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

SABRINA FONTES CARVALHO

1462253

Personal Software Process for Engineers: Part I

program 1

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 Tomanini.

SÃO PAULO

2016

Sumário

LISTA DE ILUSTRAÇÕES 3

Introdução 4

1 planning 5

1.1 DESCRIÇÃO do problema 5

1.2 requisitos 5

1.3 estimativas 6

2 development 7

2.1 design 7

2.1.1 DIAGRAMA DE CASO DE USO 8

2.1.2 DIAGRAMA DE OBJETOS 9

2.1.3 DIAGRAMA DE COMPORTAMENTO DE OBJETOS 10

2.1.4 DIAGRAMA DE SEQUÊNCIA DE MENSAGEM DINÂMICA 11

2.1.5 DIAGRAMA DE ATIVIDADES 12

2.1.6 DIAGRAMA DE MÁQUINA DE ESTADOS 13

2.1.7 DIAGRAMA DE CLASSES 14

2.1.8 DIAGRAMA DE COMPONENTES 15

2.1.9 DIAGRAMA DE DEPLOY 16

2.2 CODE 16

2.2.1 CÓDIGO 16

2.3 COMPILE 18

2.4 TEST 19

2.4.1 EVIDÊNCIAS DE TESTE 19

3 postmortem 22

ReferÊncias 23

Apêndice 1 - PSP0 PROject plan summary 24

Apêndice 2 - PSP TIME RECORDING LOG 25

Apêndice 3 - PSP defect recording log 26

LISTA DE ILUSTRAÇÕES

TABELA 1 - psp0 process script 4

TABELA 2 - psp0 planning script 5

TABELA 3 - psp0 development script 7

FIGURA 1 - modelo de visão 4+1 da arquitetura 8

FIGURA 2 - Diagrama de caso de uso 9

FIGURA 3 - Diagrama de objetos 10

FIGURA 4 - Diagrama de comportamento de objetos 11

FIGURA 5 - Diagrama de SEQUÊNCIA DE MENSAGEM DINÂMICA 12

FIGURA 6 - Diagrama de ATIVIDADES 13

FIGURA 7 - Diagrama de MÁQUINA DE ESTADOS 14

FIGURA 8 - Diagrama de CLASSES 15

FIGURA 9 - Diagrama de componentes 15

FIGURA 10 - Diagrama de deploy 16

FIGURA 11 - Código da classe principal.java 17

FIGURA 12 - código da classe estatistica.java 18

TABELA 4 - DADOS PARA TESTE 19

FIGURA 13 - evidência de teste da coluna 1 20

FIGURA 14 - evidência de teste da coluna 2 20

FIGURA 15 - finalização sem a inserção de dados 20

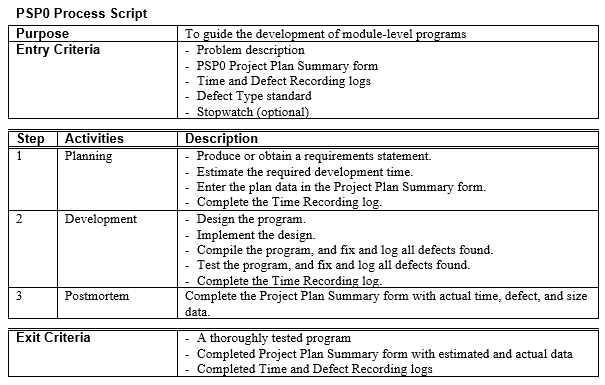
FIGURA 16 - fluxo alternativo 21

TABELA 5 - resultado esperado 21

Introdução

O trabalho está sendo desenvolvido com o objetivo de aplicar a metodologia *Personal Software* *Process* (PSP) no desenvolvimento de pequenos programas. Além dessa técnica, a arquitetura 4+1 está sendo utilizada para modelagem. Esse trabalho trata especificamente do ciclo de desenvolvimento do programa 1, utilizando inicialmente o PSP0 *Process Script*, conforme segue na Tabela 1.

1. psp0 process script

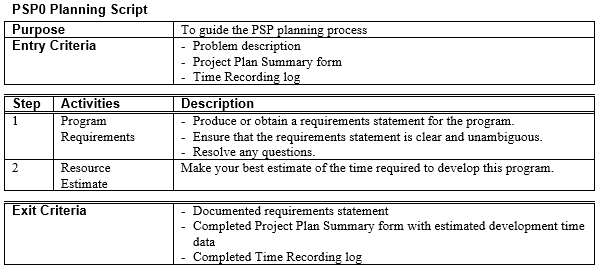


Conforme demonstrado na figura 1, o PSP0 se divide em três fases: *Planning*, *Development* e *Postmortem*, ambas as fases serão abordadas nesse trabalho.

# planning

Nessa fase é feito o levantamento dos requisitos, a descrição do problema, o preenchimento da data planejada no *Project Plan Summary* e o preenchimento do *Time Recording log* referente a esta fase (*Plan*), conforme pode ser conferido na Tabela 2.

1. psp0 planning script



## DESCRIÇÃO do problema

Usando PSP0, escreva um programa que calcule a média e o desvio padrão de um conjunto de *n* números reais.

Seu programa poderá ler os *n* números reais a partir do teclado, arquivo ou qualquer outra fonte.

Use uma lista encadeada para armazenar os *n* números para os cálculos. Se necessário, uma variável ou array estático, banco de dados ou outra estrutura de dados podem ser usados para manter os dados.

## requisitos

Para o programa 1, têm-se os seguites requisitos:

* Calcular média e desvio padrão de um conjunto de *n* números reais;
* Ler *n*  números reais do teclado, de um arquivo ou qualquer outra fonte;
* Usar lista encadeada para gravar os números para o cálculo. Caso necessário, pode-se utilizar uma variável ou array estático, banco de dados ou outra estrutura de dados para armazenar os dados;
* Testar o programa.

## estimativas

A estimativa foi feita com base em especulação, uma vez que o desenvolvimento desse programa nos moldes do PSP e utilizando dos diagramas da arquitetura 4+1, está sendo realizado pela primeira vez.

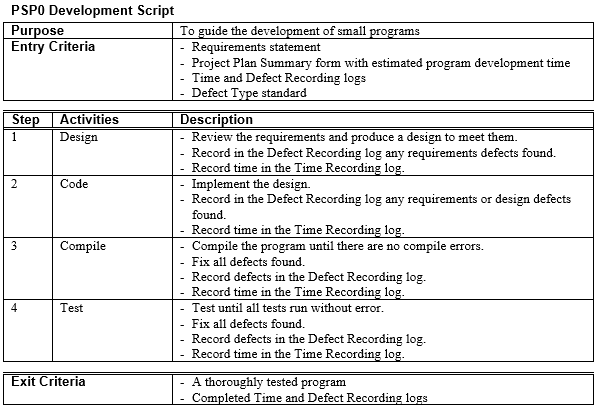
Conforme o PSP0 determina, a estimativa pode ser encontrada no PSP0 *Project Plan Summary* na coluna *Plan*, ao final desse documento, no Apêndice 1.

Além disso, deve ser preenchido o *Time Recording log* referente a fase Planning. Esse formulário pode ser encontrado no Apêndice 2, ao final desse documento.

# 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.

1. psp0 development script



## design

Nessa fase são feitos os diagramas que auxiliam na construção e implementação do programa, respeitando a arquitetura 4+1, exemplificada na figura 1, e utilizando os requisitos produzidos na fase de Planning. Nessa fase é preenchido o *Defect Recording log*, encontrado no Apêndice 3, afim de registrar eventuais erros encontrados, e e a linha correspondente a essa subfase no *Time Recording log*, que pode ser encontrado no Apêndice 2.

1. modelo de visão 4+1 da arquitetura

Designn

Componente

Processo

Deploy

CDU

Os diagramas da fase de Design são Diagrama de Caso de Uso (DCU), Diagrama de Objetos (DOB), Diagrama de Sequência de Mensagens (DSM), Diagrama de Comportamento de Objeto (DCO), Diagrama de Classes (DCL), Diagrama de Máquina de Estado (DME) e Diagrama de Atividades (DAT).

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

### DIAGRAMA DE CASO DE USO

O diagrama de caso de uso tem o objetivo de auxiliar no levantamento de requisitos. No Programa 1, foi identificado como autor o engenheiro de software, uma vez que o cálculo de desvio padrão é utilizado no próprio PSP, mais especificamente no PROBE que é utilizado a partir do PSP1 para estimar com maior certeza o ciclo de desenvolvimento.

A fronteira foi identificada como o próprio Programa de Análise Estatística, por causa da baixa complexidade do Programa 1. Abaixo do diagrama pode ser observado a descrição de como o caso de uso funciona.

Todo o diagrama de caso de uso está demonstrado na Figura 2.

1. Diagrama de caso de uso

C:\Users\IBM_ADMIN\Downloads\Diagrama de Caso de Uso.png

### DIAGRAMA DE OBJETOS

O diagrama de objetos tem o objetivo de retratar os objetos instanciados na aplicação. No Programa 1, foi identificado quatro objetos principais, são eles :Estatistica, :Principal, sc:Scanner e lista:LinkedList, conforme pode ser visto na Figura 3.

O objeto :Estatistica é uma instância da classe Estatística, onde se calcula a média e o desvio padrão.

O objeto :Principal é a instância da classe Principal que faz as chamadas das demais classes e é responsável por receber o *input* do usuário e apresentar o *output*.

O objeto sc:Scanner é a instância da classe *Scanner*, que é responsável por apresentar os dados na tela, assim como capturar dados digitados pelo usuário.

O objeto lista:LinkedList é a instância da classe LinkedList, onde são armazenados os dados inseridos pelo usuário, no diagrama são encontradas outras instâncias dessa classe, uma vez que ela pode ser utilizada várias vezes durante a execução do programa.

1. Diagrama de objetos

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

### DIAGRAMA DE COMPORTAMENTO DE OBJETOS

O diagrama de comportamento de objetos tem o objetivo de mostrar as interações entre os objetos instanciados na aplicação. Conforme a Figura 4, o objeto Principal instancia a classe Scanner que captura o *input* de dados feito pelo usuário, demosntrado pelo fluxo 1 no diagrama. O objeto Scanner devolve o valor para o objeto Principal que armazena-o na primeira instância do objeto lista:LinkedList. O objeto lista:LinkedList é utilizado pelo objeto :Estatistica para a realização dos cálculos, devolvendo ao objeto :Principal o resultado que será apresentado na tela para o usuário.

1. Diagrama de comportamento de objetos

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

### DIAGRAMA DE SEQUÊNCIA DE MENSAGEM DINÂMICA

O diagrama de sequência de mensagens dinâmicas tem o objetivo de retratar a sequência de processos entre os objetos. Conforme a Figura 5, o usuário informe os valores capturados pelo objeto Principal, esse objeto aramazena os dados na instância do *LinkedList*, devolvendo a lista completa ao objeto Principal, que realiza a chamada do objeto Estatística onde são feitos os cálculos, consultando os valores inseridos na *LinkedList*. Ao término do cálculo, o objeto Estatística devolve o resultado para o Principal, que apresenta o resultado na tela.

1. Diagrama de SEQUÊNCIA DE MENSAGEM DINÂMICA

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

### DIAGRAMA DE ATIVIDADES

O diagrama de sequência de mensagens dinâmicas tem o objetivo de retratar a sequência de processos entre os objetos. Conforme a Figura 5, o usuário informe os valores capturados pelo objeto Principal, esse objeto aramazena os dados na instância do *LinkedList*, devolvendo a lista completa ao objeto Principal, que realiza a chamada do objeto Estatística onde são feitos os cálculos, consultando os valores inseridos na *LinkedList*. Ao término do cálculo, o objeto Estatística devolve o resultado para o Principal, que apresenta o resultado na tela.

1. Diagrama de ATIVIDADES

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

### DIAGRAMA DE MÁQUINA DE ESTADOS

O diagrama de máquina de estados tem o objetivo de demonstrar o funcionamento do programa através das transições de estado. Conforme a Figura 7, o programa começa no estado inicializado, após o início é solicitado ao usuário a inserção de dados, o programa adiciona os dados na LinkedList e, se os dados não foram inseridos, o estado muda para finalizado, se os dados forem inseridos, o estado muda para validando os dados. Se os dados não forem válidos, o programa muda para estado finalizado, se os dados forem válidos, o estado muda para calculando a média e desvio padrão e o resultado é apresentado na tela. Após a exibição, o programa muda para estado finalizado.

1. Diagrama de MÁQUINA DE ESTADOS

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

### DIAGRAMA DE CLASSES

O diagrama de classe tem o objetivo de retratar as classes existentes no programa. Conforme a Figura 8, foram identificadas quatro classes que interagem entre si, sendo elas a Principal, que é responsável pela chamada das demais classes, a Estatística, que é responsável pelos cálculos, a Scanner, que é responsável por capturar os dados inseridos pelo usuário e apresentar o resultado na tela, e a LinkedLista, que é responsável por armazenar os valores inseridos pelo usuário.

1. Diagrama de CLASSES

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

### DIAGRAMA DE COMPONENTES

O diagrama de classe tem o objetivo de representar a estrutura do programa, descrevendo os componentes, interfaces e dependências. Conforme a Figura 9, foi identificado no programa 1 um código fonte que é executado na versão 1.8 do JRE.

1. Diagrama de componentes

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

### DIAGRAMA DE DEPLOY

O diagrama de deploy, também conhecido como diagrama de instalação, representa a arquitetura física necessária ao programa. Conforme a Figura 10, foi identificado que o hardware necessário para a execução do programa é a própria máquina do usuário.

1. Diagrama de deploy

C:\Users\IBM_ADMIN\Downloads\Diagrama de Topologia de Servidores.png

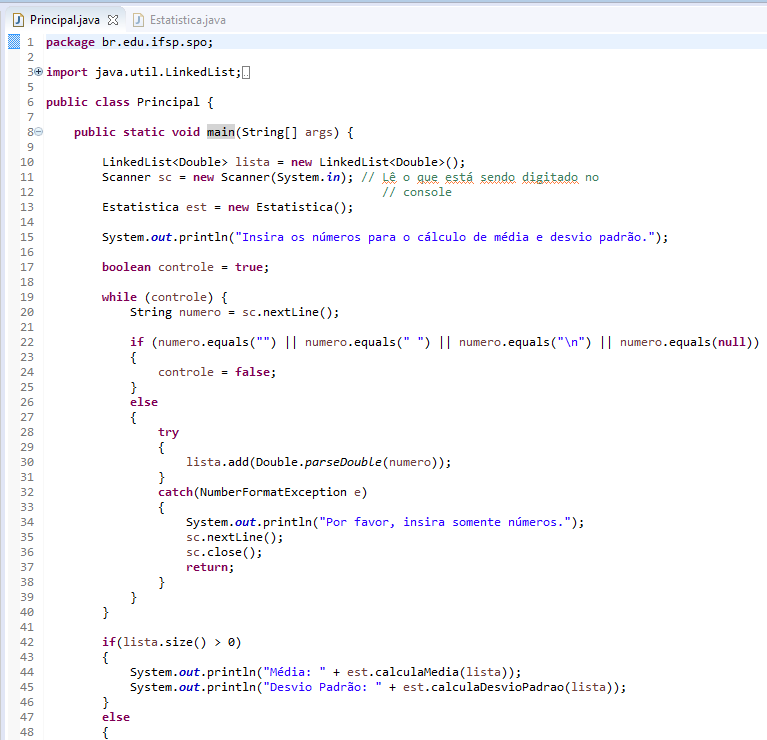
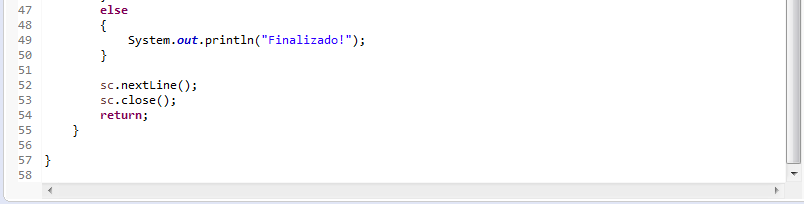
## CODE

Nessa fase é realizada a implementação do *design* feito na fase anterior, além de preencher o *Defect Recording log*, encontrado no Apêndice 3, para registrar eventuais erros encontrados, e preencher a linha correspondente a essa subfase no *Time Recording log*, que pode ser encontrado no Apêndice 2.

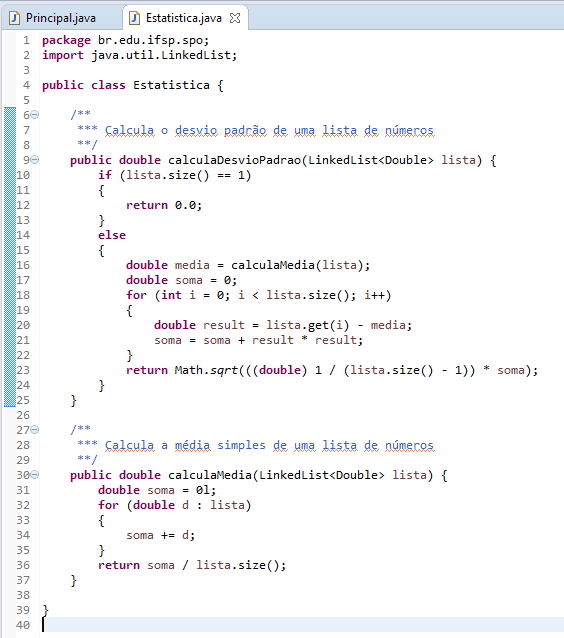
### CÓDIGO

Nas Figuras 11 e 12 está o código do programa que foi codificado em Java, conforme orientação do exercício.

1. Código da classe principal.java



1. código da classe estatistica.java



## COMPILE

Na fase *Compile*, o programa que foi implementado na fase *Code* é compilado e todos os defeitos que forem encontrados, são corrigidos. Além disso, o *Defect Recording log*, encontrado no Apêndice 3, é preenchido afim de registrar eventuais erros encontrados, assim como o *Time Recording log*, que pode ser encontrado no Apêndice 2, tem alinha preenchida referente a subfase *Compile*.

## TEST

Na fase *Test*, todos os testes são executados sem nenhum erro, caso algum erro seja encontrado, deve ser corrigido. Além disso, o *Defect Recording log*, encontrado no Apêndice 3, é preenchido afim de registrar eventuais erros encontrados, assim como o *Time Recording log*, que pode ser encontrado no Apêndice 2, tem alinha preenchida referente a subfase *Test*.

### EVIDÊNCIAS DE TESTE

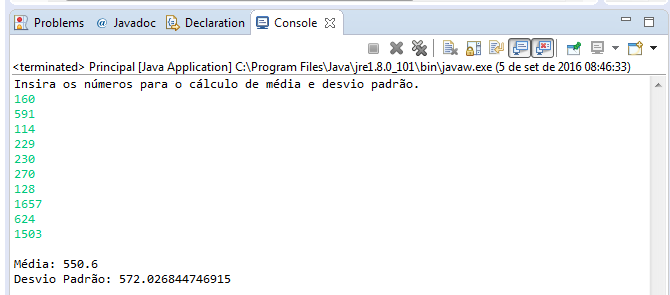
Conforme o PROG1 solicita, ao menos dois testes devem usar os dados das colunas da Tabela 4 e os resultados esperados estão na Tabela 5.

1. DADOS PARA TESTE

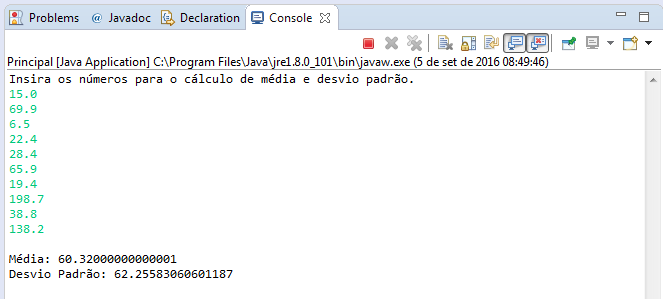
|  |  |
| --- | --- |
| **Estimate Proxy Size** | **Development Hours** |
| 160 | 15.0 |
| 591 | 69.9 |
| 114 | 6.5 |
| 229 | 22.4 |
| 230 | 28.4 |
| 270 | 65.9 |
| 128 | 19.4 |
| 1657 | 198.7 |
| 624 | 38.8 |
| 1503 | 138.2 |

Nas Figuras 13, 14, 15 e 16 estão as evidências de teste conforme os dados fornecidos pela Tabela 4, esses dados estão inseridos na Tabela 5.

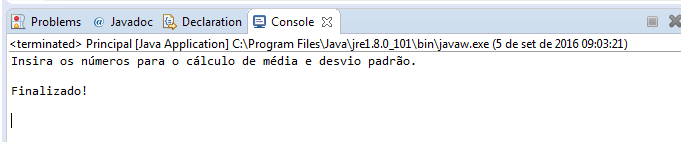
1. evidência de teste da coluna 1



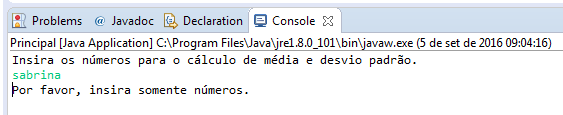
1. evidência de teste da coluna 2



1. finalização sem a inserção de dados



1. fluxo alternativo



1. resultado esperado

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Expected Value** | | **Actual Value** | |
|  | **Mean** | **Std. Dev** | **Mean** | **Std. Dev** |
| Table 1: Column 1 | 550.6 | 572.03 | **550.6** | **572.026844746915** |
| Table 1: Column 2 | 60.32 | 62.26 | **60.3200000000000** | **62.25583060601187** |

# postmortem

A fase *Postmortem* é a última fase do PSP0. Nela o *Project Plan Summary,* encontrado no Apêndice 1, é completado com o tempo real gasto em cada atividade, assim como os defeitos e dados.

Além disso, os outros formulários são revisados em busca de algum erro em cada fase ali registrada. O *Defect Recording log,* que está no Apêndice 3, 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 Apêndice 2, é revisto para assegurar sua veracidade.

ReferÊncias

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

IBM Knowledge Center. **Diagramas de Componentes**, Setembro 2016. Disponível em: < http://www.ibm.com/support/knowledgecenter/pt-br/SS8PJ7\_9.0.0/com.ibm.xtools.modeler.doc/topics/ccompd.html>.

1. PSP0 PROject plan summary

**PSP0 Project Plan Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 05/09/2016 |
| Program | Média e Desvio Padrão | Program # | 1 |
| Instructor | Carlos/ Franciso / João | Language | Java |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Time in Phase (min.)** | **Plan** |  | **Actual** |  | **To Date** |  | **To Date %** |
| Planning |  |  | 23 |  | 23 |  | 10,55 |
| Design |  |  | 135 |  | 135 |  | 61,92 |
| Code |  |  | 42 |  | 42 |  | 19,26 |
| Compile |  |  | 3 |  | 3 |  | 1,37 |
| Test |  |  | 7 |  | 7 |  | 3,21 |
| Postmortem |  |  | 8 |  | 8 |  | 3,66 |
| Total | 360 |  | 218 |  | 218 |  | 100 |
|  |  |  |  |  |  |  |  |
| **Defects Injected** |  |  | **Actual** |  | **To Date** |  | **To Date %** |
| Planning |  |  | 0 |  | 0 |  | 0 |
| Design |  |  | 0 |  | 0 |  | 0 |
| Code |  |  | 1 |  | 1 |  | 100 |
| Compile |  |  | 0 |  | 0 |  | 0 |
| Test |  |  | 0 |  | 0 |  | 0 |
| Total Development |  |  | 1 |  | 1 |  | 100 |
|  |  |  |  |  |  |  |  |
| **Defects Removed** |  |  | **Actual** |  | **To Date** |  | **To Date %** |
| Planning |  |  | 0 |  | 0 |  | 0 |
| Design |  |  | 0 |  | 0 |  | 0 |
| Code |  |  | 1 |  | 1 |  | 100 |
| Compile |  |  | 0 |  | 0 |  | 0 |
| Test |  |  | 0 |  | 0 |  | 0 |
| Total Development |  |  | 1 |  | 1 |  | 100 |
| After Development |  |  | 0 |  | 0 |  |  |

1. PSP TIME RECORDING LOG

**PSP Time Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 05/09/2016 |
| Program | Média e Desvio Padrão | Program # | 1 |
| Instructor | Carlos/ Franciso / João | Language | Java |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Project** | **Phase** | **Start Date and Time** | **Int. Time** | **Stop Date and Time** | **Delta**  **Time** | **Comments** |
| 1 | Planning | 04/09 - 08:15 | 3 min | 04/09 - 08:41 | 26 min |  |
| 1 | Design | 04/09 - 08:45 | - | 04/09 - 11:26 | 135 min |  |
| 1 | Code | 04/09 - 13:18 | - | 04/09 - 15:00 | 42 min |  |
| 1 | Compile | 04/09 - 15:01 | - | 04/09 - 15:04 | 3 min |  |
| 1 | Test | 04/09 - 15:05 | - | 04/09 - 15:12 | 7 min |  |
| 1 | Postmortem | 04/09 - 15:13 | - | 04/09 - 15:21 | 8 min |  |

1. PSP defect recording log

**PSP Defect Recording Log**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Sabrina Fontes Carvalho | Date | 05/09/2016 |
| Program | Média e Desvio Padrão | Program # | 1 |
| Instructor | Carlos/ Franciso / João | Language | Java |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
| 1 |  | | 04/09 |  | 1 |  | 20 |  | Code |  | Code |  | 3 |  |  |
| Description: | | | Problemas de implementação da classe Scanner. | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Project |  | | Date |  | Number |  | Type |  | Inject |  | Remove |  | Fix Time |  | Fix Ref. |
|  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Description: | | |  | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |