Design Flaws

Design Defects = Flaws

- Recall that software defects consist of both flaws and bugs
  - **Flaws** are problems in the **design**
  - **Bugs** are problems in the **implementation**

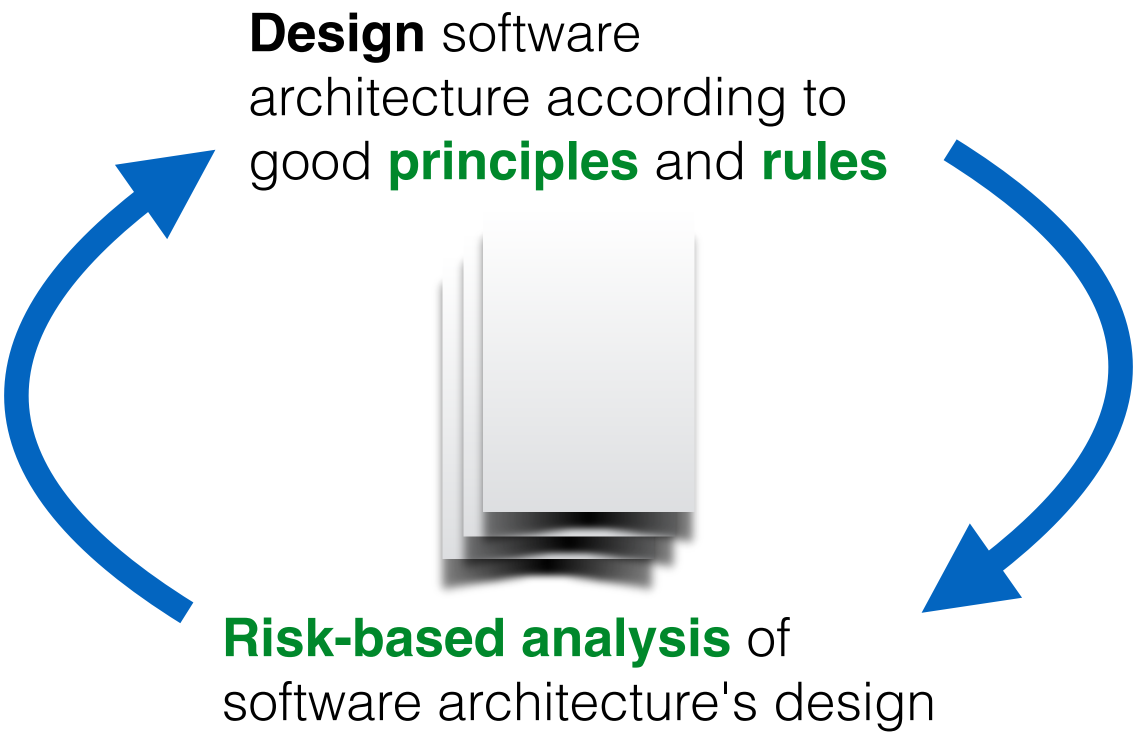
- **We avoid flaws during the design phase**

- According to Gary McGraw,
  50% of security problems are flaws
  - So this phase is very important

Design vs. Implementation?

- **Many different levels of system design decisions**
  - *Highest level*: main actors (**processes**), **interactions**, and programming language(s) to use
  - *Next level*: **decomposition** of an actor **into modules/ components**, identifying the core functionalities and how they work together
  - *Next level*: how to **implement data types** and **functions**, e.g., purely functionally, or using parallelism, etc.
- Last two could be implementation *or* design, or both
  - The distinction is a bit fuzzy

Secure Software Design



Design software architecture according to good **principles** and **rules**

Risk-based analysis of software architecture's design

Principles and Rules

- A **principle** is a high-level design goal with many possible manifestations
- A **rule** is a specific practice that is consonant with sound design principles
  - The **difference between these two can be fuzzy**, just as design vs. implementation is fuzzy.
    - For example, there is often a *principle underlying specific practices*
  - **Principles often overlap**

- The **software design phase** tends to **focus on principles** for avoiding flaws

Categories of Principles

- **Prevention**
  - **Goal**: Eliminate software defects entirely
  - **Example**: Heartbleed bug would have been prevented by using a type-safe language, like Java
- **Mitigation**
  - **Goal**: Reduce the harm from exploitation of unknown defects
  - **Example**: Run each browser tab in a separate process, so exploitation of one tab does not yield access to data in another
- **Detection** (and **Recovery**)
  - **Goal**: Identify and understand an attack (and undo damage)
  - **Example**: Monitoring (e.g., expected invariants), snapshotting

The Principles

- **Favor simplicity**
  - Use fail-safe defaults
  - Do not expect expert users
- **Trust with reluctance**
  - Employ a small trusted computing base
  - Grant the least privilege possible
    - Promote privacy
    - Compartmentalize
- **Defend in Depth**
  - Use community resources - no security by obscurity
- **Monitor and trace**

Classic Advice

The classic reference on principles of secure design is **The Protection of Information in Computer Systems**, by Saltzer and Schroeder (in 1975)

Principles

- **Economy of Mechanism**
- **Fail-safe Defaults**
- **Complete mediation**
- **Open design**
- **Psychological acceptability**
- **Separation of privilege**
- **Least privilege**
- **Least common mechanism**
- **(Work factor)**
- **(Compromise recording)**

http://web.mit.edu/Saltzer/www/publications/protection/Basic.html

Comparing to our list

- Several principles reorganized/renamed
  - *Separation of privilege* has elements of our **compartmentalization**, **defend in depth**
  - *Open design* is like **use community resources**, but did not anticipate open-source code

- **Monitoring** is added
  - Their focus on prevention of attack, rather than recovery

- “Principle” of *complete mediation* dropped
  - CM not a *design* principle, but a rather an implementation requirement