0 | | |  | 0 |
| Design |  |  | 2 | | |  | 2 | | |  | 40 |
| Code |  |  | 1 | | |  | 2 | | |  | 40 |
| Compile |  |  | 1 | | |  | 1 | | |  | 10 |
| Test |  |  | 0 | | |  | 0 | | |  | 0 |
| Total Development |  |  | 4 | | |  | 5 | | |  | 100 |
|  |  |  |  | | |  |  | | |  |  |
| **Defects Removed** |  |  | **Actual** | | |  | **To Date** | | |  | **To Date %** |
| Planning |  |  | 0 | | |  | 0 | | |  | 0 |
| Design |  |  | 2 | | |  | 2 | | |  | 40 |
| Code |  |  | 1 | | |  | 2 | | |  | 40 |
| Compile |  |  | 1 | | |  | 1 | | |  | 10 |
| Test |  |  | 0 | | |  | 0 | | |  | 0 |
| Total Development |  |  | 4 | | |  | 4 | | |  | 100 |
| After Development |  |  | 0 | | |  | 0 | | |  |  |

##### PROCESS IMPROVEMENT PROPOSAL (PIP)

1. process improvement proposal

**PSP Process Improvement Proposal (PIP)**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date |  |
| Program | Contagem de linhas e elementos de um programa | Program # | 2 |
| Instructor | Carlos / Francisco / João | Language | Java |

|  |
| --- |
| **Problem Description** |
| Briefly describe the problems that you encountered. |
| Encontrei problemas na implementação da classe de leitura do arquivo. A classe que inicialmente havia |
| decidido utilizar se mostrar mais complexa do que uma classe pronta que atendia o necessário para |
| atender a necessidade do PROG2. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| **Proposal Description** |
| Briefly describe the process improvements that you propose. |
| Durante o processo, pude perceber que quanto mais tempo se gasta durante o planejamento, menos tempo |
| se perde com correções de bugs e nas fases finais do ciclo de desenvolvimento. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| **Other Notes and Comments** |
| Note any other comments or observations that describe your experiences or improvement ideas. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

##### TIME RECORDING LOG

1. time recording log

**PSP Time Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 15/10/2016 |
| Program | Contagem de linhas e elementos de um programa | Program # | 2 |
| Instructor | Carlos / Francisco / João | Language | Java |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project** | **Phase** | **Start Date and Time** | **Int. Time** | **Stop Date and Time** | **Delta**  **Time** | **Comments** |
| 2 | Planning | 15/10 – 13h16 | - | 15/10 – 13h59 | 43 min |  |
| 2 | Design | 16/10 – 10h27 | - | 17/10 – 22h56 | 165 min | O desenvolvimento dessa fase foi feito em dias distintos, por causa de outras atividades realizadas. |
| 2 | Code | 18/10 – 12h37 | - | 18/10 – 23h11 | 78 min | Houveram pausas para se realizar outras tarefas. |
| 2 | Compile | 18/10 – 23h12 | - | 18/10 – 23h28 | 16 min |  |
| 2 | Test | 18/10 – 23h28 | - | 19/10 – 23h34 | 6 min |  |
| 2 | Postmortem | 19/10 – 22h54 |  | 19/10 – 23h36 | 42 min |  |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| Defect Types |  |
| 10 Documentation | 60 Checking |
| 20 Syntax | 70 Data |
| 30 Build, Package | 80 Function |
| 40 Assignment | 90 System |
| 50 Interface | 100 Environment |

##### DEFECT RECORDING LOG

1. defect recoding log

**PSP Defect Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 19/10/2016 |
| Program | Contagem de linhas e elementos de um programa | Program # | 2 |
| Instructor | Carlos / Francisco / João | Language | Java |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
| 2 |  | | 16/10 |  | 1 |  | 10 |  | 1 |  | 1 |  | 3 |  |  |
| Description: | | | Encontrado erro no diagram de classe, durante a realização do diagrama de objetos. Foi | | | | | | | | | | | | | |
| verificado que mais três variáveis eram necessárias. | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
| 2 |  | | 16/10 |  | 2 |  | 10 |  | 1 |  | 1 |  | 2 |  |  |
| Description: | | | Foi identificado um erro de duplicidade de fluxos no diagram de sequência, por simples | | | | | | | | | | | | | |
| falta de atenção e foi corrigido no mesmo momento. | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
| 2 |  | | 18/10 |  | 3 |  | 20 |  | 1 |  | 1 |  | 1 |  |  |
| Description: | | | Erro na compilação. Faltou um ponto e vírgula. | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
| 2 |  | | 18/10 |  | 4 |  | 80 |  | 1 |  | 1 |  | 8 |  |  |
| Description: | | | Erro na codificação. O caminho do arquivo não estava sendo gravado corretamente na | | | | | | | | | | | | | |
| variável, por isso a leitura não estava sendo feita. | | | | | | | | | | | | | | | | |

#### RESULTADOS

##### PLANNING

A fase de *planning,* a execução aconteceu abaixo do tempo estimado.

Esse resultado aconteceu por já ter uma pequena experiência com o programa anterior, mesmo assim as estimativas foram feitas acrescentando um tempo a mais do que o utilizado no programa anterior, por causa do aumento da dificuldade no programa 2 e a necessidade de se fazer mais pesquisas, por se tratar de um assunto que não era conhecido.

##### DEVELOPMENT

Na fase de *design,* foi usado quase todo o tempo estimado, porém a estimativa também não aconteceu de forma acertiva, uma vez que foi gasto menor tempo para a análise e os diagramas. Esse resultado ocorreu, porque já se tinha os requisitos e o caso de uso desenhados da fase anterior (*planning*), de qualquer forma, essa fase foi a mais extensa de todas as fases, mas crucial para as fases restantes.

Na fase de *code,* o tempo estimado não foi totalmente utilizado, apesar dos erros encontrados e corrigidos. Isso se deve a pesquisa e ao planejamento feito durante a fase de *design*.

Na fase de *compile,* um erro foi encontrado e logo corrigido. O erro ocorreu por causa de falta de atenção e foi corrigido no mesmo momento. Não foi gasto todo o tempo estimado, por causa da fase anterior, que foi bem construída.

Na fase de *test,* nenhum erro foi encontrado, uma vez que erros foram identificados nas fases posteriores, *compile* e *test*, fazendo com que os testes tenham sido bem rápidos de fazer e assertivos, conforme pode ser visto no Apêndice 1.4.2.4.

##### POSTMORTEM

Na fase de *Postmortem,* foi feita a revisão de todos os documentos que pertencem ao PSP 0.1 e preenchimento das estatísticas totais do projeto.

Apesar do programa 2 ser de maior complexidade que o programa 1, inclusive em números de linhas, pode ser verificado no *Project Plan Summary*, que houve um tempo reduzido de desenvolvimento e também houve a reutilização de código.

## CONCLUSÕES

Através dos resultados que foram apresentados ao longo desse relatório, foi provado que uma aplicação bem planejada resulta em maior qualidade e assertividade na codificação e menor tempo de desenvolvimento.

Foi possível concluir também que houve um aumento de maturidade, por mais que pequeno, mas significativo, na utilização do metodologia do PSP.

## ELEMENTOS PÓS-TEXTUAIS

### REFERÊNCIAS

DRAW IO DIAGRAMS. **Draw IO Diagrams**. Disponível em: < https://www.draw.io/>.

LARMAN, CRAIG. **Utilizando UML e Padrões, Uma introdução à análise e ao projeto orientados a objetos**. Porto Alegre: Bookman, 2000.

ORACLE, JAVA™ PLATFORM STANDARD ED. 8. **Class FileReader**. Disponível em: <http://docs.oracle.com/javase/8/docs/api/?java/io/FileReader.html>.

ORACLE, JAVA™ PLATFORM STANDARD ED. 8. **Class JFileChooser**. Disponível em: < https://docs.oracle.com/javase/7/docs/api/javax/swing/JFileChooser.html>.

SUN MICROSYSTEMS, INC. **Java Code Conventions**, Setembro 1997. Disponível em: < http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>.

WEBGOAL. **Cronômetro Online**. Disponível em: <http://cronometronline.com.br/>.

### APÊNDICES

#### JAVA CODING STANDARD

1. coding satandard

**Coding Standard Template**

|  |  |
| --- | --- |
| Purpose | To guide the development of JAVA programs |
| Program Headers | Begin all programs with a descriptive header. |
| Header Format | /\*\*  \* Contagem de linhas e elementos de um programa  \*  \* O Programa 2 é a implementação de um programa que  \* conta linhas de código, métodos e classes de programas.  \*  \* @autor Sabrina Fontes Carvalho  \* @versão 1.0  \* @data 19/10/2016  \*/ |
| Listing Contents | Provide a summary of the listing contents. |
| Contents  Example | /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Listing Contents:  /\* Reuse instructions  /\* FileManager.java  /\* Compilation instructions  /\* Includes  /\* java.util.regex  /\* java.io.File  /\* Class declarations:  /\* public class Principal  /\* public class FileManager  /\* public class Contador  /\* Source code in: workspace/Programa2  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |
| Reuse Instructions | * Describe how the program is used. Provide the declaration format, parameter values and types, and parameter limits. * Provide warnings of illegal values, overflow conditions, or other conditions that could potentially result in improper operation. |
| Reuse Example | /\*  \* Método utilizado para selecionar um diretório via JFileChooser  \* @return File Retorna um arquivo File com o diretório selecionado  \*/  public static File selecionaDiretorio() {  .  .  .  }  /\*  \* Método utilizado para pegar o caminho de todos os arquivos  \* dentro de um diretório.  \*  \* @param caminho  \* @return ArrayList com os caminhos  \*/  public static ArrayList<String> obterCaminho(String path) {  .  .  .  } |
| Identifiers | Use descriptive names for all variables, function names, constants, and other identifiers. Avoid abbreviations or single letter variables. |
| Identifier Example | ArrayList<String> arrayPaths; /\* This is GOOD \*/  ArrayList<String> a; /\* This is BAD \*/ |

(continued)**Coding Standard Template (continued)**

|  |  |
| --- | --- |
| Comments | * Document the code so that the reader can understand its operation. * Comments should explain both the purpose and behavior of the code. * Comment variable declarations to indicate their purpose. |
| Good Comment | if (sCurrentLine.endsWith("{")) //Essa linha termina com “{“? |
| Bad Comment | if (sCurrentLine.endsWith("{")) //Verifica se a linha termina com “{“ |
| Major Sections | Precede major program sections by a block comment that describes the processing that is done in the next section |
| Example |  |
| Blank Spaces | * Write programs with sufficient spacing so they do not appear crowded. * Separate every program construct with at least one space. |
| Indenting | * Indent every level of brace from the previous one. * Open and closing braces should be on lines by themselves and aligned with each other. |
| Indenting  Example | public int contaLinhas() {  int LOCCounter = 0;  for (String path : caminhosLista) {  int classLOCCount = 0;  try (BufferedReader br = new BufferedReader(new FileReader(path))) {  String sCurrentLine;  while ((sCurrentLine = br.readLine()) != null) {  if (sCurrentLine.endsWith("{")) {  classLOCCount++;  LOCCounter++;  } else if (sCurrentLine.endsWith(";")) {  classLOCCount++;  LOCCounter++;  }  }  System.out.println("LOC da classe " + path + " = " + classLOCCount);  } catch (IOException e) {  e.printStackTrace();  }  }  return LOCCounter;  } |
| Capitalization | * Capitalized all defines. * Lowercase all other identifiers and reserved words. * Messages being output to the user can be mixed-case so as to make a clean user presentation. |
| Capitalization Example | public int contaLinhas()  int classLOCCount = 0;  public class Contador |

#### JAVA SIZE COUNTING STANDARD

1. size couting standard

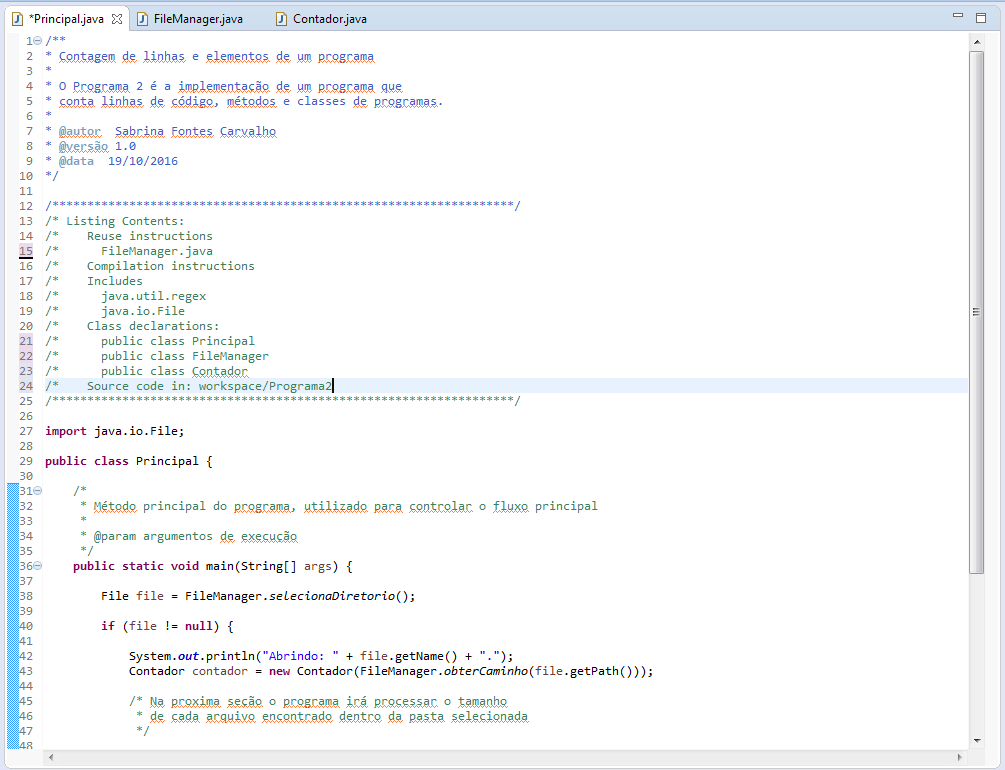
**Size Counting Standard**

|  |  |  |  |
| --- | --- | --- | --- |
| Definition Name: | Contagem elementos de um programa | Language: | Java |
| Author: | Sabrina Fontes Carvalho | Date: | 10/10/2016 |

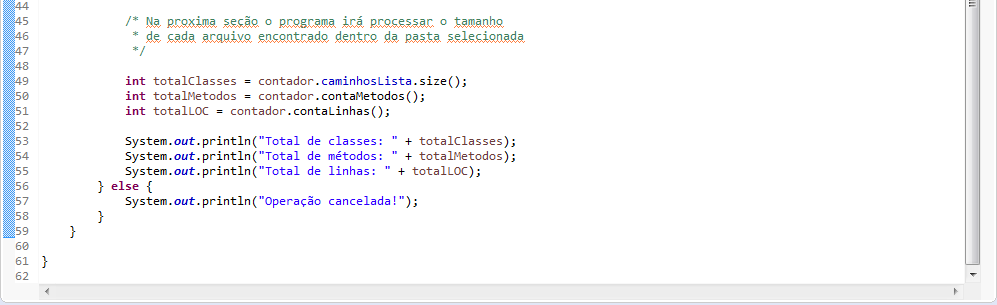
|  |  |  |
| --- | --- | --- |
| **Count Type** | **Type** | **Comments** |
| Physical/Logical | Lógico |  |
| **Statement Type** | **Included** | **Comments** |
| Executable | Sim |  |
| Nonexecutable: |  |  |
| Declarations | Não |  |
| Compiler Directives | Não |  |
| Comments | Não |  |
| Blank lines | Não |  |
| Other elements |  |  |
|  | Sim | Conta todos os métodos, incluindo o método construtor. |
|  | Sim | Conta todas as classes, incluinda a classe Principal (Main). |
|  |  |  |
| **Clarifications** |  | **Examples/Cases** |
|  |  |  |
| Begin...end | Note 1 |  |
| End symbols | Notes 1,2 |  |
|  |  |  |
| Note 1 |  | Conta todas as vezes em que há a ocorrência das palavras “public”, “private” e “protected”. |
| Note 2 |  | Conta toda vez em que o “;” ou o “}” aparecem. |

#### CÓDIGO

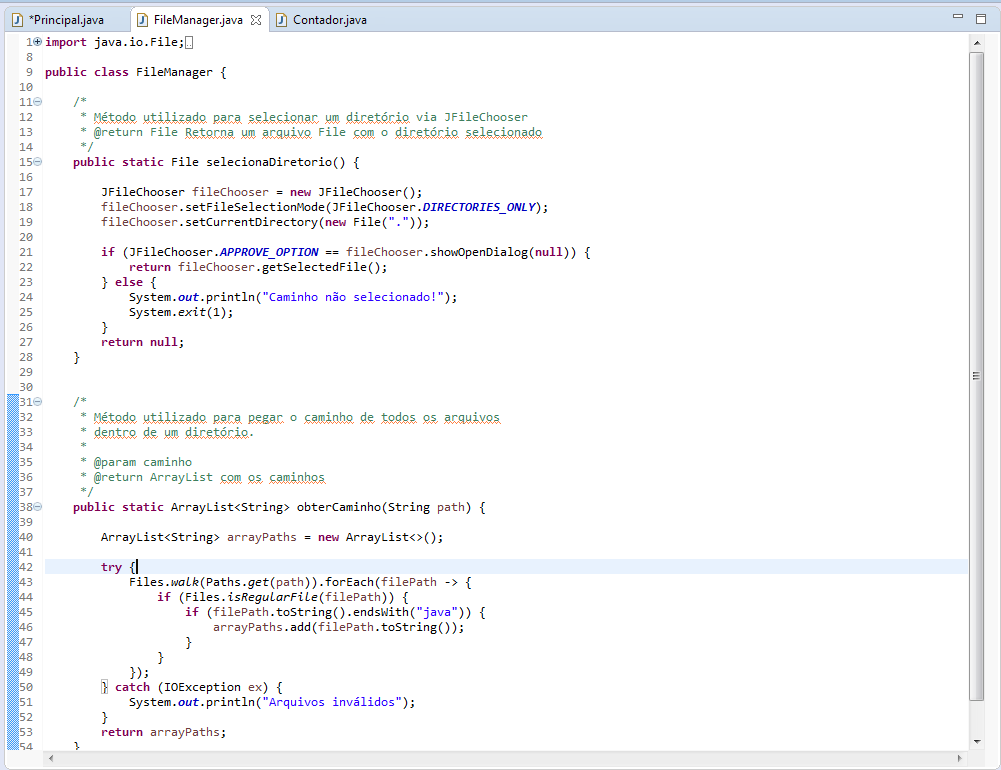
1. CÓDIGO DA CLASSE PRINCIPAL – PARTE 1



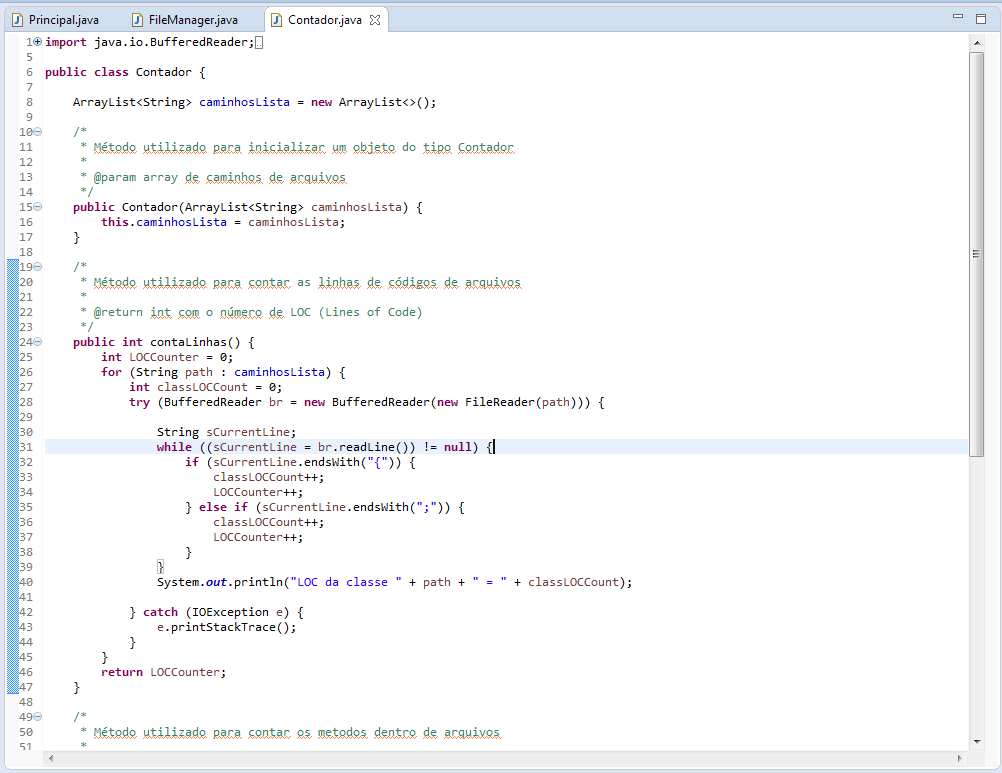
1. CÓDIGO DA CLASSE PRINCIPAL – PARTE 2



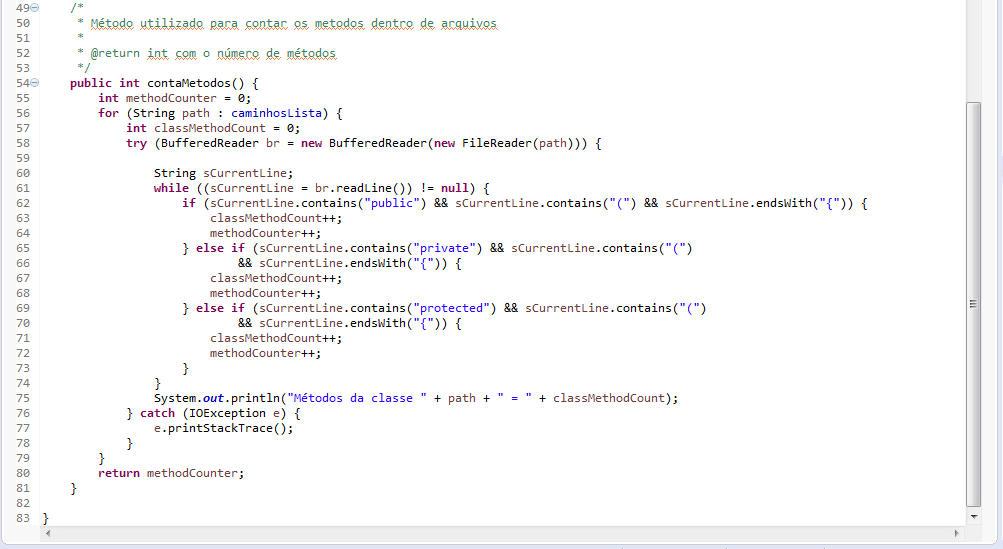
1. CÓDIGO DA CLASSE FILEMANAGER



1. CÓDIGO DA CLASSE CONTADOR – PARTE 1

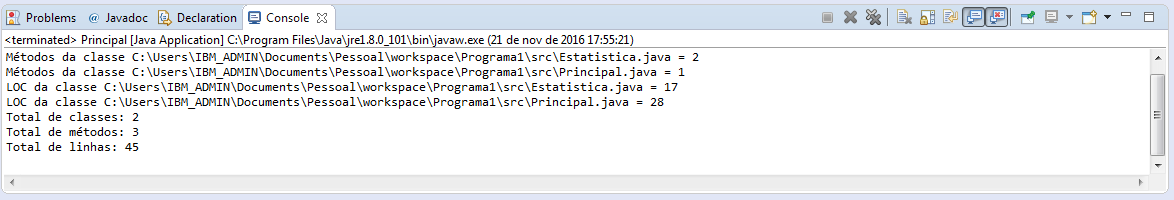


1. CÓDIGO DA CLASSE CONTADOR – PARTE 2



#### EVIDÊNCIAS DE TESTE

1. contagem de elementos do prog1



1. CONTAGEM DE ELEMENTOS DO PROG2

