SOPAS - Submissão Online Para Análise de Software

José Pedro Silva Pedro Faria Ulisses Costa

Engenharia de Linguagens
Projecto integrado

March 14, 2011

Index

Até agora:

Concretizado até ao ínicio da segunda fase:

- Descrição do sistema ✓
- Modelação formal e informal do problema ✓
- Modelo de dados ✓
- Ínicio da implementação e respectivo tool demo ✓

Motivação e Objectivos

Objectivos para segunda fase:

- Terminar a aplicação web
  - compilar e executar o código fonte submetido
  - guardar e apresentar resultados
- Investigação das métricas existentes
- Scripts auxiliares
- Exploração de um frontend

Index

Index

Implementação: até à segunda fase

Já implementado para o último checkpoint:

- Criação de contas de utilizador (grupo)
- Associação de concorrentes a determinado grupo
- Criação de concursos
- Criação de enunciados (através da interface web ou submetendo em formato xml)
- Inserção de baterias de teste para os enunciados
- Submissão de programas para avaliação

Index

Implementação: linguagens de programação

Configuração de linguagens de programação:

- Estando a linguagem correctamente configurada no servidor, é simples preparar o sistema de submissão para avaliar código submetido nessa linguagem
- Para isso basta inserir o comando usado para compilar e para executar, que por exemplo, em C seria:
  String compilação: *gcc -O2 -Wall #{file}*
  String de execução default: *./a.out*
  String de execução para makefile: *./#{file}*

Index

Implementação: Compilação

- Caso seja necessário compilar o código fonte submetido, é usada a string de compilação definida aquando da configuração da linguagem
- Se for submetido um ficheiro comprimido que inclua um makefile, é executado o comando *make* e, o nome do executável criado é obtido a partir de um script perl

Index

Implementação: Execução

- Para executar o programa para os diferentes inputs, é usada a string de execução simples (no caso de ser submetido apenas um ficheiro) ou a string de execução para makefile (no caso de ser submetido um makefile)
- Para cada input o comando é corrido uma vez
- O output é capturado e comparado com o esperado
- É guardada a percentagem de testes no qual o código submetido passou

Implementação: Apresentação de resultados

- A qualquer altura o utilizador pode consultar os resultados das últimas submissões (suas ou dos restantes participantes)
- Pode também consultar os seus melhores resultados, para cada enunciado

Num Tentativa	Grupo	Concurso	Enunciado	Resultado	Compilou	Terminou a execução
62	Example User	concurso 2 (Activo)	enunciado 1	0.0	true	false
61	Example User	test contest (Activo)	teste2	100.0	true	false
60	Example User	test contest (Activo)	teste2	100.0	true	false

Index

Scripts auxiliares

- Script (em Perl) para obter o nome do executável gerado pelo makefile (para C)
- Script (em Perl) que gera estatísticas relativamente à quantidade de ficheiros submetidos para cada linguagem de programação

Script makefile.pl

- Utiliza o módulo perl *Makefile::Parser* para fazer parse do makefile, e obter o nome do executável gerado
- No caso de não ser definido um nome para o output, retorna *a.out*
- TODO: Suportar mais linguagens par além do C.

Script count.pl

- Dada uma pasta, explora recursivamente os seus directórios, e extraí várias estatisticas relativas á quantidade de número de linhas
  - Número de linhas de código por linguagem
  - Número de linhas comentadas por linguagem
  - Rácio entre linhas de código e número de ficheiros para cada linguagem
  - Percentagem de linguagem mais usadas no projecto
  - ...
- Utiliza o módulo perl *GD* para gerar gráficos

Index

Análise Dinâmica

Log analysis extrair informação dos logs

Testing investigar comportamento através de casos

Debugging bom para detectar defeitos

Instrumentation monitorizar e medir o nível de performance

Profiling investigação sobre o comportamento de um programa (CPU, mem)

Benchmarking comparação de medidas

Análise Estática

Syntax checking atestar a correcção da linguagem

Type checking garantir restrição dos tipos

Decompilation inferir ou descobrir código através de binário

Code metrics tirar conclusões sobre a qualidade

Style checking verificar determinadas regras que se acredita serem boas prácticas

Verification reverse engineering verificar se a implementação cumpre a especificação

Métricas de qualidade de software

$$\text{Line Coverage} = \frac{\text{Nr of test lines}}{\text{nr of tested lines}}$$

$$\text{Decision coverage} = \frac{\text{Nr of test methods}}{\text{Sum of McCabe complexity}}$$

$$\text{Test granularity} = \frac{\text{Nr of test lines}}{\text{nr of tests}}$$

$$\text{Test efficiency} = \frac{\text{Decision coverage}}{\text{line coverage}}$$

Bug Patterns

```
// foo.c file
#include <stdio.h>

int main() {
    char *a = "I like you";
    char *b = "I hate you";

    if(&a < &b) a = *(&a + 1);
    else        a = *(&a - 1);

    printf("%s\n", a);
}
```

```
[ulissesaraujocosta@maclisses:c]-$ gcc -o foo foo.c
[ulissesaraujocosta@maclisses:c]-$ ./foo
I hate you
```

Source Lines of code (SLOC)

Este tipo métricas diz respeito à informação que uma linha de código pode conter.

```
for (i = 0; i < 100; i += 1) printf("hello"); /* How many
    lines of code is this? */
```

- 1 Linha fisica de código (LOC)
- 2 Linhas logicas de codigo (LLOC) (o for e o printf)
- 1 Linha de comentário

Index

Language.C

Language.C é um front-end para Haskell que implementa a linguagem C e muitas extensões do GCC.

```
data CStat = CLabel  Ident CStat [CAttr] NodeInfo
           | CCase CExpr CStat NodeInfo
           | CCases CExpr CExpr CStat NodeInfo
           | CDefault CStat NodeInfo
           | CExpr (Maybe CExpr) NodeInfo
           | CCompound [Ident] [CBlockItem] NodeInfo
           | CIf CExpr CStat (Maybe CStat) NodeInfo
           | CSwitch CExpr CStat NodeInfo
           | CWhile CExpr CStat Bool NodeInfo
           | CFor (Either (Maybe CExpr) CDecl) (Maybe CExpr) (Maybe CExpr) CStat
             NodeInfo
           | CGoto Ident NodeInfo
           | CGotoPtr CExpr NodeInfo
           | CCont NodeInfo
           | CBreak    NodeInfo
           | CReturn (Maybe CExpr)NodeInfo
           | CAsm CAsmStmt NodeInfo
```

Language.C - Utilização (simples)

```
module Main where

import Language.C
import Language.C.System.GCC
import Language.C.Data.Ident
import System.Environment

process :: String -> IO ()
process file = do
    stream <- parseCFile (newGCC "gcc") Nothing [] file
    case stream of
        ( Left error  ) -> print error
        ( Right cprog ) -> print "OK"

main :: IO ()
main = do
    files <- getArgs
    mapM_ process files
```

Index

Terminal Interface

- Criado para facilitar a manutenção e o acesso à base de dados.
- Interface utilizada apenas pelos **administradores** do sistema.

Terminal Interface - Perl

Escolheu-se esta linguagem devido:

- Rápida implementação
- Vasta diversificação de módulos

Desses módulos, usa-se e pretende-se usar:

- **DBIx::Class**
- **Term::Readline**
- **Digest::SHA2**
- **XML::DT**

Estado actual

Exemplo actual da interface:

```
File  Edit  View  Terminal  Help
Dados do User: zacarias

        Email → example-82@railstutorial.org
        Admin → f

Pretender efectuar alterações no user?[S/N]
s

Qualquer campo que não pretenda alterar deixe em branco

→ Inserir novo nome: Zella Douglas
→ Inserir novo email:
→ Admin? [sim]:


Novos dados do utilizador:

Nome: Zella Douglas
Email: example-82@railstutorial.org
Admin: f

Pretender confirmar as alterações? [S/N]
s

User alterado

<======>
Sair? [S/N]
```

Index

Conclusão e trabalho futuro

- Implementar as várias métricas descritas no relatório para C com o FrontEnd
- Melhorar a utilização pelo terminal e as suas funcionalidades
- Tornar a interface utilizador mais intuitiva e mais agradável de utilizar

Perguntas

?