DESARROLLO DE UNA APLICACIÓN WEB PARA LA AUTOMATIZACIÓN DE ENTREGA Y CORRECCIÓN DE TRABAJOS DE PROGRAMACIÓN

Oscar Augusto Chamat Caicedo
Director: Ángel García Baños, Ph.D.

Escuela de Ingeniería de Sistemas y Computación
Laboratorio de Computación Evolutiva y Vida Artificial

Agosto 2014

Índice

Índice

Objetivo General

Desarrollar una aplicación Web que permita la calificación automática de trabajos de programación.

Metodología Usada

Se utilizo para desarrollar el trabajo de grado la metodología XP (Extreme Programing).
Como metodología (forma de hacer las cosas) en lo posible se respeta su orden en la exposición.

Índice

Primer Plan De Entregas

- Pero primero para recortar algo de tiempo de la expocisión:
  - *Historias de usuario, manejo, codificacion, etc.
  - *Y luego cada etapa hasta volver a las historias de usuario(iterar).
  - *Dos iteraciones, etapa de manejo.

	Semana																										
Historia de Usuario	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
A1	■	■	■	■	■	■	■	■																			
A2, A3, A4									■	■	■	■															
A5, A6, A7												■	■	■													
A8, A9, A10															■	■											
A11,A12, A13																	■	■	■	■							
A14, A15, A16, A17, A18																					■	■	■				
A19																								■	■		
A20																										■	■

Figura 1: Primer Plan de entregas

Manejo

- Debido a problemas de diseño y sobreestimación se hizo necesaria lo que es conocido en Programación Extrema como "arreglar xp cuando esta roto" que permitió adaptar la anterior metodología BDD gracias a sus similitudes.
- También permitió eliminar dos historias de usuario -gráficas de grupo-.

Historias De Usuario (Planeación)

- Realizar una tabla estadística con los resultados generados en cada una de las pruebas para todos los estudiantes con un trabajo de programación teniendo en cuenta los apartados mencionados (documentación, estilo, errores de código fuente y errores en los resultados). Y que sea visualizable para cualquier usuario que tenga que ver con el curso a el cual se asigno el trabajo de programación.
- Realizar una tabla estadística con los resultados generados en cada una de las pruebas para todos los estudiantes con todos los trabajo de programación teniendo en cuenta los apartados mencionados (documentación, estilo, errores de código fuente y errores en los resultados). Y que sea visualizable para cualquier usuario que tenga que ver con el curso a el cual se asigno el trabajo de programación.

Segundo Plan De Entregas

	Semana							
Actividad	01	02	03	04	05	06	07	08
B1	■	■	■	■				
B2			■	■				
B3					■			
B4					■			
B5					■			
B6						■		
B7						■		
B8						■		
B9			■	■	■	■	■	

Figura 2: Segundo Plan de entregas

Iteración

- Debido al proceso de manejo con ciertos entregables y la redefinición de algunas partes del proyecto se comienza una nueva iteración.
- En esta iteración se tienen en cuenta de nuevo: Historias de usuario y plan de entregas.

Historias De Usuario

- Buscar una aplicación o escribir el código necesario que cumpla las siguientes condiciones y tenga los siguientes módulos:
  - Integración con el software manejo educativo moodle 2.5(compatible 2.7).
  - Una interfaz que permita a un docente crear trabajos de programación.
  - Un modulo que permita la subida de archivos a la aplicación, para su calificación, configurable para indicar en qué trabajo de programación será calificado.
  - Un sistema que muestre los errores especificados por el compilador del código fuente.
  - Un sistema que permita la creación de pruebas por medio de entradas y salidas por archivos de texto.

Historias De Usuario

- Generar la especificación del formato con el cual se comparará la documentación de los archivos de código fuente para los lenguajes: java 1.6, scheme 5.0 o C++ 4.1.2.
- Crear un sistema de comparación de la documentación de los archivos de código fuente contra el formato especificado y configurable para su correspondiente lenguaje.
- Documentación de código que permita la reutilización de el modulo(métodos y variables públicos). -Común a todos módulos-.
- Especificación de los dos formatos más populares de programación con los cuales se comparará el estilo de los archivos de código fuente para los lenguajes: java 1.6, scheme 5.0 o C++ 4.1.2.

Historias De Usuario

- Crear un sistema de comparación de los estilos de los archivos de código fuente contra el formato especificado para su correspondiente lenguaje.
- Modificación del sistema de creación de trabajos de programación que permita al mismo la funcionalidades de: para un trabajo de programación con su lenguaje estipulado entre java 1.6, scheme 5.0 o C++ 4.1.2 se pueda escoger un estilo de programación y un tipo de documentación que pueda ser aplicado a el lenguaje en particular en un trabajo de programación particular.
- Manuales de uso.-De nuevo común a todos los módulos-

Historias De Usuario

- Modificar el sistema de creación de trabajos de programación para que permita las funcionalidades de: para un lenguaje estipulado entre java 1.6, scheme 5.0 o C++ 4.1.2 se pueda generar un trabajo de programación que al enviar una solución se pueda en forma de varios archivos(como por ejemplo diferentes clases en diferentes archivos en el caso de Java).
- *Módulo que usando los módulos desarrollados para la calificación de: la documentación, el estilo, los errores del código fuente y los resultados de las pruebas unitarias genere los datos necesarios para llenar la tabla descrita.
    - Interfaz gráfica que revele un informe de los resultados de las pruebas.

Historias De Usuario(Tareas No Finalizadas)

- Generación de los artefactos de documentación más populares para Extreme Programming (adaptación de los antiguos artefactos incompletos).
- Traducir aplicación de ingles a español.

Historias De Usuario (Tareas No Finalizadas)

- Luego de terminar la funcionalidad de la aplicación hay que mejorar la calidad de los instaladores e incluir la creación del demonio de la actividad 2.
- Generar las conclusiones del proyecto.

Índice

Diseño

- Se decide usar el lenguaje nativo de moodle (php) y en el proceso se descubren aplicaciones que podrían servir como base:
  - Se revisan las aplicaciones para escoger la mejor (se encuentran en el estado del arte):
    - Schemeassessment.
    - Prakomat.
    - Tamarin.
    - google summer code Onlinejudge / Epaile.
    - Online Judge 2 plugin for Moodle 2 de Sun Zhigang para moodle 2.3, 2.4.
    - AutoGrader.

Diseño

- Para mejorar la conexión entre demonios y la seguridad del cliente ejecutado en moodle se utilizo:
  - La solución salio de la forma en que se conectan los diferentes demonios a el sistema de domjudge-judgehost (conexión directa de los clientes a la base de datos del servidor).

Diseño

- De este sale la metáfora del sistema:

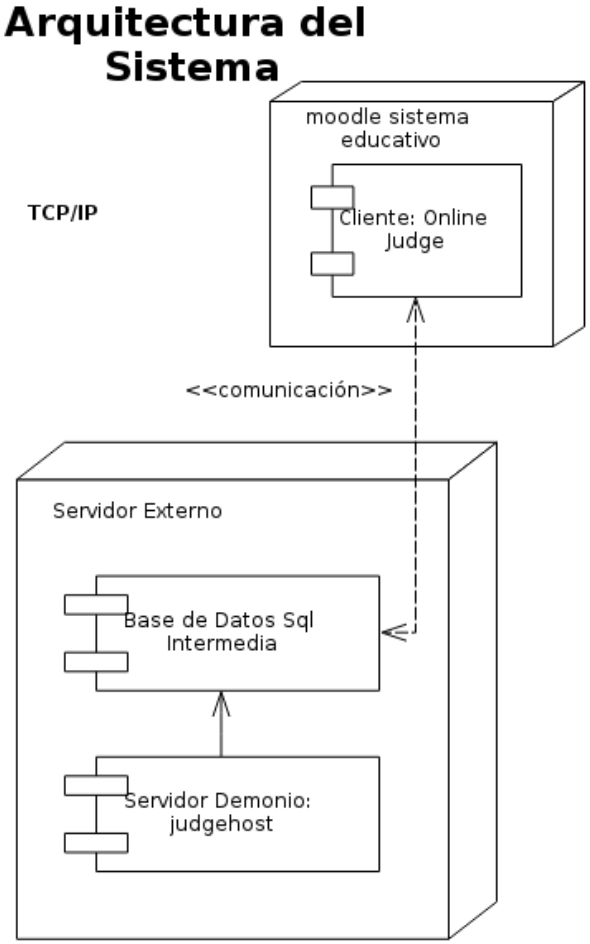


Figura 3: Arquitectura del sistema

Índice

Aplicación Base Moodle

- Como se le dio solucion a las historias de usuario. Y las particularidades de este desarrollo.

Aplicación Base Moodle

- Se escoge como aplicación base en moodle (cliente) la aplicación Online Judge 2 plugin for Moodle 2:
  - De las aplicaciones disponibles sirve para la versión más reciente de moodle (2.4).
  - Además de las funcionalidades descritas cuenta la posibilidad de asignar diferente peso a cada caso de prueba.
  - Y de proveer feedback cuando se tiene un caso de prueba fallido.
  - Como punto contrario dependía de ideone un servicio online que aunque contara con varios lenguajes no permitía el envió de multiarchivos.
- Se puede configurar en el:
  - Máximo tamaño del envió (MB), la máxima cantidad de archivos enviada(max 1000)
  - El lenguaje (C++, Java, Racket).
  - El tiempo máximo de ejecución(el sistema corta la ejecución).
  - Configurar si la compilación cuenta en la calificación de los estilos (documentación, indentación).
  - Radio de error de presentación.

Calificación De Varios Archivos

- Para soportar la calificación de un envió con archivos múltiples se modifico la aplicación cliente eliminado a ideone y conectando a domjudge-judgehost atraves de la metáfora del sistema.
- Domjudge-judgehost
  - Sirvió como maqueta para la ejecución de las demás funcionalidades del proyecto.
  - Permite la ejecución de un entorno chroot que aumenta en forma considerable la ejecución de los envíos.
  - Puede ejecutar múltiples demonios en la misma maquina o en diferentes maquinas y que todos puedan calificar al tiempo ya sea para aumentar la eficiencia o para evitar fallas del sistema.

Calificación De Varios Archivos

- Luego de su instalación se pudieron implementar fail-safe's en los estados de la base de datos que ayudan a que el demonio de Domjudge-judgehost no se estanque por funcionamiento no esperado del demonio del lado del cliente o viceversa.
- Domjudge-judgehost permite la inclusión de cualquier otro lenguaje siguiendo el manual incluido en el informe.
- Por la simple implementación y la forma de comunicación (redirección a archivos) de las partes del judgehost cualquiera de estas puede ser simplemente reemplazada o reescrita o incluso añadida con relativa simpleza.

Formato De Calificación De La Documentación

- Hay muchísimos estilos de documentación; ninguno estándar reconocido.
- Se decide entonces utilizar la popularidad y la usabilidad:
  - Se decide usar el estilo oficial de doxygen @etiqueta.
  - Doxygen sirve como generador de documentación automático en: C++, C, Objective-C, C#, PHP, Java, Python, IDL - Corba, Microsoft, y los sabores UNO / OpenOffice -, Fortran, VHDL, Tcl.
  - La idea es que sea algo que se aplique más allá de la academia.

Sistema De Calificación De La Documentación

- A julio-2014 para la implementación que considera la calidad de la documentación como una magnitud medible no es posible encontrar antecedentes.(Se encuentran forzadores de documentación).
- La idea es simple:

$$calificacion = \frac{etiquetas-validas}{etiquetas-posibles}$$

Figura 4: Radio de etiquetas validas

Sistema De Calificación De La Documentación

- Se reduce el subconjunto de etiquetas calificables a @author, @version, @copyright, @license, @package porque la inclusión correcta de la calificación de las demás etiquetas requeriría la creación de parser especial para cada lenguaje en el juez.
  - Un parser especial porque los parser que generan los arboles sintácticos eliminan los comentarios.
  - El parser es necesario porque una función en c++ y en racket pueden tener parámetros de valor default y estos valores default ser objetos(o estructuras) por si mismos y tener métodos con etiquetas de documentación y así sucesivamente.
    - (define (f [arg (define (func...)]) ...
    - int f(int x=0, struct a = struct foo{ bool ...});
- Implementado en java.

Formato De Calificación De La Indentación

- Se utiliza una forma del archivo formateada pretty-print(no exactamente solo indentación).
- Los formatos como es costumbre se especifican mediante un párrafo sencillo en lenguaje natural.
  - bloques de codigo y parentesis de cerrada.

```
(dotimes (i 10)
  (if (evenp i)
      (do-something i)
      (do-something-else i)))
```

Figura 5: Estilo Lisp

Sistema De Calificación De La Indentación

- Sin antecedentes para convertir su medida en una magnitud.
- La cantidad de programas que realizan pretty-print sin involucrar una gui son muy limitados:
  - Se utilizan casos especiales:
    - En C++/Java se utiliza indent (gnu) configurable por argumentos a la consola (diferentes estilos).
    - En racket la única solución con pretty-print correcto y sin ciclos infinitos es la creación de un script que utiliza la función (pretty-print) del propio racket.

Sistema De Calificación De La Indentación

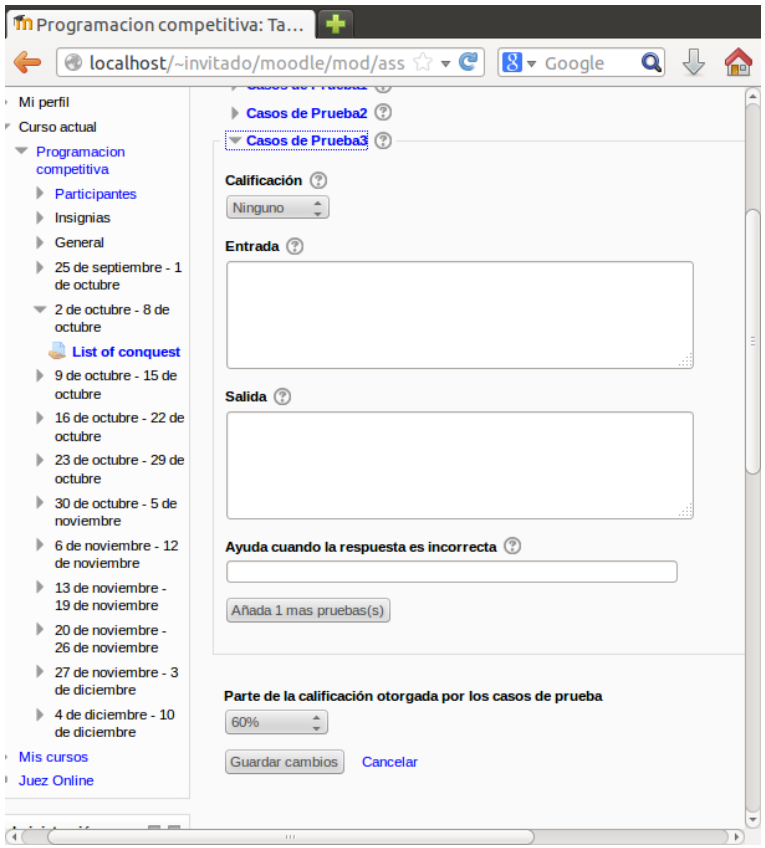
- La implementación de la comparación entre los archivos indentados o no indentados no es simple tradicionalmente:
  - Se comparan carácter a carácter haciendo un espacio de más catastrófico
  - O se comparan omitiendo los espacios.
- Entonces se comparan como dos cadenas de ADN (Python):
  - Encuentra la subsecuencia contigua más larga (LCS).
  - Recursivamente se aplica a las piezas de las secuencias a la izquierda y a la derecha de la subsecuencia coincidente.
- El valor de igualdad se calcula como:

$$\frac{2(largo - coincidencias)}{suma - largos - cadenas}$$

Figura 6: Radio de igualdad

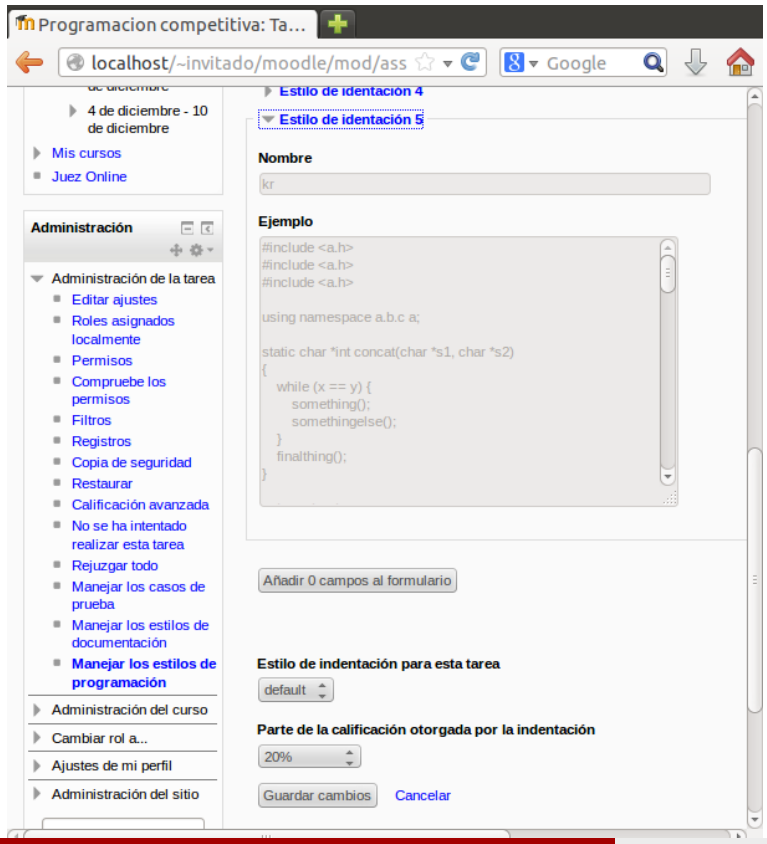
Configuración De Un Trabajo De Programación (Casos De Prueba)

- Todos extienden de la misma clase y se usan extendiendo la navegación a nuevos menús.



Configuración De Un Trabajo De Programación (Indentación)

- El sample de cada estilo es representativo contiene lo necesario para indentar bien cualquier archivo (Clases, métodos, objetos, if, estructuras de control, etc).



Configuración De Un Trabajo De Programación (Documentación)

- Cada profesor que cree su propio estilo pasa a ser el dueño del mismo, cualquier otra persona lo puede usar en su tarea pero solo el dueño lo puede modificar.

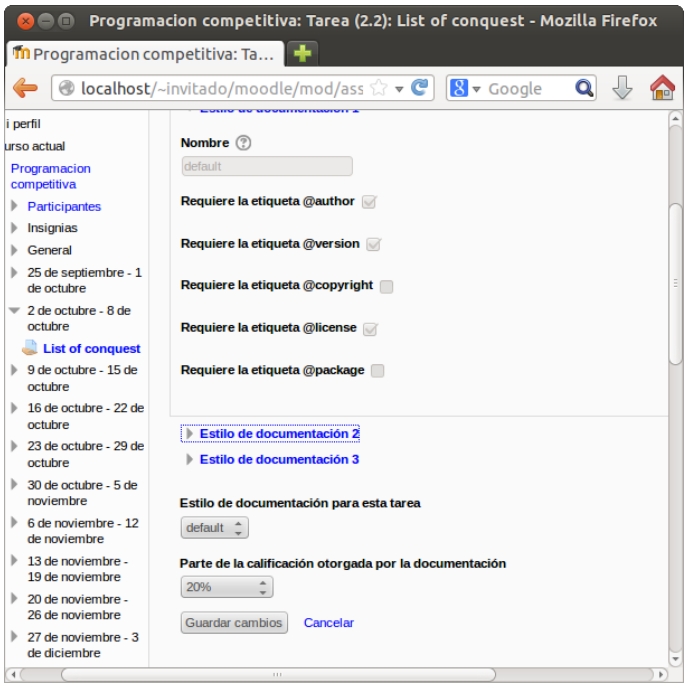


Figura 9: Configuración indentación de tarea

Generar / Mostrar Notas Individuales

- El encargado de calcular la nota final es el demonio judged.php que se encuentra del lado del cliente que tiene en cuenta:
  - La compilación.
  - El estado de los testcases.
    - Presentation Error
  - La nota de los estilos.
  - Los diferentes porcentajes de cada uno.

Generar / Mostrar Notas Individuales

- Se encarga de mostrar los diferentes estados de calificación de una trabajo de programación.
  - Juzgando, en espera, fallido, error de presentación, etc.
- El cronjob de moodle se encarga de revisar si un envió lleva más de un tiempo (configurable desde la aplicación) en espera y entonces envía emails que se pueden configurar para que lleguen a el administrador de moodle o a todas las personas que pueden crear trabajos de programación.

Generar / Mostrar Notas Individuales

- Muestra los resultados específicos por caso de prueba:

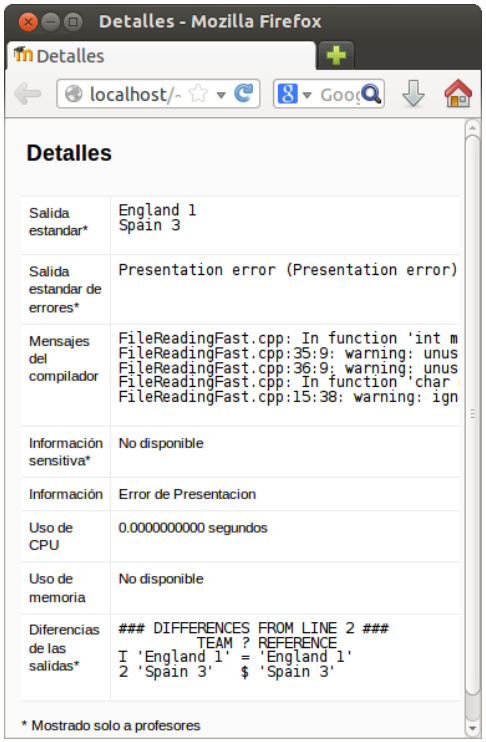


Figura 10: Detalles De La Calificación De Un Caso de Prueba

Índice

Instalación base para pruebas.

- Para realizar pruebas de un sistema con base de datos se necesita una base de datos limpia:
  - En moodle cuando se instala su suit de pruebas el se encarga de eso.
  - En el servidor Domjudge-Judgehost fue necesaria la creación de un apartado en la instalación para crear esta base de datos con sus usuarios.
- Se genera una librería que sirve para simplificar la generación de: trabajos de programación, casos de prueba y estilos.

Librería de generación

- Se necesita probar la librería de generación para pruebas:
  - Se generan 3 trabajos de programación con diferentes lenguajes y notas máximas.
  - A una de estos trabajos se le generan tres casos de pruebas.
    - Uno de los casos de pruebas se borra.
    - Uno se modifica.
  - Se genera dos estilos de documentación.
    - Uno se modifica.
    - El otro se borra.
  - Se le cambia el estilo de documentación al trabajo de los 3 casos.
  - También el de indentación.
  - Se hace un envió (cambiando el usuario) a el trabajo de programación.
  - Se chequea la calificación del envió y de cada uno de sus casos de prueba.
- Todos y cada uno de los pasos descritos son probados con assert contra los cambios que deberían realizar en la base de datos.

Indentación

- Para la indentación hay un archivo de prueba similar para cada lenguaje.
- Se crean 5 trabajos de programación todos con el mismo estilo de indentación y mayor porcentaje de valor en la indentación para ver las discrepancias.
  - El envió al primer trabajo de programación debe volver con calificación baja de indentación.
  - El segundo tiene que tener nota media.
  - El tercero tiene que tener nota perfecta.
  - El cuarto contiene errores de compilación (faltan paréntesis, o corchetes o comillas)
    - viene a probar que el ni el indentador ni el comparador tienen problema con estos archivos
    - o se van a quedar en un ciclo infinito
  - El quinto contiene un archivo txt que no tiene nada que ver con el lenguaje.
    - Prueba lo que el cuarto con una estructura diferente.

Documentación

- Tiene una estructura igual que la prueba de indentación.
- Se crean 5 trabajos de programación todos con las mismas características que en indentación.
  - El primero tiene que volver con 0 de nota de indentación.
  - El segundo tiene que tener nota media.
  - El tercero tiene que tener nota perfecta.
  - El cuarto contiene errores de compilación.
    - viene a probar que el comparador no tiene un problema con estos archivos.
    - o se van a quedar en un ciclo infinito
  - El quinto contiene un archivo txt que no tiene nada que ver con el lenguaje.
- El envió de nota perfecta tiene 3 archivos independientes uno en el que la documentación esta en la cabecera del archivo, *el otro tiene etiquetas dentro de strings y su documentación esta en la mitad del archivo, *el final tiene la documentación separada por etiquetas repartida en el archivo.

Test extras

- Se incluye un test que revisa la **seguridad del sistema**(configuración correcta de los permisos) viendo que ninguno de los envíos pueda acceder a ningún archivo importante del sistema.(Un archivo por lenguaje).
- Se incluye un test con **porcentaje 0** en todos sus apartados para ver que esto no modifique la forma de calificación o genere problemas de ejecución.

Test extras

- Se incluye una prueba que asegura la funcionalidad de la posibilidad de **cambiar la notas de un envió luego de calificado**:
  - Se cambia la nota de la documentación.
  - Se cambia la nota de la indentación.
  - Se cambia el estado de calificación de todos los casos de prueba (documentados abajo).
  - Y se asegura que todos estos cambios tenga impacto en la nota total.
  - Se cambia la nota total.

Casos de Prueba de Resultado

- Se tienen en cuenta los diferentes resultados que se pueden generar al calificar un envió y se realizan pruebas para estos.
- Todas las pruebas que se comentaran son realizadas 3 veces una por cada lenguaje en un archivo diferente.
- También se chequean los estados de cada prueba contra la base de datos.

Casos de Prueba de Resultado

- Test de envió aceptado;
  - Se crea un trabajo de programación con 3 casos de pruebas.
  - Se realiza un envió.
  - Se chequea que su calificación y la de sus casos de prueba sea “accepted”, correcta.
- Test de error en tiempo de ejecución;
  - Mismo proceso que en envió aceptado pero 2 casos de prueba y
  - Su resultado debe ser “runtime-error”, error en tiempo de ejecución.
- Test de error de cantidad de tiempo excedida:
  - Igual que error en tiempo de ejecución pero se espera un “time-limit-exceed”.
    - El tiempo de ejecución del programa a superado el limite establecido para el trabajo de programación.
    - El sistema termina la ejecución de estos envíos.

Casos de Prueba de Resultado

- Test de respuesta incorrecta.
  - Se espera “wrong-answer”, respuesta incorrecta.
- Test de error de presentación
  - Se espera “presentation-error”, error de presentación.
  - Se cambia el radio de error de presentación a un valor medio para que este valor se represente en la nota total.
- Test de error de compilación
  - Se espera “compilation-error”, error de compilación.
  - Cada archivo tiene 2 trabajos de programación uno en el que la compilación cuenta en la nota de los estilos y otro en el que no cuenta.
    - Se evalúa cada uno.

Casos de Prueba de Resultado

- Test de error de cantidad de memoria excedida:
  - Se espera “memory_limit_exceed”, cantidad de memoria excedida.
  - En los lenguajes interpretados la maquina virtual es la primera que activa el error y el juez registra un error en tiempo de ejecución.
- Test de error de cantidad de output excedida.
  - Se espera “output_limit_exceed”. El programa genero demasiada salida.
  - Debido al tamaño máximo de la salida este test siempre termina con el resultado time-limit-exceed o memory-limit-exceed.

Índice

Conclusiones

- La planeación en problemas de largo alcance es una tarea complicada que solo se mejora con experiencia y con constante reimaginación de como lidiar con todos los problemas posibles que se pueden encontrar en cualquier despliegue.
- Un sistema de calificación automático es un punto critico en la educación ya que promete mejorar de forma considerable la calidad de los estudiantes de sistemas por que les permite llevar la teoría a la verdadera practica con parámetros claros y realmente probados sobre sus programas. Además al facilitar la calificación aumenta de forma significativa la cantidad de practica que puede tener un estudiante programando.
- El hecho de que los trabajos sobre la valoración numérica de los estilos ya sea de indentación o de documentación del código sean prácticamente inexistentes presenta un vació interesante en las ciencias de la computación y como se están midiendo dos valores que pueden tener tanta importancia en la escalabilidad de cualquier aplicación.

Conclusiones

- La utilización de esta herramienta permite que los profesores se eviten mucho de trabajo repetitivo y poder concentrarse en áreas más interesantes de la enseñanza e incluso reducen altamente la posibilidad del error humano en las calificaciones de los estudiantes.
- La utilización de código libre permite que una sola persona cree aplicaciones complejas y con algún nivel de confiabilidad con un esfuerzo menor al requerido al desarrollar estas aplicaciones desde el inicio.
- Las practicas de desarrollo de software basadas que alientan realizar las pruebas primero(TDD, BDD y recientemente XP) se nos han venido vendiendo como la solución a todos los problemas del software actual, pero como todo ingeniero de software debe saber no existe un "Golden Hammer" el enfoque excesivo en las pruebas no es la solución a todo. Este acercamiento (en mi opinión y la de otros desarrolladores "tddIsdeath") si se utiliza de forma excesiva puede causar que la calidad de los diseños se vea reducida incluyendo muchos objetos de servicio y/o compromisos en el diseño que reducen la calidad del proyecto.

Índice

Trabajos futuros

- Extender la funcionalidad de las opciones de calificación del estilo de indentación para permitir más opciones y por lo tanto permitir la especificación de estilos más flexibles que se podrían acercar más a las diferentes configuraciones permitidas por los formateadores de estilo del entorno de programación eclipse. Y también poder extender las posibilidades de la calificación de los comentarios.
- Permitir que los estilos de programación e indentación puedan ser descargables y que se puedan compartir entre profesores.
- Incluir en el entorno de calificación restricciones sobre la cantidad de memoria permitida en un programa sin que estos sean los estándares del lenguaje de programación utilizado en el trabajo de programación.
- La posibilidad de un curso autocalificable para aprender técnicas sobre algún tema en especial, en donde podría tomar como ejemplo portal USACO.

Trabajos futuros

- Mediante la adición del tiempo de envió de un trabajo de programación como algo calificable se podrían organizar maratones de programación para todas las personas con acceso al la aplicación de forma simple.
- Incluir más lenguajes de programación basándose en algunos de los scripts ya existentes para su compilación y extendiendo la funcionalidades implementadas para la calificación del estilo y de la documentación.
- Dar la posibilidad a los usuarios estudiantes de incluir pruebas o "testcases" para un trabajo de programación en concreto y así no solo aumentar la cantidad de pruebas realizables para un trabajo de programación sino también poder incluir una bonificación para la persona que cree estos nuevos casos de pruebas.

Trabajos futuros

- Incluir un foro que tenga información de links e información sobre las diferentes técnicas de programación usadas en cada uno de los trabajos de programación cuyo funcionamiento sea parecido a Algorithm Tutorials de topcoder.
- Permitir la capacidad de no solo generar pruebas para el código que tengan en cuenta su salida para una entrada dada sino también que se puedan realizar pruebas unitarias sobre el código mandado y algunas pruebas automáticas sobre el funcionamiento del software como por ejemplo las realizadas por selenium. Lo anterior no solo para los usuarios que crean los trabajos de programación sino también permitir que los estudiantes envíen sus propias pruebas de código.

Trabajos futuros

- Incluir la posibilidad de usar PMD(no tiene nombre oficial) sobre el código enviado esto para detectar posibles bugs, código muerto o código subóptimo o sobrecomplicado.
- La posibilidad de ordenar las envíos de soluciones por tiempos de ejecución y por uso de memoria para conocer cuales son las mejores y además incluir la posibilidad de calificar la limpieza del código.
- Mediante la adición de la posibilidad de revisar el código de diferentes envíos hechos por otros estudiantes a un trabajo de programación en concreto los estudiantes podrían enviar su código la cantidad de veces que quieran antes de la fecha de entrega limite así pueden cambiar su código teniendo en cuenta cosas que aprendieran del código de los demás o de los comentarios sobre su propio código.