Proyecto de Trabajo Profesional

Título

Plataforma educativa: Batalla Espacial 2.0

Autor

Mariano Patricio Tugnarelli
padrón 78945
marianotugnarelli@gmail.com

Tutores

Lic. Rosa Graciela Wachenchauzer (Tutor)
rositaw@gmail.com

Ing. Juan Gabardini (Co-tutor)
jgabardini@gmail.com

Índice

Objetivo

Construir una plataforma que pueda ser utilizada como ejemplos concretos de temas teóricos y como framework para desarrollar ejercicios y trabajos prácticos en los cursos introductorios de ingeniería de software.

Introducción

La concepción del presente trabajo fue construida a partir de tres perspectivas complementarias. Una perspectiva teórica, una perspectiva práctica y una perspectiva didáctica.

La perspectiva teórica está dada por el trabajo de Bertrand Meyer, presentado en su libro "*Touch of Class. Learning to Program Well with Objects and Contracts*" ***[Meyer09]***. La propuesta de Meyer se consolida como el marco teórico de este trabajo.

La perspectiva práctica es el resultado de trabajar en la industria, particularmente en la construcción de software, durante el transcurso de mis estudios universitarios.

La perspectiva didáctica se fue consolidando a partir de mi desempeño como auxiliar docente en la Facultad de Ingeniería de la Universidad de Buenos Aires y en la Universidad de Tres de Febrero.

Desde la perspectiva teórica

Bertrand Meyer describe en su libro las técnicas que ha aplicado por varios años en el curso "*Introduction to Programming*" en el ETH Zurich (*Eidgenössische Technische Hochschule Zürich, Swiss Federal Institute of Technology Zurich*), tomado por todos los estudiantes que ingresan a la carrera de Ciencias de la Computación (*Computer Science*).

Las ideas innovadoras que Meyer utiliza para estructurar el curso resultan interesantes y especialmente atractivas por encontrar sustento en observaciones realizadas en el ámbito profesional y académico (observaciones que serán descriptas en las siguientes secciones). Estas ideas se materializan en los siguientes conceptos:

- Currículo invertido, también llamado *outside-in* (de afuera hacia adentro).
- Uso generalizado de técnicas orientadas a objetos y guiadas por modelos (*object-oriented and model-driven techniques*).
- Diseño por contrato.

- Planteo y discusión de las incumbencias, problemáticas y responsabilidades de la ingeniería de software (*software engineering concerns*) desde el comienzo.

Los conceptos planteados por Meyer como lineamientos generales del plan de estudio formalizan las percepciones recogidas en el ámbito práctico y didáctico, razón por la cuál, se adopta su propuesta como marco teórico para el desarrollo de este trabajo.

Desde la perspectiva práctica

En el ámbito profesional se advierten múltiples evidencias que respaldan los lineamientos expuestos por Meyer. Estas evidencias se podrían sintetizar en las siguientes observaciones:

- Construcción de módulos en lugar de aplicaciones completas.
- Utilización intensiva y extensiva de bibliotecas de componentes y *frameworks*.
- Trabajo en equipo.
- Aseguramiento de la calidad del software.

Desarrollemos cada una de estas observaciones indicando su correlación con los conceptos de la perspectiva teórica.

Desde la perspectiva práctica: Construcción de módulos

Comenzaremos analizando el hecho de que son muy pocas las oportunidades de construir una aplicación completa desde cero (*from scratch*), desde sus cimientos, partiendo del punto de entrada de la aplicación, teniendo la posibilidad de seleccionar la tecnología, definir la arquitectura, diseñar todos los módulos y construir toda pieza de software que constituya la aplicación. Son ejemplos de esto la construcción de pequeños scripts utilitarios para manipular archivos, monitorear recursos o censar el tráfico en una interfaz de red.

Este primer escenario resulta poco habitual y se limita a la resolución de algunos problemas aislados.

En contraposición, el escenario que se presenta con mayor frecuencia involucra la construcción de pequeñas piezas de software, módulos que se ensamblan con otros, en un contexto complejo. Un contexto que nos impone restricciones tecnológicas y decisiones arquitectónicas. Un contexto que nos exige hacer frente a responsabilidades para con otros módulos y que nos provee de componentes y servicios.

Este segundo escenario nos presenta desafíos diferentes. Aparecen nuevas responsabilidades que en el primer escenario no existen:

- Responsabilidades para con el resto del equipo. Como proveedor de un módulo que otro miembro del equipo utilizará para construir su propio módulo. Fundamentalmente se requiere que el módulo:
  - Tenga definida una interfaz clara.
  - Sea cohesivo.
  - Provea información respecto a su forma de utilización, ya sea como documentación formal del contrato expuesto por su interfaz y/o como ejemplos de su utilización.
  - Cuente con pruebas unitarios que ofrezcan mínimas garantías de su corrección funcional.
- Responsabilidades para con la arquitectura de la aplicación. Como proveedor de un módulo que se ensambla con otros en un marco definido, el módulo deberá ocupar el lugar dentro de la estructura de módulos que la arquitectura definió.

Desde la perspectiva práctica: Biblioteca de componentes y frameworks

En la actualidad uno de los aspectos que caracteriza a la construcción de software es la utilización intensiva y extensiva de bibliotecas de componentes y *frameworks*.

Consideramos que es intensiva por el hecho de ser permanente su uso. Es decir, en todo momento interactuamos con las piezas de software provistas por una biblioteca o un *framework*.

Consideramos que es extensiva porque se usan en todos los contextos para resolver múltiples problemas.

Existen innumerables bibliotecas de componentes y *frameworks*, en su gran mayoría originados en torno a comunidades virtuales y organizados como proyectos *open source,* que nos proveen piezas de software con altos niveles de reutilización y que nos permiten colaborar activamente en su desarrollo.

Son frecuentes las oportunidades que se presentan para utilizar bibliotecas de componentes y *frameworks* maduros con altos niveles de calidad. Así como también, las oportunidades de utilizar diferentes soluciones (bibliotecas y *frameworks*) para resolver el mismo problema.

También se presentan oportunidades para realizar aportes a estos proyectos a partir de:

- Detección de fallas.
- Identificación de defectos.
- Corrección de defectos.

- Identificación de mejoras.
- Construcción de extensiones desarrolladas en el contexto de un problema particular que la pieza original no tenía contemplado.

Estas experiencias permiten formar amplios criterios tanto de uso como de construcción.

Resulta interesante esta distinción por el hecho de que ambos puntos de vista son complementarios. La experiencia de utilizar una pieza de software y la experiencia de ser proveedor de una pieza de software son complementarias porque existe un efecto de retroalimentación entre ellas.

En el rol de usuario:

- Experimentamos el proceso de aprendizaje de uso de la pieza.
- Percibimos sus atributos de calidad, como por ejemplo corrección, claridad, robustez y flexibilidad.
- Identificamos ventajas y desventajas de su diseño e implementación.

En el rol de proveedor:

- Analizamos los requerimientos.
- Definimos la interfaz.
- Evaluamos los posibles diseños e implementaciones.
- Generamos la documentación.

Desempeñar ambos roles iterativamente permite comprender intrínsecamente, desde la propia experiencia, las necesidades a satisfacer y las dificultades asociadas a satisfacerlas, el proceso de diseño de una interfaz y el ejercicio de su utilización. Resulta en consecuencia un refinamiento progresivo de ambas habilidades. Se desarrollan por igual mejores capacidades para construir módulos como también para utilizarlos.

Desde la perspectiva práctica: Trabajo en equipo

El trabajo en equipo es una constante presente en todo desarrollo de software de mediano y gran porte. Desde pequeños equipos locales hasta grandes equipos distribuidos alrededor del mundo.

En este contexto se requiere compromiso, colaboración y comprensión de los criterios de descomposición y de composición modular.

La descomposición modular permite dividir el problema en partes e implementar la solución de cada una de estas partes como módulos, haciendo posible la división del trabajo entre los miembros del equipo.

La composición modular hace posible ensamblar la solución a un problema a partir de los módulos construidos por diferentes miembros del equipo o por bibliotecas de componentes y *frameworks* provistos por terceros.

La colaboración entre los miembros del equipo potencia las capacidades individuales. La utilización de un módulo construido por otro miembro del equipo debe ser un ámbito cotidiano de aprendizaje y mejora. Aprendemos del resto del equipo y aportamos nuestro conocimientos para aumentar la calidad del módulo.

El compromiso de cada uno de los miembros del equipo debe reflejarse en la calidad de los módulos construidos y en el respeto con el resto del equipo al utilizar un módulo construido por otro miembro.

Desde la perspectiva práctica: Aseguramiento de la calidad

Como mencionábamos en la sección anterior cada miembro del equipo debe estar comprometido con la calidad de todo módulo que el equipo construye, así como también con la calidad del producto final.

Las pruebas unitarias automatizadas son un primer paso. Además de aportar al aseguramiento de calidad (permitiendo detectar errores en etapas tempranas de las construcción, haciendo posible la ejecución de pruebas de regresión, etc.) sitúan temporalmente en el rol de usuario al constructor del módulo.

Técnicas como *TDD* (*Test Driven Development*, desarrollo guiado por pruebas) proponen utilizar las pruebas unitarias automatizadas como herramienta de especificación, para diseñar la interfaz del módulo.

Desde la perspectiva didáctica

A partir de las diferentes experiencias en el ámbito educativo fui percibiendo e identificando problemáticas recurrentes en el proceso de aprendizaje. Para resolver estas dificultades ensayamos con los equipos de docentes múltiples aproximaciones alineadas con las observaciones descriptas en la perspectiva práctica y en concordancia con la propuesta de Meyer, dando origen al siguiente conjunto de estrategias:

- Proporcionar ejemplos concretos y simples, pero completos y descriptivos.
- Platear desafíos en un contexto atractivo.
- Usar primero y construir después.
- Plantear estrategias de trabajo que contemplen las diferencias en la formación y los conocimientos adquiridos previamente (*background*) por los alumnos.
- Definir lineamientos que promuevan el trabajo en grupo.

- Evitar introducir un elemento nuevo de complejidad en el proceso de aprendizaje dado por el idioma.

Explicaremos ahora el propósito de cada una de ellas.

Desde la perspectiva didáctica: Los ejemplos que usamos

Los ejemplos utilizados deben ser elegidos cuidadosamente tal que sean concretos y simples, pero a la vez completos y descriptivos; tanto los problemas como las resoluciones mostradas deben reunir estas características.

Los problemas utilizados para ejemplificar un concepto o introducir un tema deben permitirle al alumno identificar con claridad la problemática bajo análisis; por tanto, deben ser simples y concretos. Pero también es necesario que se presenten en un contexto más amplio, no trivial, que exponga todas las dimensiones del problema y demuestre una aplicación real; deben ser descriptivos y completos.

Las resoluciones mostradas deben ser suficientemente simples y concretas para lograr transmitir los conceptos con claridad. Pero además deben poder describir su aplicabilidad en un contexto más complejo, su impacto sobre el resto del sistema, sus ventajas y desventajas.

Desde la perspectiva didáctica: Contexto atractivo

Para obtener mejores rendimientos de los alumnos es necesario presentar ejemplos y ejercicios que les resulten atractivos e interesantes, que capturen su atención.

Las problemáticas vinculadas de una u otra manera a un contexto lúdico es una aproximación para conseguir esto. Así como también la construcción de módulos en el contexto de aplicaciones similares a aquellas con las que interactúan a diario; aplicaciones que, por ejemplo, tienen interfaces gráficas atractivas, en lugar de basadas en consola.

Desde la perspectiva didáctica: Usar primero y construir después

Como lo señalábamos anteriormente ambas perspectivas, uso y construcción, son complementarias. No obstante, en el proceso de aprendizaje, resulta natural usar primero una pieza de software, entendiendo qué problema resuelve, qué alternativas ofrece, qué ventajas y qué desventajas tiene, para luego comprender cómo diseñarla, cómo construirla y cómo probarla.

En muchas oportunidades los alumnos deben construir soluciones para problemas que no comprenden o que no pueden visualizar. Pero, el usar un módulo que resuelve ese problema les aportan nuevas herramientas prácticas y conceptuales para entenderlo. Además, construir una solución para un problema conociendo otras soluciones mejora sensiblemente la calidad del resultado obtenido.

Por ejemplo tomemos como temática de enseñanza la estructura de datos Conjunto.

Una primera aproximación didáctica, ampliamente usada, involucra describir alguna situación problemática para la que esta estructura resulta ser central en la solución. A continuación se explican las características del Conjunto, describiendo sus operaciones y axiomas, indicando cómo se aplica a la solución. Finalmente se presentan múltiples variantes de implementación de esta estructura de datos, haciendo un análisis comparativo entre las mismas.

En esta contexto a los alumnos no les resulta fácil llevar adelante las implementaciones de Conjunto porque no comprenden el problema de fondo, confunden las post-condiciones de las operaciones porque no perciben con claridad cuál es el objetivo de las mismas y el contexto de su utilización, dejan de lado operaciones porque que no les resultan necesarias y no separan correctamente la interfaz de la implementación simplemente porque están aprendiendo ambas cosas (interfaz e implementación) a la vez.

Esto es análogo a solicitarle a una persona que nunca vio siquiera una bicicleta que construya una, explicándole antes (claro está que con el máximo de los detalles) cuál es su propósito, qué características tiene, cómo se usa y cuáles son los principios mecánicos de su funcionamiento. Tal vez alcance el objetivo propuesto, pero el esfuerzo invertido sería enorme en comparación con el resultado obtenido.

Una mejor aproximación, involucra que antes de explicar las variantes de implementación se les proporcione una implementación correcta, simple y completa de Conjunto, describiendo sólo su interfaz (es decir ignorando su implementación, pasándola por alto temporalmente) . Luego se le presentan al alumno problemas concretos para que los resuelva prácticamente usando el componente proporcionado (predicamos con el ejemplo el concepto de ignorancia selectiva). Cuando los alumnos han sido capaces de resolver múltiples problemas siendo usuarios de un módulo que expone la interfaz de Conjunto es oportuno explicarles su implementación (el secreto del módulo, lo que antes pasamos por alto), como así también otras variantes posibles.

Para ese entonces, la única complejidad que tiene llevar adelante la construcción de un Conjunto está asociada con los detalles de implementación y no con su interfaz. El alumno tiene claro (está seguro) cómo es el contrato que debe honrar porque él fue antes usuario del mismo. Esto suaviza sensiblemente la curva de aprendizaje.

Volviendo a la analogía antes propuesta, los resultados van a ser mejores (tanto en términos de eficiencia de recursos como de calidad) si a la persona a la que le encargamos la construcción de la bicicleta, en primer lugar le damos una bicicleta, le enseñamos a usarla y la usa, para que finalmente cumpla con el trabajo encomendado.

Desde la perspectiva didáctica: Conocimientos previos

Son significativas las diferencias en el *background* de cada uno de los alumnos en los primeras materias vinculadas a la Ingeniería de Software. Existen alumnos que tienen una formación formal previa, otros que tienen conocimientos adquiridos informalmente en el ámbito laboral o siendo autodidactas y otros que tienen su primer acercamiento en el contexto universitario.

Es necesario contemplar este abanico de posibilidades, percibiendo esto como una oportunidad en lugar de un obstáculo. Debemos enfocarnos en:

- Aprovechar los conocimientos previos, potenciándolos.
- Construir un contexto en el que los que saben pueden aprender más, evitando el síndrome del alumno aburrido porque conoce los temas que se explican.
- Crear un ambiente de colaboración entre los distintos perfiles de alumnos. Los alumnos con menos conocimientos obtienen ayuda de un par y los que poseen conocimientos previos los desarrollan ante la necesidad de transmitirlos (adquieren múltiples perspectivas diferentes del mismo concepto).

Desde la perspectiva didáctica: Trabajo en grupo

El trabajo en grupo debe concluir con la conformación de equipos de trabajo, que permitan a los miembros colaborar en un ambiente de compromiso y responsabilidad.

Compromiso y responsabilidad para con el resto de los miembros del equipo como también para con los módulos desarrollados. Las problemáticas de los ejercicios y trabajos planteados deben ser acordes.

Desde la perspectiva didáctica: Idioma

El idioma no debe representar una barrera para aprender de un problema o de su solución. Son pocos los alumnos que manejan con suficiente fluidez el inglés como para que no represente un factor complejidad externo e innecesario.

Las librerías de componentes y *frameworks* por tanto, deberían estar en español, en particular aquellas vinculadas con el contexto concreto del problema, que describen con sus abstracciones el modelo de la realidad analizada.

Antecedentes

Como mencionaba en los párrafos previos, participé de múltiples ensayos tendientes a resolver las dificultades observadas en el ámbito educativo y disminuir la brecha existente entre éste y el ámbito laboral. A continuación describiré aquellos que han tenido un gran impacto sobre el presente trabajo.

Los resultados de las experiencias serán explicados con posterioridad.

Primera versión

En los meses previos al inicio del primer cuatrimestre del 2006, junto al equipo de ayudantes de la materia Algoritmos y Programación III de la Facultad de Ingeniería de la UBA desarrollamos la primera versión del software que llamamos Batalla Espacial, con el objetivo de utilizarlo como herramienta para el desarrollo los Trabajos Prácticos.

La Batalla Espacial se concibió originalmente como un juego de tablero para programadores donde existen Civilizaciones con Naves que parten de una Base, pueden trasladar sustancias que encuentren en el espacio y tienen poder de ataque.

Se planteó como Trabajo Práctico llevar adelante el objetivo del juego que consiste en programar una Civilización, con su Comandante y Pilotos capaces de llevar hasta su Base la mayor cantidad posible de sustancias teniendo en cuenta que en la misma partida pueden existir otras Civilizaciones.

Uso y extensión

Para el siguiente cuatrimestre, se ampliaron, corrigieron y refinaron las reglas del juego, proponiendo el mismo enunciado como primer Trabajo Práctico y planteando una segunda parte que consistió en desarrollar un módulo que permita recuperar, editar y guardar la configuración del Tablero de la Batalla Espacial.

En esta oportunidad agregamos un nuevo desafió para los alumnos, no solo era necesario construir un módulo en el contexto de una aplicación existente, sino que también era necesario indagar, comprender y extender la plataforma, incorporando el editor de Tableros.

Ampliando la cobertura

A partir del primer cuatrimestre del 2008 en el equipo de docentes de la carrera de Ingeniería en Computación de Universidad Nacional de Tres de Febrero comenzamos a utilizar el contexto descripto por la Batalla Espacial para construir algunos ejemplos de problemas y resoluciones en las materias Estructura de Datos I, Lenguaje de Programación I y Estructura de Datos II.

En estas materias fueron planteados Trabajos Prácticos como pequeñas extensiones a la Batalla Espacial, utilizando las reglas del juego para describir un contexto más extenso que la problemática concreta del enunciado propuesto.

Ampliando el alcance

En el segundo cuatrimestre del 2009 desarrollé una extensión con el propósito de utilizar la Batalla Espacial en las clases iniciales de Estructura de Datos I (UNTref) para demostrar conceptos elementales como: objeto y contrato. En esta materia los alumnos tienen su primer acercamiento con los conceptos de la ingeniería de software.

Visión

La Batalla Espacial fue originalmente construida como el contexto de Trabajo Práctico para Algoritmos y Programación III, luego sufrió algunas pequeñas ampliaciones y ajustes para poder ser utilizada en otras materias.

Los docentes que actualmente la usamos notamos que no nos es suficiente. La necesidad pasa por tener un contexto más completo, que abarque los aspectos descriptos en las secciones anteriores. Como usuarios, sentimos que las extensiones realizadas no son suficientes y que futuras ampliaciones están limitadas por las características dieron origen al desarrollo.

Prueba de concepto

Considero que el desarrollo realizado hasta el momento representa una prueba de concepto, satisfactoria por cierto, para el presente trabajo.

Por tanto, creo que existe una oportunidad concreta de satisfacer las necesidades expuestas hasta el momento tomando como punto de partida la definición original de la Batalla Espacial, ampliando sus horizontes y haciendo una *refactorización* completa del código hasta aquí construido.

Alcance

Desarrollar los temas teóricos de la ingeniería de software que serán soportados por la plataforma. Estos son los objetivos temáticos de aprendizaje que se detallan en la próxima sección.

Construir la plataforma educativa junto con el material de soporte para las actividades propuestas de cada uno de los objetivos temáticos de aprendizaje.

Proveer en la plataforma herramientas que permitan el análisis experimental de la eficiencia algorítmica de los módulos internos y los desarrollados por los alumnos.

Hacer de este trabajo un proyecto *open source*, con el objetivo de que docentes y alumnos tengan a su disposición todo el material generado y puedan colaborar con extensiones, correcciones y mejoras.

Objetivos temáticos

Los objetivos temáticos de aprendizaje seleccionados son:

- Encapsulamiento
- Algoritmos de ordenamiento
- Pruebas
- Estructuras dinámicas de datos
- Herencia y Polimorfismo
- Grafos

Materiales

Los materiales didácticos a desarrollar para cada uno de los objetivos temáticos de aprendizaje seleccionados son:

- Desarrollo teórico
    - Descripción de la problemática en el contexto de la plataforma.
    - Ejemplo aplicación del concepto.
- Ejercicio práctico
    - Enunciado del ejercicio práctico.
    - Resolución tipo del ejercicio práctico.
    - Preguntas guía para elaborar el informe.

Requerimientos Funcionales

La plataforma educativa tendrá las siguientes funcionalidades:

- Gestión de módulos
  - Cargar dinámicamente módulos desarrollados por alumnos y docentes como solución a los ejercicios propuestos.
  - Cargar dinámicamente módulos desarrollados por alumnos y docentes que reemplacen módulos internos de la plataforma.
  - Ejecutar los módulos cargados dinámicamente.
- Eficiencia computacional
  - Activar y desactivar la recolección de datos durante la ejecución de los módulos para la construcción de métricas de eficiencia espacial y temporal.
  - Visualizar las datos recolectados de eficiencia espacial y temporal filtrando por:
    - módulos
    - ejecuciones
    - tamaños de problema
  - Exportar los datos para que puedan ser manipularlos mediante una planilla de cálculos.

Requerimientos no Funcionales

Hardware

Los requerimientos mínimos de hardware para la instalación y ejecución de la plataforma educativa son:

- procesador: 512 MHz
- memoria RAM: 128 MB
- espacio en disco: 10 MB

Software

El software de base requerido para instalar y ejecutar la plataforma educativa lo constituye una implementación de JRE (*Java Runtime Environment,* ***[JavaSE]***) versión 1.6. Existen múltiples productos disponibles para todas las arquitecturas y plataformas existentes, entre las que se destacan:

- OpenJDK (implementación open source) ***[JDK-Open]***

- Sun (Oracle) JRE (implementación propietaria y gratuita) ***[JDK-Sun]***

Para desarrollar módulos que se ejecuten sobre la plataforma educativa (desarrollo de ejercicios y prácticas) se requiere una implementación de JDK (*Java Development Kit,* ***[JavaSE]***) versión 1.6, existiendo también múltiples productos disponibles de manera gratuita para todas las arquitecturas y plataformas existentes, entre las que se destacan:

- Open JDK (implementación *open source*) ***[JDK-Open]***
- Sun (Oracle) JDK (implementación propietaria y gratuita) ***[JDK-Sun]***

Se recomienda además, aunque es opcional, la utilización de un ambiente integrado de desarrollo para Java (Java IDE, *Integrated Development Environment*) que mejore la experiencia de alumnos y docentes en la construcción de software. Cualquier producto de los que existen en el mercado (propietarios y *open source)* pueden ser utilizados de manera indistinta, quedando la selección concreta a consideración del usuario.

Portabilidad

Tanto la plataforma educativa como los módulos desarrollados por alumnos y docentes deben ser portables, siendo independiente su ejecución del sistema operativo y el *hardware*.

Extensibilidad

La plataforma debe definir un mecanismo simple y a la vez potente para agregar nuevos objetivos temáticos de aprendizaje.

La implementación debe cumplir con el principio de abierto-cerrado. Debe estar abierta a su extensión pero cerrada para su uso.

Arquitectura

La Plataforma Educativa Batalla Espacial 2.0 se construirá como una aplicación de escritorio (*standalone*) utilizando la plataforma de desarrollo JavaSE 6 (Java Standard Edition 6 ***[JavaSE]***).

Será estructurada según el patrón de arquitectura MVC (*Model View Controller* ***[Burbeck87]***) y haciendo fuerte hincapié en la modularidad de su elementos componentes.

La interfaz gráfica de usuario (G*UI, graphical user interface*) será construida utilizando *Swing* ***[Swing]***.

Utilizará una base de datos embebida (Apache Derby ***[Derby]***) para almacenar los datos de eficiencia computacional registrados durante la ejecución de los módulos.

Herramientas de Desarrollo

Hardware

Para llevar adelante el desarrollo del presente trabajo profesional se utilizará el siguiente hardware:

- procesador: AMD Athlon 64 x2 Dual Core 5200+
- memoria: 6 GiB
- espacio en disco: 1 GiB

Software

Software de base

- Sistema operativo: Ubuntu 10.04 ***[Ubuntu]***
- JDK: Sun (Oracle) JDK 1.6.22 ***[JDK-Sun]***

Servicios

- Google groups ***[GoogleGroups]***
  Servicio gratuito de listas de correo:
  http://groups.google.com/group/tp-batalla-espacial
  http://groups.google.com/group/batalla-espacial
- Google code ***[GoogleCode]***
  Servicio gratuito de hosting y administración de proyectos de desarrollo de software:
  http://code.google.com/batalla-espacial

Herramientas

- Eclipse Helios (3.6) ***[Eclipse]***
  IDE. Open source.
- Apache Maven ***[Maven]***
  Herramienta de build. Open Source.
- Visual Paradigm for UML ***[VP]***
  Editor de UML. Community Edition (para usos educativos)

Frameworks

- pruebas unitarias: jUnit ***[JUnit]***
- persistencia: JPA (Java Persistence API) ***[JPA]***, implementación Hibernate ***[Hibernate]***
- interfaz de usuario: Swing ***[Swing]***
- base de datos embebida: Apache Derby ***[Derby]***
- grafos: JUNG (Java Universal Network and Graph API) ***[JUNG]***

Planificación

Duración: 4 meses.

Iteraciones de 2 semanas.

Entregas

Plan de liberaciones (entregas) cada 2 iteraciones.

Entrega 1

- Refactorización del código legado.
- Identificación de las oportunidades en el código (puntos de extensión).
- Selección de las funcionalidades a desarrollar a partir de los objetivos temáticos de aprendizaje y las oportunidades identificadas en el código. (Construir un listado de requerimientos priorizado)
- Identificación de las funcionalidades primarias y secundarias.
- Prototipo de la API de medición experimental de eficiencia algorítmica.
- Prototipo de la interfaz de usuario (UI) de visualización y análisis de las mediciones registradas de eficiencia algorítmica.

Entrega 2

- Desarrollo de las funcionalidad primarias.
- API de medición experimental de complejidad algorítmica.
- Interfaz de usuario (UI) de visualización y análisis de las mediciones registradas de eficiencia algorítmica.

Entrega 3

- Desarrollo de las funcionalidad secundarias.

Entrega 4

- Producto final.
- Presentación.
- Documentación.

Funcionalidades

Cada una de las funcionalidades está compuesta por:

- Soporte en la plataforma
- Documentación
- Material y enunciado de actividad tipo
- Resolución de la actividad tipo
- Material de actividad tipo para el docente
- Resolución de la actividad tipo para el docente

Referencias

Bibliografía

[Burbeck87]

Steve Burbeck. Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC).

http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

accedido el 7/03/2011

[Gamma95]

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. ***Design Patterns. Elements of Reusable Object-Oriented Software.*** Addisson Wesley, 1995.

[Meyer09]

Bertrand Meyer. ***Touch of Class. Learning to Program Well with Objects and Contracts.*** Springer, 2009.

[Meyer85]

Bertrand Meyer. ***Object-Oriented Software Construction.*** Prentice Hall, 1985, 2da. Edición 1997.

Vínculos

[Derby]

Apache Derby.

http://db.apache.org/derby/

accedido el 18/03/2011

[Eclipse]

Eclipse. IDE open source.

http://www.eclipse.org/

accedido el 20/03/2011

[GoogleCode]

Google Code. Servicio gratuito de hosting y administración de proyectos de desarrollo de software.

http://code.google.com/

accedido el 21/03/2011

[GoogleGroups]

Google Groups. Servicio gratuito de listas de correo.

http://groups.google.com/

accedido el 21/03/2011

[Hibernate]

JBoss Hibernate. Persistence framework, JPA implementation [JPA].

http://www.hibernate.org/

accedido el 05/12/2010

[JavaSE]

Java Platform, Standard Edition.

http://www.oracle.com/technetwork/java/javase/overview/index.html

http://download.oracle.com/javase/6/docs/

accedidos el 20/03/2011

[JDK-Sun]

Java Development Kit. Implementación propietaria y gratuita de JDK de Sun (Oracle).

http://www.oracle.com/technetwork/java/javase/downloads/index.html

accedido el 20/03/2011

[JDK-Open]

Open JDK. Implementación open source de JRE y JDK.

http://openjdk.java.net/

accedido el 20/03/2011

[JPA]

Java Persistence API. Framework de persistencia.

http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html

accedido el 05/12/2010

[JRE-Sun]

Java Runtime Environment. Implementación propietaria y gratuita de JRE de Sun (Oracle).

http://www.java.com

accedido el 20/03/2011

[JUNG]

Java Universal Network/Graph Framework.

http://jung.sourceforge.net/

accedido el 21/12/2011

[JUnit]

JUnit. Testing framework.

http://www.junit.org/

accedido el 15/03/2011

[Maven]

Apache Maven.

http://maven.apache.org/

accedido el 20/03/2011

[Swing]

Graphical User Interface (GUI) framework.

http://download.oracle.com/javase/6/docs/technotes/guides/swing/index.html

accedido el 17/12/2010

[Ubuntu10]

Ubuntu 10.04. Sistema operativo open source.

http://www.ubuntu.com/

accedido el 20/03/2011

[VP]

Visual Paradigm for UML.

http://www.visual-paradigm.com/

accedido el 20/03/2011