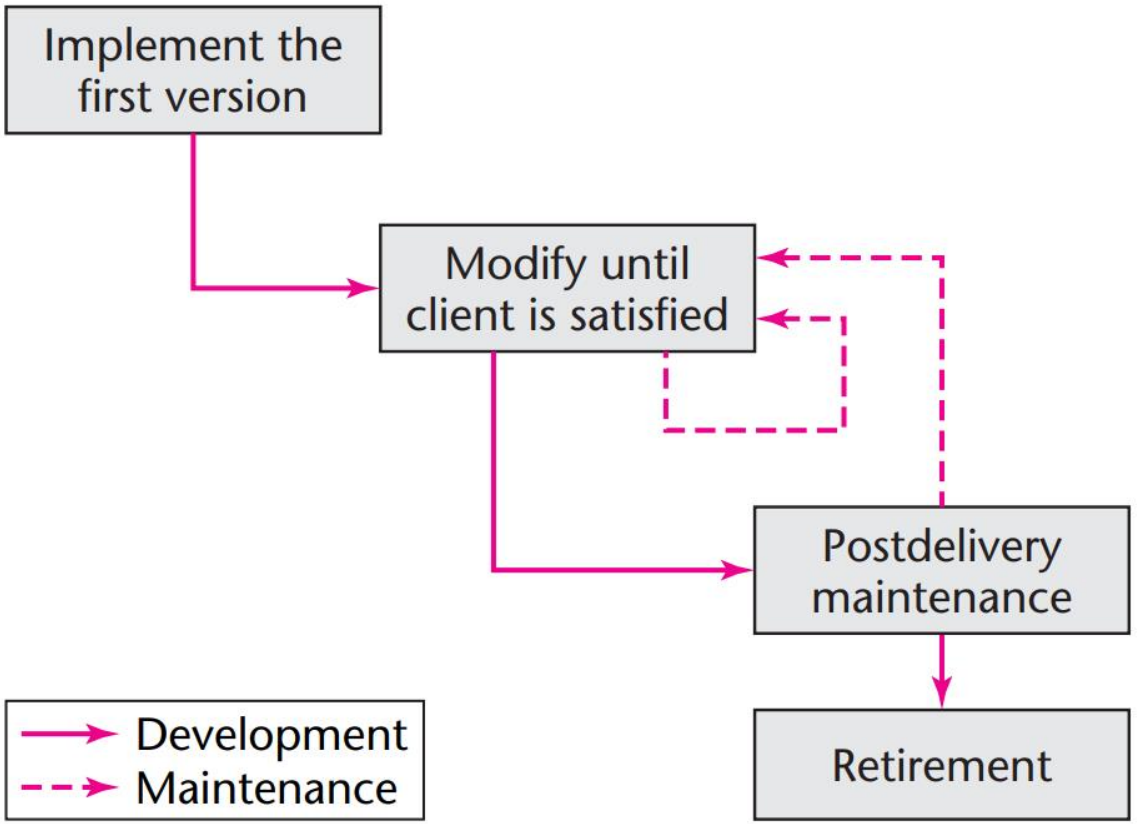
Software Life-Cycle Models (Schach Chap2)

- We Examine a few (of many) Life-cycle Models:
  - Code (Build) and Fix
  - Waterfall
  - Spiral
  - Agile

- Focus on how Phases of Software Development are Incorporated.
- How models evolved.
- How Customer needs are met.

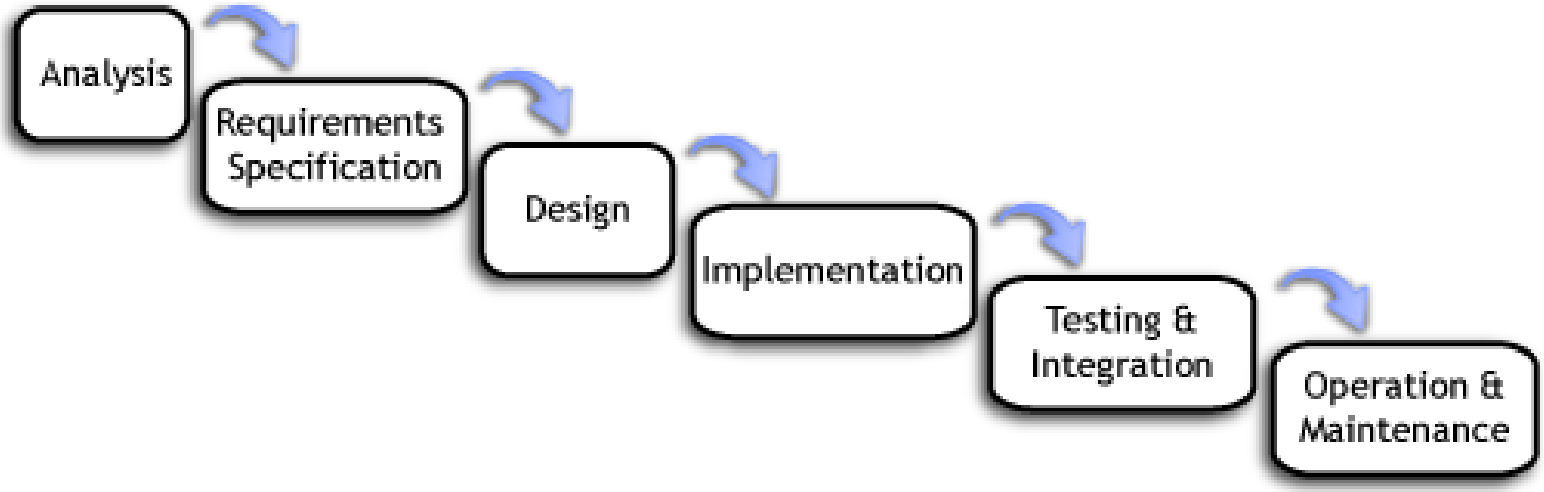
Code (Build) and Fix Model

- No specifications
- No formal design process



Waterfall Model (circa 1970)

- A first cut at improving “code (build) and fix"
  - Each phase of the lifecycle represented as a discrete entity
  - Original: Finish a phase and never re-visit that phase
  - Tweaked over time to include feedback loops between phases
  - Intentionally Documentation-driven



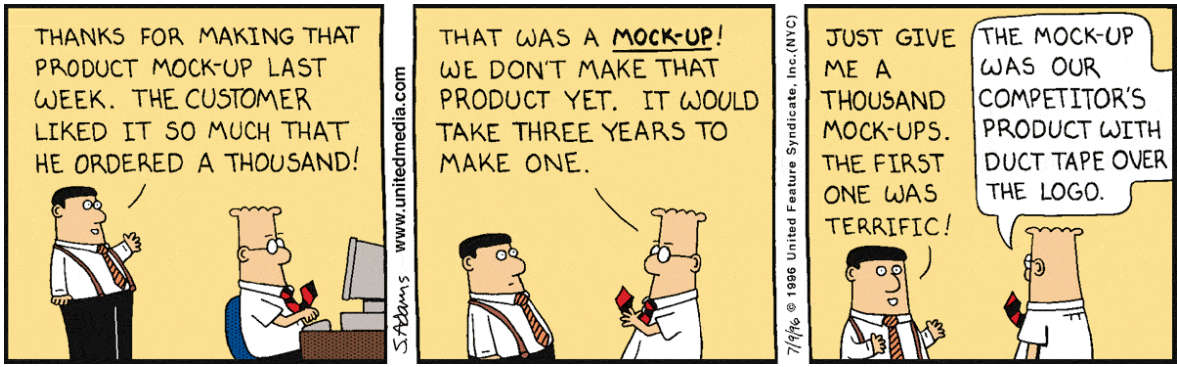
Aside: Requirements Phase (Schach Chap 11)

"I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant!"

Misconception: Must determine what client ***wants***

Reality: Must determine client's ***needs***

- Rapid prototyping
  - Key functionality
  - What client sees
- Scenarios of Use



- **Goal**: *objectively validate-able* requirements

Aside: Requirements Phase (Schach Chap 11)

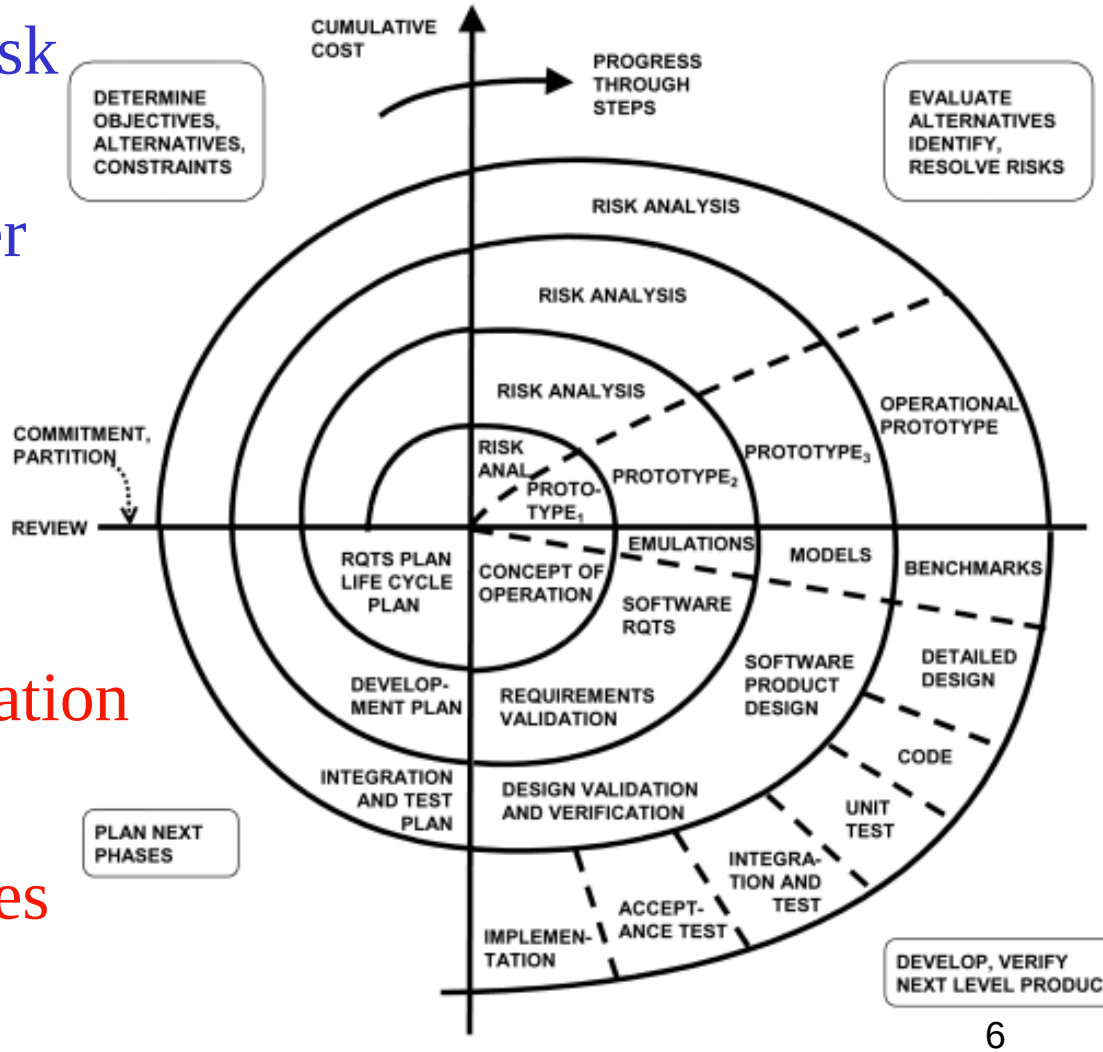
- **Goal**: System's Requirements and the set of Acceptance Test Plan Test Cases that objectively validate them
  - **Normal** test cases demonstrate that the software meets the indicated functional requirement.
    - *Expected* uses of the system
  - **Abnormal** test cases give the result of the software entering an unusual state, such as when a user provides invalid input.
    - *Unexpected* (but possible) uses of the system
  - **Not Useful:** Ones that describe states that *properly running* software *cannot* be placed in.

    Not testable - Avoid such attempts

- Ex: Checkout registers:
  What's wrong with the below Requirement?!?
  - Checkout registers must be *fast.*

Spiral Model (circa 1980s -> manages risk)

- *Evolutionary* development, initially just define high priority requirements:
  - Prototype, access risk get user feedback
  - Continue with lower priority items.
  - Features:
    - Commitment partition/cancellation
    - Risk Analysis
    - Lots of prototypes
    - User Feedback



Analysis of Spiral Model

- Strengths
  - No distinction between development, maintenance
  - **Risk-driven** (focus resources where needed)

- Weaknesses
  - **Intent**: Only useful for large/in-house software (can be cancelled w/o breaking a contract if deemed too risky)
  - **Cost:** RA is too **costly** to use for small projects
  - **Risk-driven** (what if **poor** risk evaluation?)

Agile Model circa 2000s

- Development tasks broken down into small increments with minimal planning. Iterations are of very short duration that typically last from 1 to 4 weeks.
  - Each development cycle:
    - Requirements/Spec adjustments
    - Design/coding
    - unit testing, and acceptance testing.
  - Agile: Embraces changing customer requirements.