testing rocks!

debugging sucks.
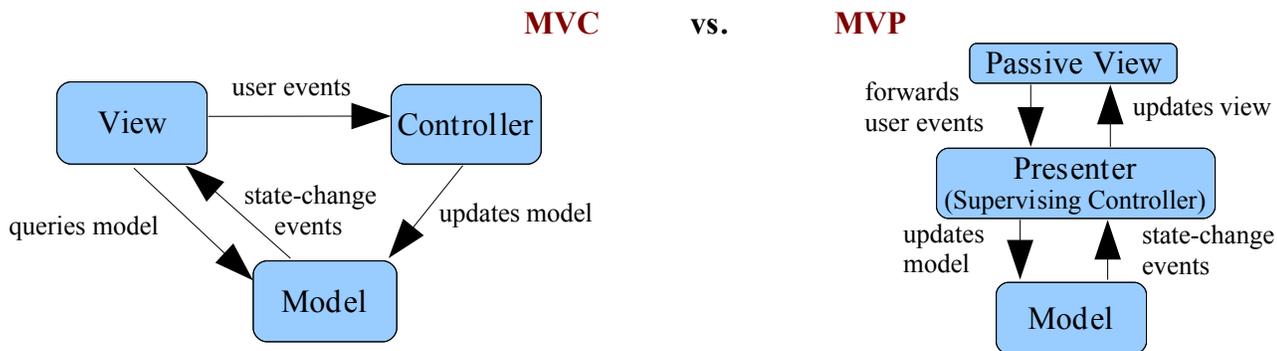
Be an MVP of GUI Testing

With all the sport drug scandals of late, it's difficult to find good role models these days. However, when your role model is a Domain Model (object model of the business entities), you don't need to cheat to be an *MVP*--Use *M*odel-*V*iew-*P*resenter!

MVP is very similar to MVC (Model-View-Controller). **In MVC, the presentation logic is shared by Controller and View,** as shown in the diagram below. The View is usually derived directly from visible GUI framework component, observing the Model and presenting it visually to the user. The Controller is responsible for deciding how to translate user events into Model changes. **In MVP, presentation logic is taken over entirely by a Supervising Controller, also known as a Presenter**.

The View becomes passive, delegating to the Presenter.

```
public CongressionalHearingView() {
  testimonyWidget.addModifyListener(
    new ModifyListener() {
      public void modifyText(ModifyEvent e) {
        presenter.onModifyTestimony(); // presenter decides what action to take
      }});
}
```

The Presenter fetches data from the Model and updates the View.

```
public class CongressionalHearingPresenter {
  public void onModifyTestimony() {
    model.parseTestimony(view.getTestimonyText()); // manipulate model
  }
  public void setWitness(Witness w) {
    view.setTestimonyText(w.getTestimony()); // update view
  }
}
```

This separation of duties allows for **more modular code**, and also **enables easy unit testing of the Presenter and the View.**

```
public void testSetWitness() {
  spyView = new SpyCongressionalHearingView();
  presenter = new CongressionalHearingPresenter(spyView);
  presenter.setWitness(new Witness("Mark McGwire", "I didn't do it"));
  assertEquals( "I didn't do it", spyView.getTestimonyText());
}
```

Note that this makes use of a perfectly legal injection -- Dependency Injection.

More information, discussion, and archives:
http://googletesting.blogspot.com