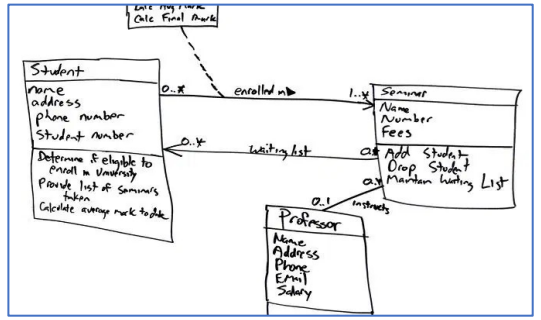
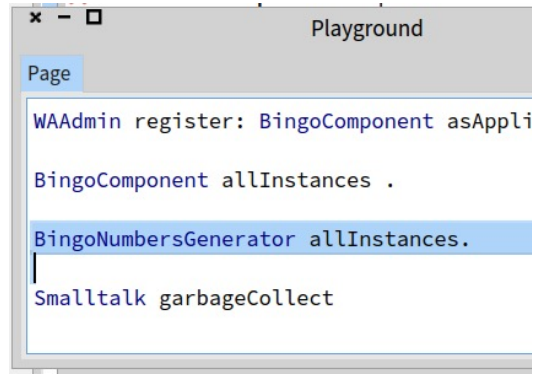
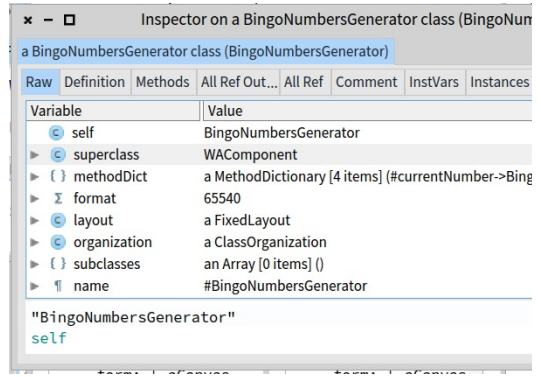
Separando código de presentación y de dominio

El código de dominio

- Hasta ahora hicimos aplicaciones OO sin interfaz
- Teníamos modelos de objetos que implementaban los requerimientos
  - Les pedíamos cosas desde el playground
  - Los veíamos con los inspectores (inspect-it y print-it)
- Escribiamos tests Sunit para asegurar que nuestra aplicación cumplía con los requerimientos

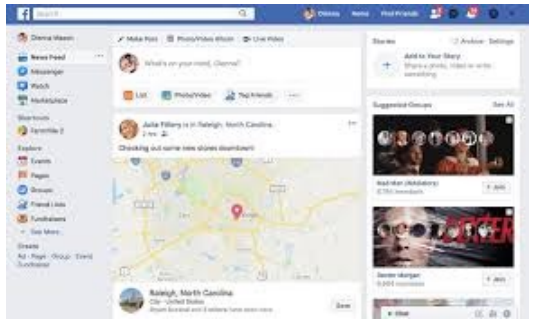






El código de presentación

- Al programar la “capa” de presentación, vemos que se trata de un animal bien diferente
- Se requieren tecnologías y herramientas específicas
  - A veces distintos lenguajes
  - A veces varias plataformas, varias interfaces
- Se habla de métaforas, estética, usabilidad, experiencia de usuario
  - Lo que implica incolucrar a expertos en esos dominios



Código de presentación vs. Código de dominio

- Cualquier código que hace algo con la interfaz de usuario, solo debe estar relacionado a la interfaz de usuario. Le llamamos “código de presentación”
  - Si hace algo con la interfaz, no debe manipular los datos (salvo para formatearlos), ni hacer cálculos.
- Si hace cálculos, transforma datos, hace validaciones, interactúa con otros sistemas, le llamamos “código de dominio”
  - Si es código de dominio, no debe tener ninguna referencia al “código de presentación”

¿Por que separarlos?

- El estilo con el que programamos el código de presentación, y la complejidad de este difiere del estilo y complejidad del código de dominio.
  - Mantenerlos separados permite que nos concentremos en una sola cosa a la vez
  - El código de presentación suele utilizar librerías que solo sirven para presentación.
  - Los tiempos con los que cambia uno y otro son diferentes
- Nos permite tener múltiples presentaciones para un mismo código de dominio
  - El código de dominio es mas fácil de portar

Separating User Interface Code

Martin Fowler

The first program I wrote on a salary was scientific calculation software in Fortran. As I was writing, I noticed that the code running the primitive menu system differed in style from the code carrying out the calculations. So I separated the routines for these tasks, which paid off when I was asked to create higher-level tasks that did several of the individual menu steps. I could just write a routine that called the calculation routines directly without involving the menus.

Thus, I learned for myself a design principle that's served me well in software development: Keep your user interface code separate from everything else. It's a simple rule, embodied into more than one application framework, but it's often not followed, which causes quite a bit of trouble.

user interface code as *presentation code* and the other code as *domain code*.

When separating the presentation from the domain, make sure that no part of the domain code makes any reference to the presentation code. So, if you write an application with a WIMP (windows, icons, mouse, and pointer) GUI, you should be able to write a command line interface that does everything that you can do through the WIMP interface—without copying any code from the WIMP into the command line.

Why do this?

Following this principle leads to several good results. First, this presentation code separates the code into different areas of complexity. Any successful presentation requires a fair bit of programming, and the complexity inherent in that presentation differs in style from the domain with which

M. Fowler, "Separating user interface code," in *IEEE Software*, vol. 18, no. 2, pp. 96-97, March-April 2001.