MVC / MVP

Reid Holmes

Background

‣ MVC started w/ Smalltalk-80

‣ Java UI frameworks & EJBs reignited interest

‣ Also prevalent in GWT and .NET development

MVC Motivation

‣ UI changes more frequently than business logic

  ‣ e.g., layout changes (esp. in web applications)

‣ The same data is often displayed in different ways

  ‣ e.g., table view vs chart view

  ‣ The same business logic can drive both

‣ Designers and developers are different people

‣ Testing UI code is difficult and expensive

‣ Main Goal: Decouple models and views

  ‣ Increase maintainability/testability of system

  ‣ Permit new views to be developed

Model

- Contains application data

  - This is often persisted to a backing store

- Does not know how to present itself

- Is domain independent

- Are often Subjects in the Observer pattern

View

‣ Presents the model to the user

‣ Allows the user to manipulate the data

‣ Does not store data

‣ Is configurable to display different data

Controller

▸ Glues Model and View together

▸ Updates the view when the Model changes

▸ Updates the model when the user manipulates the view

▸ Houses the application logic

▸ Loose coupling between Model and others

▸ View tightly cohesive with its Controller

Abstract topology

Concrete topology

Factory f = GWT.create(Factory.class);
ViewController c = new ViewController();
View v = f.createView(c);

[gwt.xml maps Factory.class
 to the right type]

Interaction mechanism

‣ User interacts with the UI (View)

‣ UI (View) notifies controller of changes

‣ Controller handles notifications, processing them into actions that can be performed on the model

‣ Controller modifies the model as required

‣ If the model changes, it fires modification events

‣ The view responds to the modification events

Benefits and tradeoffs

‣ Pro:

    ‣ Decouple view from model

        ‣ Support multiple views [collaborative views]

        ‣ Maintainability [add new views]

        ‣ Split teams [relieve critical path]

    ‣ Testability [reduce UI testing]

‣ Con:

    ‣ Complexity [indirection, events]

    ‣ Efficiency [frequent updates, large models]

MVP Motivation

‣ Take MVC a tiny bit further:

  ‣ Enhance testability

  ‣ Further separate Designers from Developers

‣ Leveraged by both GWT and .NET

Model

‣ Contains application data

  ‣ This is often persisted to a backing store

‣ Does not know how to present itself

‣ Is domain independent

‣ Often fires events to an Event Bus

View

‣ Thin UI front-end for controller

‣ Does not store data

‣ Can be interchanged easily

‣ Does not ever see or manipulate Model objects

‣ Only interacts with primitives

  ‣ e.g., (setUser(String) instead of setUser(User))

Controller

‣ Glues Model and View together

‣ Updates the view when the Model changes

‣ Updates the model when the user manipulates the view

‣ Houses the application logic

MVP Topology

Concrete MVP Topology

Concrete Example

```
Factory f = GWT.create(Factory.class);
AppController ac = new AppController(f);
ac.showMain();
-->
   View v = f.createView(new ViewController());
   Outline o = f.createOutline(new OutlineController());
```

[gwt.xml maps Factory.class
 to the right type]

```
public interface IJoinTripView {

    Widget asWidget();

    public void setPresenter(Presenter presenter);

    public interface Presenter {
        void onCancel();

        void onJoin(String string);
    }
}
```
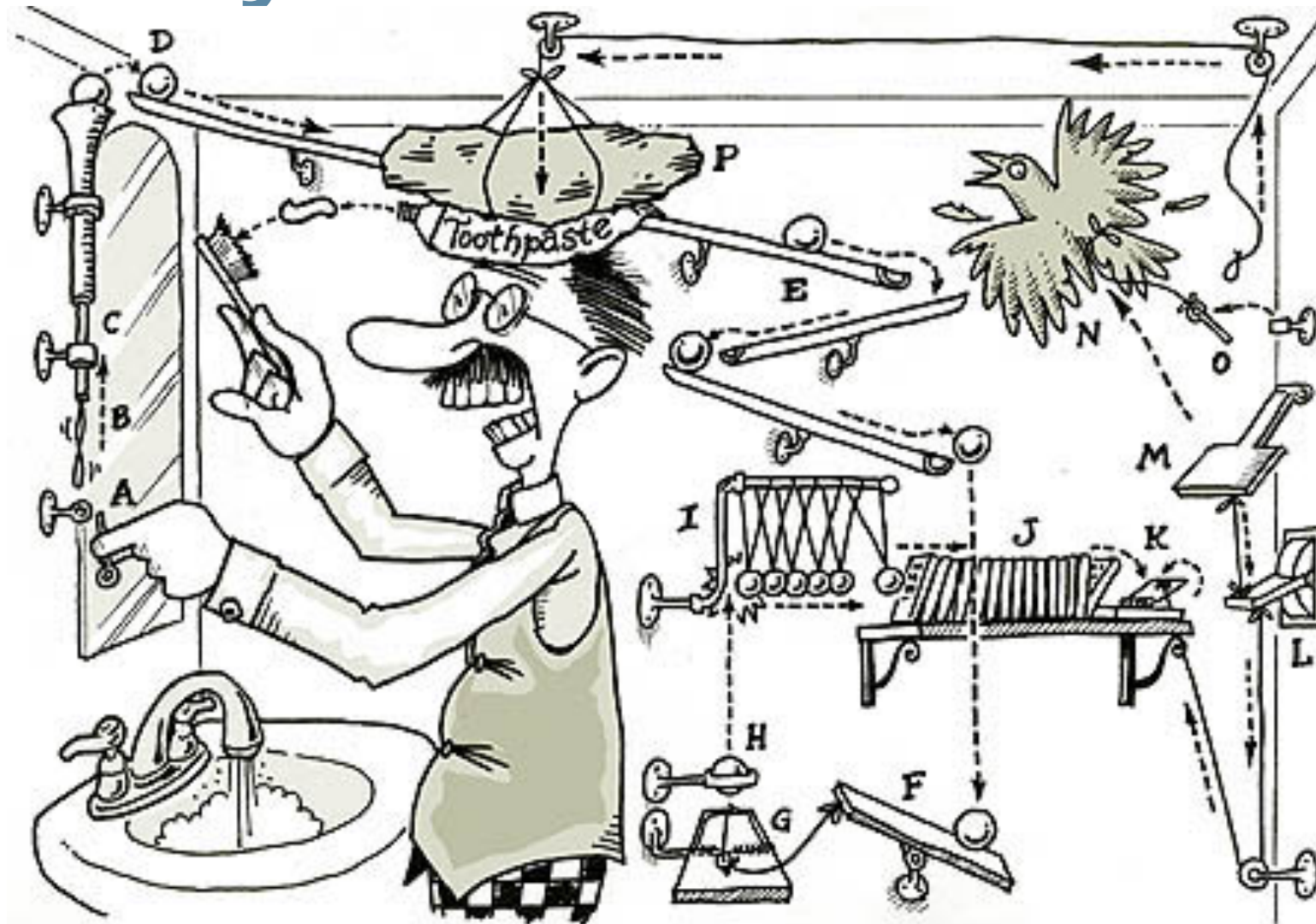
Benefits and tradeoffs

‣ Same as MVC with improved:

  ‣ Decoupling of views from the model

    ‣ Split teams [relieve critical path]

  ‣ Testability [reduce UI testing]

  ‣ A little less complex than MVC [fewer events]

Activity

▸ Apply MV<C|P> to your system.

▸ Identify your events and interfaces.

▸ Groups will present to each other in 15 minutes.