TRABAJO FIN DE GRADO

GRADO EN INGENIERIA INFORMATICA

GenoMus: rediseño y desarrollo

Implementación de un motor de cómputo funcional basado en prototipos sobre estructuras musicales.

Autor
Miguel Pedregosa Pérez

Director
Miguel Molina Solana, José López Montes

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Junio de 2022

GenoMus: rediseño y desarrollo
Implementación de un motor de cómputo funcional basado en prototipos sobre estructuras musicales.

Miguel Pedregosa Pérez

Palabras clave: *software libre*

Resumen

Same, but in English

Student's name

Keywords: *open source*, *floss*

Abstract

D. **Tutora/e(s)**, Profesor(a) del ...

Informo:

Que el presente trabajo, titulado ***GenoMus: rediseño y desarrollo: Implementación de un motor de cómputo funcional basado en prototipos sobre estructuras musicales.***, ha sido realizado bajo mi supervisión por **Miguel Pedregosa Pérez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a Junio de 2018.

El/la director(a)/es:

(Miguel Molina Solana)

Agradecimientos

Índice general

1. Introducción **15**

2. Descripción del problema **17**

3. Estado del arte **19**

4. Planificación **21**
4.1. Metodología utilizada 21
4.2. Temporización 22
4.3. Seguimiento del desarrollo 22
4.3.1. Seguimiento de funcionalidades 22
4.3.2. Control de versiones 22

5. Análisis del problema **23**
5.1. Análisis del prototipo 23
5.1.1. Ventajas del planteamiento funcional del código en JS. 23
5.1.2. Placebos del JS funcional. 23

6. Implementación **25**
6.1. genomus-core 25
6.1.1. Modelo de datos 25
6.1.2. Modelo de cómputo 25
6.2. genomus-core-js 27

7. Conclusiones y trabajos futuros **29**

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este proyecto consiste en el establecimiento de GenoMus [4] como un proyecto de software libre profesional. Todo el código que conforma la implementación del proyecto está liberado bajo la licencia GPLv3.0 [3].

... presentación de GenoMus.

…

Características que añade esta implementación:

- Control de versiones según el estándar de facto de la industria. - Mejoras en DX: código "hackable Mejoras en eficiencia en las áreas de computación intensiva de GenoMus [4]

Miguel Pedregosa Pérez

Capítulo 2

Descripción del problema

El diseño de la implementación realizada busca dar la talla en los dos siguientes rasgos principales:

- El código debe poder ser entendido y mantenido por un programador amateur. Al pertenecer al campo de las humanidades digitales tenemos que tener en cuenta a los usuarios con perfiles no técnicos. Por esto la legibilidad, simplicidad y declaratividad del código son prioritarias. - El programa debe ser lo más eficiente posible en regiones de cómputo intensivo. Aspiramos a que el límite en el volumen o complejidad de datos que el programa genera no repercuta sobre la capacidad de creación del usuario ni sobre la usabilidad del software.

Capítulo 3

Estado del arte

El software libre y sus licencias [3] ha permitido llevar a cabo una expansión del aprendizaje de la informática sin precedentes.

Capítulo 4

Planificación

4.1. Metodología utilizada

La metodología de diseño y desarrollo llevada a cabo está inspirada fuertemente por Scrum, con algunas modificaciones dadas las condiciones del proyecto. Además, se ha llevado acabo un proceso de TDD. Las condiciones del proyecto que han condicionado la propuesta de proyecto son las siguientes:

- **El perfil de usuario final.** El software a desarrollar no es un producto de usuario en el sentido tradicional. Es una serie de herramientas de desarrollo que serán utilizadas por otros programadores. Así, nuestros usuarios son los desarrolladores que usarán el software.
- **La complejidad del MVP.** El modelo de datos y cómputo de GenoMus [4] está altamente entrelazado. Esto aumenta la complejidad de la creación de cortes verticales (cita requerida) de grupos funcionales que podrían corresponder a hitos (cita requerida).
- **La inestabilidad de la capacidad del equipo de desarrollo.** El equipo de desarrollo no se podía alejar más de lo defendido como ideal por Scrum (cita requerida). El desarrollo se ha llevado a cabo únicamente por el autor a modo de side-project (cita requerida).

Ante estas condiciones, se ha llevado a cabo un proceso de desarrollo con las siguientes características:

- **División de las funcionalidades en historias agrupadas en hitos.** Las funcionalidades técnicas del producto se han dividido según el método usual en metodologías ágiles.
- **División temporal del trabajo en sprints.** El tiempo de desarrollo se ha dividido en sprints como es usual. Los sprints del proyecto han contado con la

peculiaridad de ser de tiempo variable debido a la capacidad variable de desarrollo. Los sprints han durado el tiempo necesario para completar los objetivos asignados. Finalmente, los sprints han durado una media de tres semanas.

- **TDD como testeo de usuario hasta el despliegue del MVP.** Los flujos de usuario del prototipo trabajan con las estructuras de más alto nivel de GenoMus. Al ser estas estructuras dependientes del modelo de cómputo subyacente, se ha propuesto este modelo de cómputo junto con alguna funcionalidad básica como MVP. Al no respaldar ningún flujo de usuario, el desarrollo del MVP se ha realizado utilizando TDD como verificación de correctitud.
- **Planificación dinámica de historias e hitos.** El proyecto cuenta con varios ámbitos de incertidumbre los cuales han imposibilizado el análisis de requisitos funcionales completo previo al comienzo del desarrollo. Incluso la viabilidad del producto en el tiempo propuesto era desconocida. Así, el análisis completo de requisitos se ha llevado a cabo durante el desarrollo del MVP.

4.2. Temporización

El proyecto ha sido planteado para su desarrollo entre marzo(?) y julio de 2022. Es decir cuatro meses de desarrollo.

4.3. Seguimiento del desarrollo

4.3.1. Seguimiento de funcionalidades

Es proyecto ha sido instanciado como proyecto en Jira (cita requerida). Esto ha permitido disponer de un tablero estilo KanBan (cita requerida), un Backlog de producto (cita requerida) y de una herramienta de planificación de sprints.

(imagen del kanban board)

4.3.2. Control de versiones

La totalidad del código cuenta con control de versiones a través de git y está alojado en repositorios en GitHub. Se ha seguido el siguiente flujo de git para posibilitar el desarrollo paralelo de funcionalidades y el despliegue de diferentes versiones del producto.

explicar git flow

- ramas
- squash and merge
- version tags

Capítulo 5

Análisis del problema

5.1. Análisis del prototipo

Como se ha mencionado previamente en el apartado de descripción del problema, el objetivo principal del proyecto es dotar a GenoMus de una implementación de carácter profesional a prueba de futuro que sea mantenible por perfiles no técnicos.

El prototipo de GenoMus está implementado en javascript puro para ejecución sobre NodeJS. La sintaxis utilizada busca la utilización del "dialecto"funcional de javascript. Tanto el modelo de representación de datos como el modelo de cómputo siguen generalmente patrones de diseño de programación funcional.

5.1.1. Ventajas del planteamiento funcional del código en JS.

Es bonito

5.1.2. Placebos del JS funcional.

Javascript nos proporciona azúcar sintáctico sobre la creación de objetos dinámicos que nos da la posiblidad de implementar entidades como clausuras o funciones de orden superior. Sin embargo estas construcciones que tan bien nos entran por los ojos no están acompañadas de un respaldo acorde en el entorno de ejecución de js. Ninguno de los principales motores de js incluye algún tipo de análisis de los objetos funcionales para su optimización en grupo, como procedimientos de composición de funciones o un motor de reducción de grafos de dependencias computacionales.

La composición de funciones es la característica cuya ausencia puede haber condicionado el desarrollo del prototipo. El prototipo implementa un procedimiento de almacenamiento de árboles funcionales a través de cadenas de texto. Las diferentes construcciones funcionales se almacenan como código javascript, de cuya ejecución se obtiene la evaluación del árbol. Este tipo de metaprogramación aumenta el

cómputo necesario para evaluar árboles de funciones, ya que se introduce una etapa de construcción de código y una etapa de parseo de este, validación y posterior ejecución.

Capítulo 6

Implementación

Tras analizar el prototipo se propone la implementación de las funcionalidades y críticas de GenoMus en C++. Este subconjunto de las funcionalidades se desarrollará bajo el nombre de genomus-core. Se propone reutilizar las funcionalidades secundarias del prototipo de javascript. La totalidad del paquete se desarrollará bajo el nombre de genomus-js.

Así, la base de código se dividirá en tres áreas: (???)

1. Una librería de C++ que implementa el modelo de datos y de cómputo de GenoMus: genomus-core [1].

2. Un módulo nativo de NodeJs (cita) que exponga una API de interación entre el runtime de JS y la biblioteca de C++: genomus-core-js [2].

3. Un módulo de npm que encapsule lo previamente enunciado y reutilize la lógica de I/O del prototipo: genomus-js.

6.1. genomus-core

genomus-core implementa el modelo de datos y de cómputo de GenoMus.

6.1.1. Modelo de datos

Fácil

6.1.2. Modelo de cómputo

El modelo de cómputo de GenoMus está intrínsecamente basado en el paradigma de programación funcional. Esto hace que una implementación en C++ no sea trivial por diferentes motivos:

- **Funciones dinámicas en tiempo de ejecución.** Las funciones de cómputo disponibles deben ser declarables en tiempo de ejecución. Ya que las funciones disponibles al modelo de cómputo son dependientes del estado del programa, es ideal que estas se puedan declarar dinámicamente en tiempo de ejecución. En la actual implementación, se proporciona una serie de funciones instanciadas en tiempo de carga.

- **Polimorfismo funcional.** Necesitamos construcciones que doten al modelo de cómputo de polimorfismo declarado en tiempo de ejecución. Debe ser posible no solo declarar nuevas funciones en tiempo de ejecución, sino también declarar nuevos tipos a usar como parámetros o salidas. Esto nos quita de la ecuación casi todos los constructos que C++ nos ofrece para conseguir polimorfismo dinámico, ya que el dominio del polimorfismo debe ser conocido en tiempo de compilación en C++. (cita requerida)

- **Árboles funcionales declarables y evaluables.** Toda instancia de cómputo debe no solo ser computable sino también almacenable. La instancia de árbol funcional debe existir independientemente de su evaluación. Así, una función de cómputo será un objeto invocable o functor cuya invocación contribuirá a la declaración del árbol funcional en memoria.

  (... ejemplo de código, diagrama de memoria(?))

  Esto, además de ser necesario para el funcionamiento del software, posibilita el diseño de algoritmos de cómputo sobre el árbol funcional más allá del típico recorrido por profundidad. (cita requerida - por ejemplo haskell graph reduction machine)

- **Funciones enumerables en tiempo de ejecución.** Las funciones de cómputo disponibles en tiempo de ejecución deben ser enumerables y referenciables según los diferentes métodos de acceso(cita a donde se explican las cosas codificadas/decodificadas).

Todas estas características son bien incompatibles con la metodología de desarrollo orientado a objetos estándar de C++ o bien incompatibles con la máxima del proyecto de ser asequible para programadores amateur, si pretendemos seguir las buenas prácticas de C++. Por este motivo se toma la decisión de alejarse de la práctica principal de polimorfismo funcional y de datos definida en los últimos estándares de C++ (C++11, C++17(cita requerida)) para realizar una implementación mixta entre orientada a objetos clásica y orientada a prototipos(cita requerida).

Lo que nos aporta conceptualmente el diseño orientado a prototipos es la posibilidad de manejar parcialmente el tipado de nuestros objetos en tiempo de ejecución. Por otra parte, la principal desventaja que nos trae es la existencia de errores de tipado no conocidos por el compilador, lo cual es natural cuando lo que buscamos es precisamente declarar funciones de tipado arbitrario en tiempo de ejecución.

Así, se nos presenta un compromiso entre la seguridad del código y la legibilidad de este. La implementación propuesta busca maximizar la legibilidad y flexibilidad del código minimizando la cantidad de comprobaciones de tipado en tiempo de ejecución. Además, se plantea mitigar el riesgo del tipado dinámico mediante la creación de tests de integración.

6.2. genomus-core-js

Miguel Pedregosa Pérez

Capítulo 7

Conclusiones y trabajos futuros

Bibliografía

[1] Miguel Pedregosa Pérez. genomus-core. https://github.com/mipdr/genomus-core.

[2] Miguel Pedregosa Pérez. genomus-core-js. https://github.com/mipdr/genomus-core-js.

[3] Free Software Foundation. GNU General Public License. http://www.gnu.org/licenses/gpl.html.

[4] José López Montes. Genomus. https://github.com/lopezmontes/GenoMus.