SAD - Sistema de Avaliação de Desempenho

Definição dos Testes Unitários

Versão 1

PUC-Rio - Pontífica Universidade Católica do Rio de Janeiro

SAD - Sistema de Avaliação de Desempenho
Log dos Testes Unitários
Projeto de Programação 2014.1

Histórico de Revisões

Versão	Data	Autor	Conteúdo
1	04/08/2014	Daniel Marques	Primeira versão do documento.

Documentos de Referência

Título	Versão	Data	Autor
Busted: http://olivinelabs.com/busted/	2.0	04/08/2012	Olivine-Labs
Lua Rocks (http://luarocks.org/br)	1.0	04/08/2012	Hisham Muhammad

Introdução

Este documento define os testes unitários do Sistema de Avaliação de Desempenho (SAD). Nas seções subsequentes as ferramentas auxiliares usadas nos testes são descritas. Posteriormente, os testes unitários são especificados. Por fim, o Log com o resultado dos testes é apresentado.

Busted Unit Testing

Busted é um framework para testes unitários em Lua desenvolvido pelo Olivine-Labs. este framework pode ser executado pela linha de comando. O Busted disponibiliza diversas funcionalidades como tags, asserts e múltiplos formatos de saída. Para mais detalhes sobre o Framework Busted visite http://olivinelabs.com/busted/#overview.

Execução dos Testes Unitários

Para executar os testes unitários é necessário um *sistema operacional Linux*, a *linguagem de script Lua*, o *Framework Busted* e a *biblioteca Lua lfs*. Os dois últimos podem ser obtidos e instalados através do administrador de pacotes *Lua Rocks* (http://luarocks.org/br).

Uma vez que todos os pré requisitos forem instalados, para executar os testes unitários via linha de comando, basta entrar no diretório onde estão todos os arquivos do sistema SAD e executar o script TestSad.lua por meio do Busted.

Especificação dos Testes

Um conjunto de testes unitários foi definido para cada módulo do sistema SAD. O arquivo TestSad.lua define o script de execução dos testes. Para visualizar a definição e descrição dos testes acesse este arquivo.

Log dos Testes

O Log de saída dos destes unitários é apresentado abaixo no formato TAP (Test Anything Protocol). Note que o framework Busted executa os testes em uma sequência

aleatória, e não na qual os eles foram definidos no script. Além disso, cada teste do log é composto, na maioria das vezes, por múltiplos testes menores.

1..55
1 > #prediction > #meanp > Tests the strategy MeanP with invalid arguments and throw the corresponding error
2 > #prediction > #repeatprevious > Tests the strategy RepeatPrevious with invalid data format and throw the corresponding error
3 > #sad > #Tests the main program with valid inputs.
4 > #dataentry > #loadexperiments > Loads the file instructions_003 and checks its content
5 > #persistence > #savestring > Uses the function SaveString to saves strings into a files and checks the contents
6 > #persistence > #savedata > Tests the function SaveData with invalid arguments and throw the corresponding error
7 > #statistics > #meanabserror > Tests the function MeanAbsError with invalid data format and throw the corresponding error
8 > #persistence > #savedata > Tests the function SaveData with invalid data format and throw the corresponding error
9 > #dataentry > #loadexperiments > Loads the file instructions_005 with invalid format and and throw the correct error
10 > #statistics > #rootmeansqrterror > Tests the function RootMeanSqrtError with invalid arguments and throw the corresponding error
11 > #prediction > #meanall > Tests the strategy MeanAll with a valid input 1
12 > #prediction > #morenumerousall > Tests the strategy MoreNumerousAll with a valid input 2
13 > #persistence > #savedata > Uses the function SaveData to save tables with data into files and checks the contents
14 > #dataentry > #loadexperiments > Loads the file instructions_006 with invalid format and and throw the correct error
15 > #statistics > #rootmeansqrterror > Tests the function RootMeanSqrtError with a valid inputs
16 > #statistics > #rootmeansqrterror > Tests the function RootMeanSqrtError with invalid data format and throw the corresponding error
17 > #persistence > #savestring > Tests the function SaveString with invalid arguments and throw the corresponding error
18 > #prediction > #morenumerousall > Tests the strategy MoreNumerousAll with a valid input 1
19 > #statistics > #meanabserror > Tests the fucntion MeanAbsError with valid inputs
20 > #statistics > #correctness > Tests the function Correctness with invalid data format and throw the corresponding error
21 > #dataentry > #loaddata > Loads the valid file data_003 and checks its content
22 > #statistics > #correctness > Tests the function Correctness with invalid arguments and throw the corresponding error

23 > #prediction > #meanall > Tests the strategy MeanAll with invalid arguments and throw the corresponding error
24 > #prediction > #meanp > Tests the strategy MeanP with invalid data format and throw the corresponding error
25 > #prediction > #repeatprevious > Tests the strategy RepeatPrevious with invalid arguments and throw the corresponding error
26 > #statistics > #correctness > Tests the function Correctness with valid inputs
27 > #prediction > #auxiliary > Tests if the input tables are arrays
28 > #prediction > #morenumerousp > Tests the strategy MoreNumerousP with invalid data format and throw the corresponding error
29 > #prediction > #meanp > Tests the strategy MeanP with a valid input
30 > #prediction > #repeatprevious > Tests the strategy RepeatPrevious with a valid input 3
31 > #prediction > #morenumerousp > Tests the strategy MoreNumerousP with invalid arguments and throw the corresponding error
32 > #prediction > #repeatprevious > Tests the strategy RepeatPrevious with a valid input 2
33 > #dataentry > #loadexperiments > Loads the file instructions_002 and checks its content
34 > #dataentry > #loaddata > Loads the file data_006 with invalid format and throw the correct error
35 > #prediction > #morenumerousp > Tests the strategy MoreNumerousP with a valid input 2
36 > #dataentry > #loaddata > Loads the valid data_002 file and checks its content
37 > #prediction > #morenumerousp > Tests the strategy MoreNumerousP with a valid input 1
38 > #dataentry > #loadexperiments > Loads the file instructions_001 and checks its content
39 > #prediction > #repeatprevious > Tests the strategy RepeatPrevious with a valid input
40 > #dataentry > #loaddata > Opens inexistent files and throw the correct error
41 > #dataentry > #loaddata > Loads the valid file data_005 and checks its content
42 > #prediction > #morenumerousall > Tests the strategy MoreNumerousAll with invalid arguments and throw the corresponding error
43 > #prediction > #meanall > Tests the strategy MeanAll with invalid mean argument format and throw the corresponding error
44 > #statistics > #meanabserror > Tests the function MeanAbsError with invalid arguments and throw the corresponding error
45 > #dataentry > #loaddata > Uses invalid arguments and throw the correct error
46 > #dataentry > #loadexperiments > Opens inexistent files and throw the correct error
47 > #prediction > #meanp > Tests the strategy MeanP with invalid mean argument format and throw the corresponding error
48 > #dataentry > #loadexperiments > Loads the file instructions_004 and checks its content
49 > #prediction > #meanall > Tests the strategy MeanAll with invalid data format and throw the corresponding error
50 > #dataentry > #loaddata > Loads the valid file data_004 and checks its content
51 > #dataentry > #loaddata > Loads the valid file data_000 and checks its content

52 > #prediction > #meanall > Tests the strategy MeanAll with a valid input 2
53 > #dataentry > #loaddata > Loads the valid file data_001 nd checks its content
54 > #prediction > #morenumerousall > Tests the strategy MoreNumerousAll with invalid data format and throw the corresponding error
55 > #dataentry > #loadexperiments > Uses invalid arguments and throw the correct error