Designing for Volatility

A look at volatility based decomposition and design

The One True Slide

What?

Why?

When?

How?

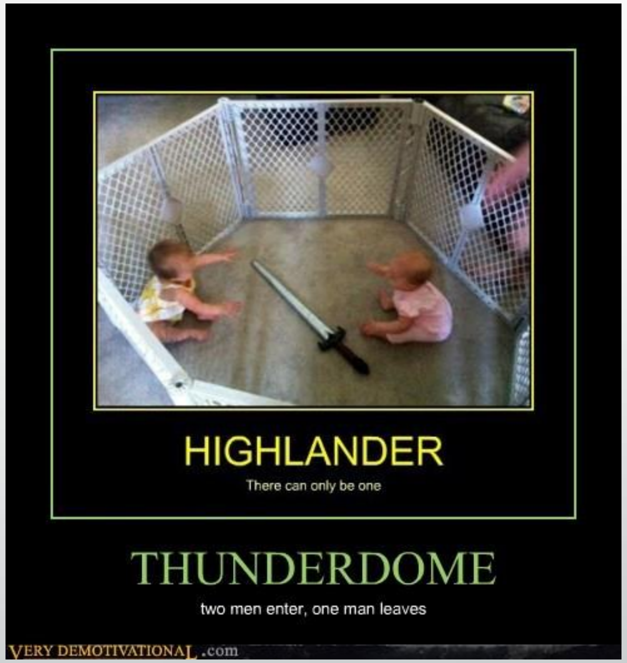
What…is volatility

- According to Vocabulary.com:
  - **Volatility** is the trait of being excitable and unpredictable. Your **volatility** might ultimately be the thing that makes you unsuitable as [an anger management counselor]. The noun **volatility** is the characteristic of changing often and unpredictably.
- TL;DR – Volatility means things change

What… is volatility based decomposition

- Two Primary Types
- Functional Decomposition
  - …is the process of taking a complex process and breaking it down into its smaller, simpler parts…based on the varying functions the process or system performs.
- Volatility based decomposition
  - …is the process of decomposing a process based not on what the system does, but rather on the volatility inherent in the process.

It’s not this serious…



In truth, projects should almost always use BOTH – just not for the same purpose

Why…

- Characteristics of Good Software Design
  - Maintainability
  - Legibility
  - Extensibility
  - Reusability

Why…

- Characteristics of Good Software Design
  - Maintainability
  - Legibility
  - Extensibility
  - Reusability

In Short: **Adaptability**

“Intelligence is the ability to adapt to change” – Stephen Hawking

Why…

- Focus on what hurts most…
  - True business logic / functional concerns tend to be surprisingly self-constraining
  - Volatile interactions hurt more than volatile business rules
- Functional decomposition ignores interaction…
- Change tends to be a much more difficult beast to manage…

(Lando didn't account for volatility)

When

- Initial focus is in the architecture/design phase
    - This is the easiest time to identify and organize volatile components
- Continued attention throughout the development lifecycle
    - Sometimes as an individual component becomes more complex, additional volatility can surface

How…

Patterns already exist to help!

SOLID Principles

- Single Responsibility – Responsibility = Reason to **Change**
- Open/Closed – Deals with how to support **change** within a concept
  - Extension versus modification
- Liskov Substitution – Makes it possible for external contract implementations to **change**
- Interface-segregation – Provides for minimal **change** impact
- Dependency Inversion – Allows all levels of the application to be insulated from **change**

Take Away

- If nothing else, remember the “One True Slide”
  - What…is likely to change
  - Why…is it likely to change
  - When…is it likely to change
  - How…is it likely to change