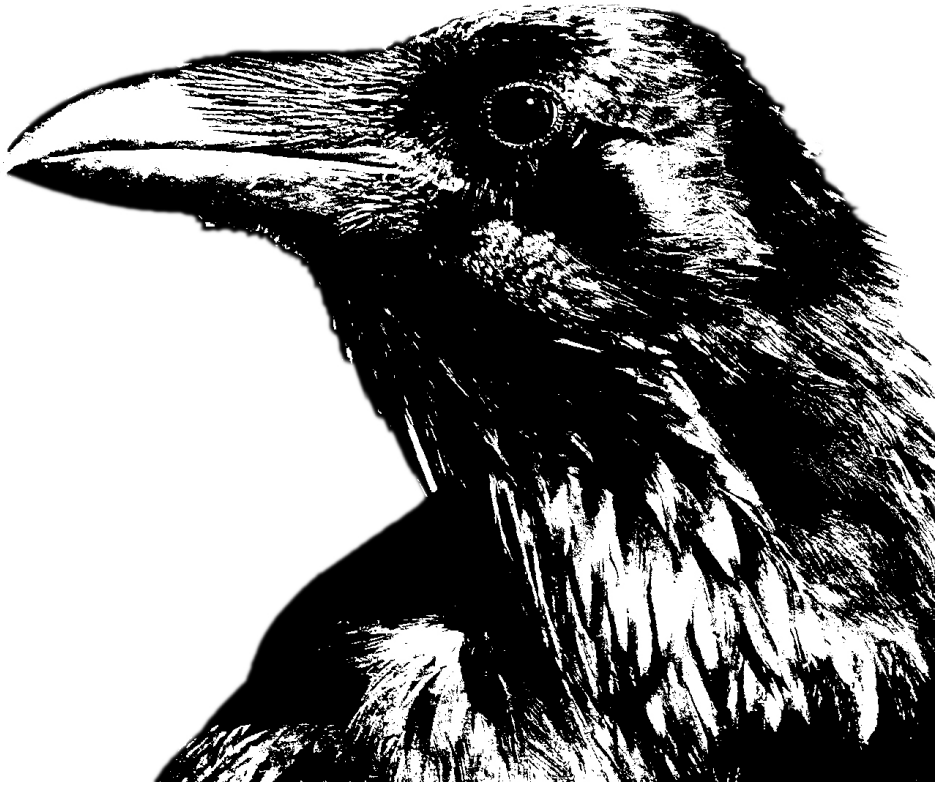
EL TIEMPO PASA Y EL SOFTWARE MUERE

Diseñando programas resistentes al cambio

Inspirado por las clases de Alexandre Bergel

Ignacio Slater Muñoz



Departamento de Ciencias de la Computación
Universidad de Chile

Índice general

La parte del libro que nadie lee

La idea de este «apunte» nació como una *wiki* de *Github* creada por Juan-Pablo Silva como apoyo para el curso de *Metodologías de Diseño y Programación* dictado por el profesor Alexandre Bergel del Departamento de Ciencias de la Computación, Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

Lo que comenzó como unas notas para complementar las clases del profesor lentamente fue creciendo, motivado por esxs alumnxs que buscaban dónde encontrar soluciones para esas pequeñas dudas que no les dejaban avanzar.

El objetivo principal del texto sigue siendo el mismo, plantear explicaciones más detalladas, ejemplos alternativos a los vistos en clases y para dejar un documento al que lxs alumnxs puedan recurrir en cualquier momento.

Este libro no busca ser un reemplazo para las clases del curso, es y será siempre un complemento.

Esta obra va dirigida a los estudiantes de la facultad así como para cualquier persona que esté dando sus primeros pasos en programación. El libro presenta una introducción al diseño de software, la programación orientada a objetos y lo básico del lenguaje de programación *Kotlin*. Se asume que los lectores tienen nociones básicas de programación, conocimiento básico de *Python* y, en menor medida, de *C*.

Antes de comenzar, debo agradecer a las personas que hicieron posible y motivaron la escritura de esto: Beatríz Graboloza, Dimitri Svandich, Nancy Hitschfeld y, por supuesto, Alexandre Bergel y Juan-Pablo Silva.

16 de agosto de 2022, Santiago, Chile

Sobre esta versión

Este libro no partía así, y si alguien leyó una versión anterior se dará cuenta. Solía comenzar con una descripción e instrucciones para instalar las herramientas necesarias para seguir este libro y eso no estaba mal, pero no me agradaba comenzar así, simplemente presentando las herramientas sin ningún contexto de por qué ni para qué las ibamos a utilizar.

Puede parecer ridículo cambiar todo lo que ya había escrito solamente por eso, pero cuando nos enfrentamos a problemas del mundo real esto comienza a cobrar más sentido. Reescribo estos capítulos por una razón simple pero sumamente importante y que será una de las principales motivaciones para las decisiones de diseño que tomaremos a medida que avancemos, en el desarrollo de software **lo único constante es el *cambio***.[1] Una aplicación que no puede adaptarse a los cambios, sin importar que tan bien funcione, está destinada a morir.

¿Qué sucede entonces con las herramientas que vamos a utilizar? Las vamos a introducir, no podemos sacar una parte tan importante, pero no las vamos a presentar todas a la vez, en su lugar las iremos explicando a medida que las vayamos necesitando.

[1] **head-first-intro**.

Parte I

Por algo se empieza

Capítulo 1

¡Kotlin!

1.1. ¿Java?

Java es uno de los lenguajes de programación más utilizados en el mundo (de ahí la necesidad de enseñarles éste y no otro lenguaje), se caracteriza por ser un lenguaje basado en clases, orientado a objetos, estática y fuertemente tipado, y (casi totalmente) independiente del sistema operativo.

¿Qué?

Tranquilos, vamos a ir de a poco. Comencemos por uno de los puntos que hizo que *Java* fuera adoptado tan ampliamente en la industria, la independencia del sistema operativo. Cuando *Sun Microsystems*[1] publicó la primera versión de *Java* (en 1996), los lenguajes de programación predominantes eran *C* y *C++* (y en menor medida *Visual Basic* y *Perl*). Estos lenguajes tenían en común que interactuaban directamente con la API del sistema operativo, lo que implicaba que un programa escrito para un sistema *Windows* no funcionaría de la misma manera en un sistema *UNIX*. *Java* por su parte planteó una alternativa distinta, delegando la tarea de compilar y ejecutar los programas a una máquina virtual (más adelante veremos en más detalle parte del funcionamiento de la *JVM* para entender sus beneficios y desventajas). Esto último hizo que, en vez de cambiar el código del programa para crear una aplicación para uno u otro sistema operativo, lo que cambiaba era la versión de la *JVM* permitiendo así que un mismo código funcionara de la misma forma en cualquier plataforma capaz de correr la máquina virtual.[2] Pasarían varios años antes de que surgieran otros lenguajes que compartieran esa característica (destacando entre ellos *Python 2*, publicado el año 2000).

[1] Actualmente *Java* es propiedad de *Oracle Corporation*

[2] Actualmente casi todos los sistemas operativos son capaces de usar la *JVM*, en particular el sistema *Android* está implementado casi en su totalidad para usar esta máquina virtual.

1.1.1. ¿Debería instalar *Java*?

> «Haz la w** que querai, pero ponte chaleco»
>
> (Anita Tijoux)

Esto es algo completamente opcional, no es necesario para este libro, pero es uno de los lenguajes más utilizados en el mundo y los contenidos de este libro son directamente aplicables a este lenguaje.

Si por alguna razón quieren instalar *Java*, puedes encontrar una guía de instalación al final de este capítulo.

1.2. ¿Kotlin?

Kotlin es un lenguaje de programación *multiplataforma*, estática y fuertemente tipado y con una inferencia de tipos más avanzada que la de *Java*.

¿Pero para qué aprender Kotlin si puedo aprender Java?

Buena pregunta.

Kotlin es un lenguaje desarrollado por *JetBrains* pensado para ser totalmente interoperable con *Java*, esto quiere decir que puedo importar código escrito en *Java* desde *Kotlin* y vice versa. En fin, existen muchísimas razones para aprender *Kotlin*, pero en vez de enumerarlas creo que es mejor que las vayan descubriendo en el transcurso de este libro.

1.3. Tipos en *Kotlin*

Al igual que *Java*, *Kotlin* es un lenguaje de tipado estático y fuerte, pero con una inferencia de tipos mucho más poderosa que la de *Java*. El sistema de tipos de *Kotlin* es un tema sumamente complejo que se escapa del alcance de este libro, así que nos limitaremos a dar una pequeña introducción a éste y lo iremos desarrollando un poco más a lo largo del libro a medida que vaya siendo necesario.

Una variable en *Kotlin* se define con la sintaxis `<val|var><id>: <type>= <value>`, entonces, si queremos definir un entero haremos:

```
val i: Int = 8000
```

Capítulo 2

IntelliJ, tu nuevo *bff*

Hasta ahora es probable que tengan experiencia utilizando algún editor de texto como *Notepad++*, *Sublime* o *Visual Studio Code*, eso es bueno, pero no suficiente para enfrentar problemas complejos (esto no es por desmerecer a esos editores de texto, también son herramientas poderosas apropiadas para otros contextos). Es aquí cuando surge la necesidad de tener un *entorno de desarrollo integrado* (IDE), un IDE es como un editor de texto cualquiera, pero poderoso. *IntelliJ* es un IDE desarrollado por *JetBrains* especializado para desarrollar programas en *Java* y *Kotlin*, y es la herramienta que usaremos de aquí en adelante.

Pero VSCode nunca me ha fallado ¿Por qué no puedo usarlo también para esto si le puedo instalar extensiones para programar en Java?

La razón es simple, pero muy importante, como dije un poco más arriba, *IntelliJ* es una herramienta diseñada específicamente para desarrollar en *Java* y *Kotlin*, esto hace que sea muchísimo más completo que un editor de texto cualquiera y que tenga facilidades para correr, configurar y testear[1] nuestros proyectos. De nuevo, no me malinterpreten, *VSCode* es una herramienta sumamente completa y perfectamente capaz de utilizarse en el mundo profesional, pero es bueno que conozcan otras alternativas también. Es posible utilizar cualquier otra herramienta que permita hacer lo mismo que *IntelliJ* para seguir este libro, pero será responsabilidad del lector o lectora adaptar lo visto aquí para funcionar con dichas herramientas.[2]

[1] Esto será sumamente importante en capítulos futuros

[2] Dicho esto, este libro es abierto a la colaboración y cualquier aporte que contribuya al aprendizaje de lxs lectorxs es bienvenido. En caso de querer colaborar pueden hacerlo mediante un *Pull Request* al repositorio de *GitHub* de este libro (`https://github.com/islaterm/software-design-book-es`), respetando las normas establecidas en el *Código de conducta* .

2.1. No termina de convencerme, pero le voy a dar una oportunidad. ¿Cómo lo instalo?

¡Esa es la actitud!

Si estas leyendo este libro y eres estudiante, tengo excelentes noticias para ti. Si no eres estudiante, tengo no tan excelentes noticias para ti.

IntelliJ viene en dos sabores, sabor *Community* y sabor *Ultimate*. Si eres un ser humano común y corriente (o un gato con acceso a internet) puedes descargar e instalar *IntelliJ IDEA Community Edition* desde el sitio oficial de *JetBrains*. ¿Muy complicado? Totalmente de acuerdo. ¿Por qué descargar el *IDE* desde la página web cuando *JetBrains* nos entrega una herramienta para facilitarnos el proceso?

2.1.1. JetBrains Toolbox

JetBrains Toolbox es una herramienta que facilita la instalación de los productos de *JetBrains*, manejar distintos proyectos y también hace más fácil actualizar las herramientas.

Capítulo 3

Git ¿Amigo o enemigo?

Una de las herramientas más importantes que deben conocer es *Git*, o algún otro *sistema de versionado*. Un *sistema de versionado* (VCS) es un programa para mantener un historial de todos los cambios realizados sobre un conjunto de archivos. Existen variados VCS como *Azure DevOps*, *CVS* o *Mercurial*, pero el más utilizado es *Git*.

Git fue creado por Linus Torvalds el aμo 2005 para poder desarrollar el núcleo de *Linux* junto a otros colaboradores. Esto deja un precedente de que *Git* fue ideado con la idea de poder trabajar de forma colaborativa en una aplicación, pero también es extremadamente útil para trabajar en un proyecto de forma individual.

Los beneficios de utilizar *Git*, y algunos otros detalles sobre cómo funciona, los iremos viendo a medida que ponemos en práctica esta herramienta.

3.1. Instalando *Git*

3.1.1. *Windows*

***Chocolatey* (Recomendado)**

Nuevamente, el método recomendado será utilizar *Chocolatey* para instalar *Git* en *Windows*.

El proceso de instalación es simple, solamente deben ejecutar *Powershell* como administrador y escribir:

```
choco install git
# O de forma equivalente:
# cinst git
```

Git for Windows

Git for Windows es un conjunto de herramientas que incluye *Git BASH* (una interfaz de consola que emula la terminal de un sistema *UNIX* que viene con *Git* instalado), *Git GUI* (una interfaz gráfica para manejar *Git*) e integración con *Windows Explorer* (esto significa que pueden hacer *clic* derecho en una carpeta y abrirla desde *Git BASH* o *Git GUI*).

Para instalarla deben descargar el cliente desde el sitio oficial de *Git for Windows* y seguir las instrucciones del instalador.

3.1.2. Linux

La forma más fácil de instalar *Git* es utilizando el gestor de paquetes del sistema operativo que estén usando, el problema de esto es que dependiendo de su distribución de *Linux* las instrucciones de instalación serán distintas, por esto se mostrará como instalar en arquitecturas basadas en *Debian* (como *Ubuntu*), para instalar en otro SO deberán revisar las instrucciones de la documentación oficiaL.

Para instalar deben abrir una consola y ejecutar:

```
sudo apt install git-all
```

3.1.3. macOS

Git mediante Xcode

La forma más simple de instalar *Git* en sistemas *macOS*

Índice