SOFTWARE PROCESS REVIEW

CST315 – Fall 2015 Revision

Arizona State University

Some notes from Sommerville 9th edition

OUTLINE

- Software Process Phases
- Software Lifecycle Models
  - Build-and-Fix
  - Waterfall model
  - Evolutionary development
  - Iterative and Incremental
  - Spiral model
  - RUP
  - Agile/XP
- Summary of Pros/Cons of each model
- Software Design and the SDLC

Hopefully these are topics you have seen in other classes or places.

In next module, we will introduce a completely new process: User-Centric Design (UCD). UCD will be the process we use in the semester project.

PRELIMINARY

- Why a process?

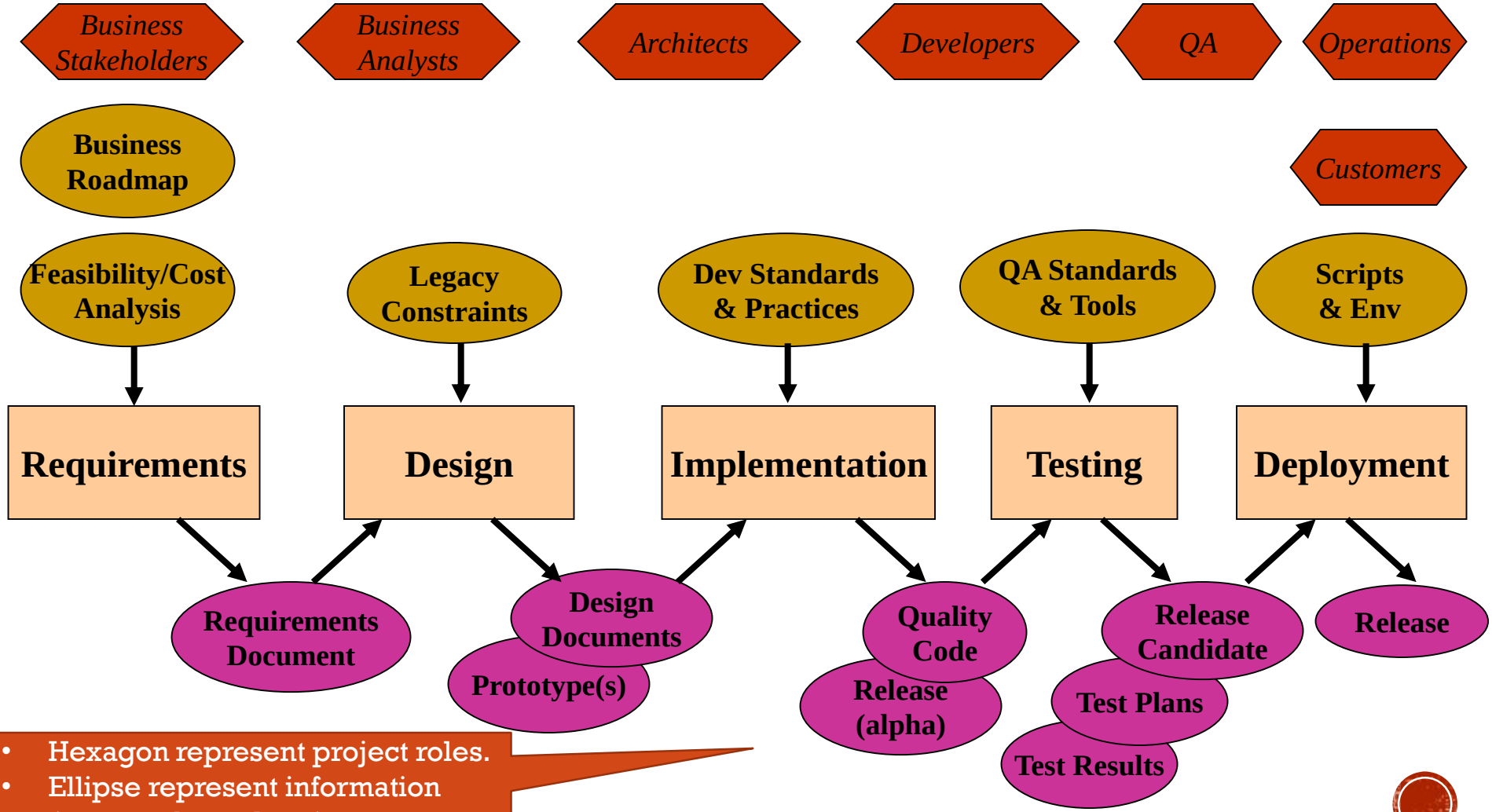
- Repeatable: everything went right - how do we do it again?

- Permits Analysis: something went wrong, need to find the root of an issue.

- Informs Estimation: can better predict time or budget when adhering to a process.

SOFTWARE PROCESS FLOW (10K FT VIEW)

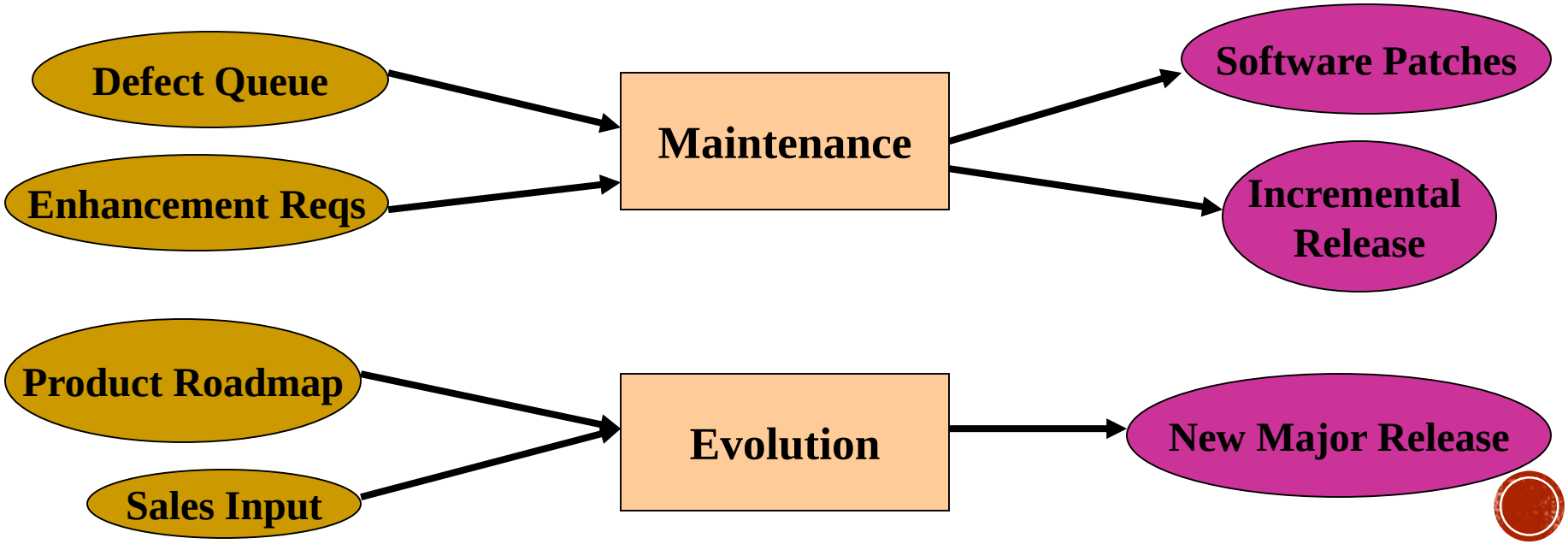
An Over-simplified View of the Software Development Lifecycle (SDLC) Process Phases



- Hexagon represent project roles.
- Ellipse represent information (e.g., work products).
- Rectangles represent phases.

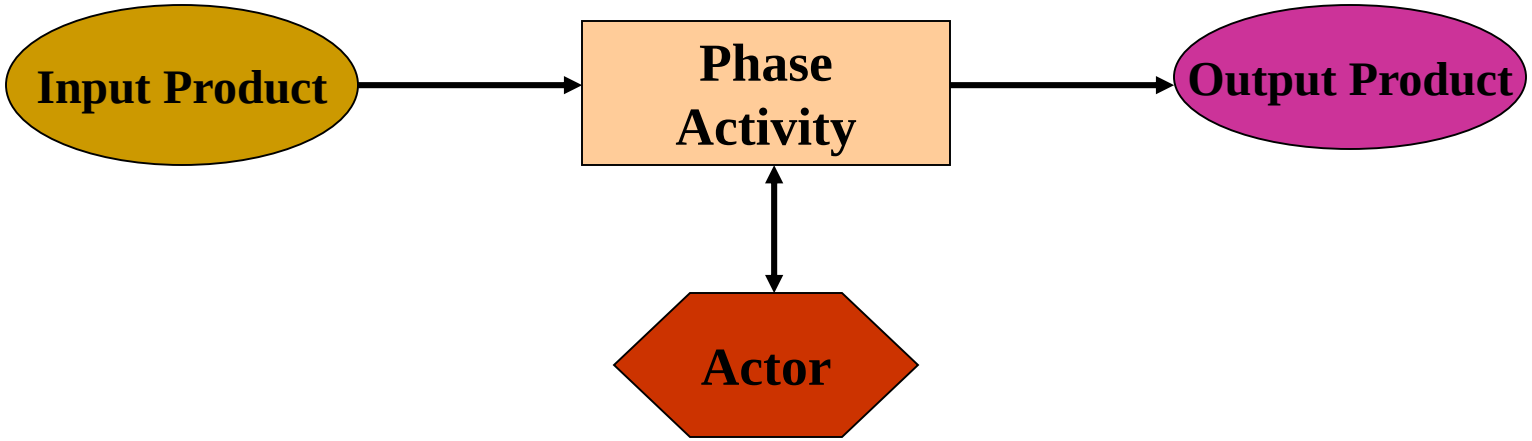
POST-RELEASE (POST-DEPLOYMENT)

- *Maintenance* and *Evolution* are discussed synonymously, but they really aren't
  - *Maintenance*: The ability to support the correct functioning of the current release in the face of incremental changes in requirements
  - *Evolution*: The ability to incorporate significant changes in product direction in the current architecture



SOFTWARE PROCESS PHASES

- As an exercise, consider the following questions:
  - How are the phases connected?
    - What transitions exist between each phase?
    - What data is needed by each phase?
    - What data is produced by each phase?
    - Who are the players (the actors) involved in each phase?
- Given the elements of each phase, what is the *lifecycle model* that captures software process?

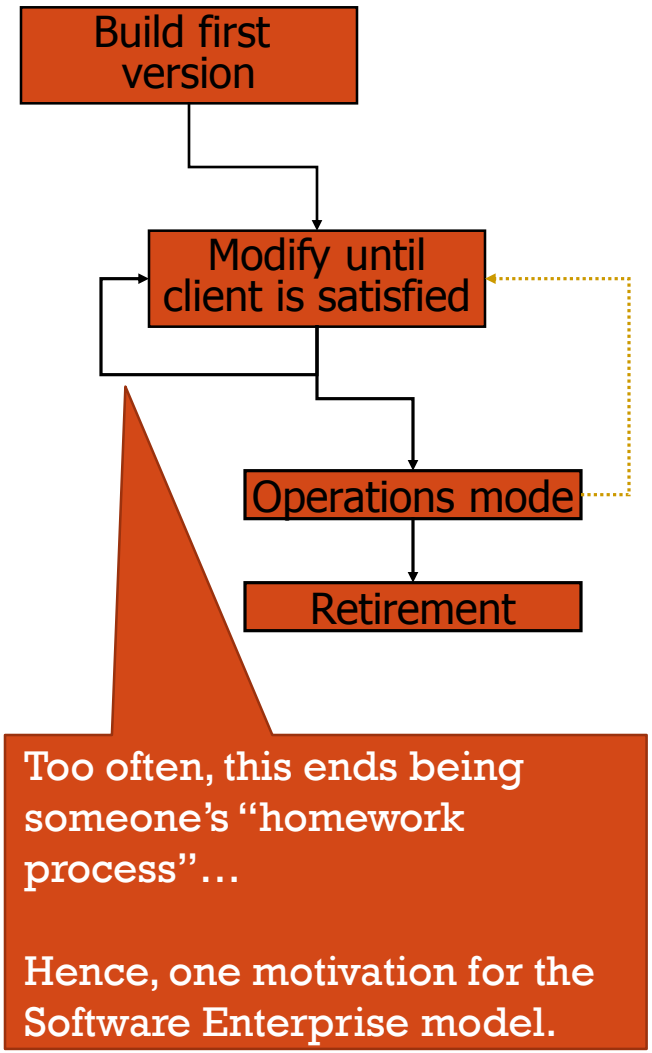


POPULAR SOFTWARE PROCESS MODELS

In this slide deck, we'll review six process models:

- Build-and-Fix
  - The "defacto" process
- Waterfall and "V" models
  - Distinct phases of specification, development & test
- Iterative and Incremental
  - Iterate over phases, build the system incrementally
- Spiral model
  - Risk management as a key phase in the process
- RUP
  - Use-case driven, architecture-centric
- Agile/XP
  - Embrace change; empower people; code!

BUILD-AND-FIX



Pros:

- No process overhead; programmers program!

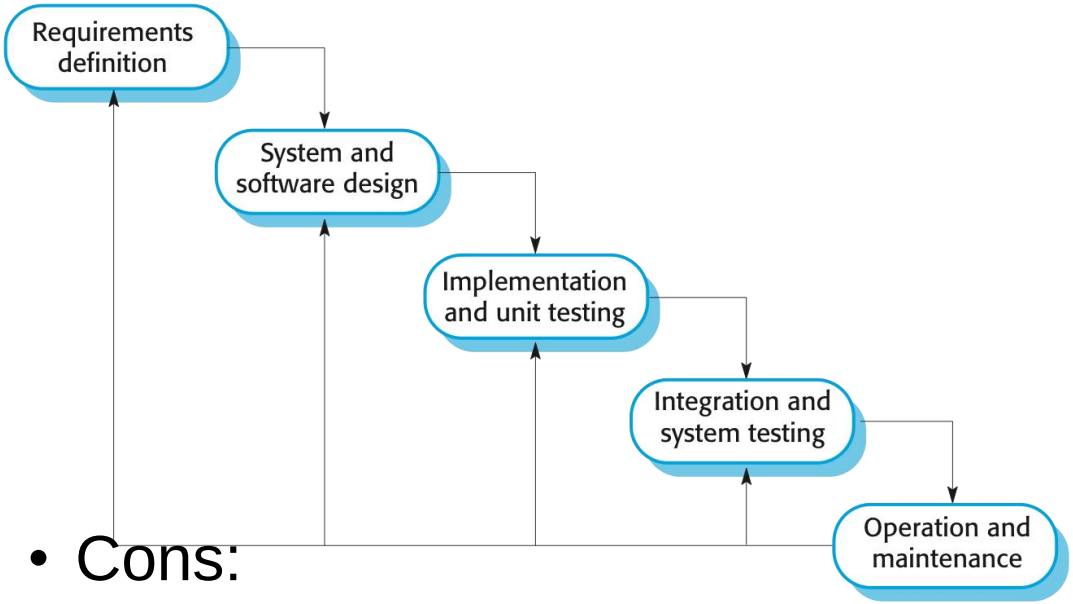
Cons:

- CHAOS!
- Requirements may change; even when they don't how do we track progress against them?
- No quality management
- No design documentation
- No sustainable architecture
- Communication problems abound!
- Budget overruns
- Product doesn't meet expectations

When to use:

- When you have no other choice!

WATERFALL MODEL



Pros:

- Each process phase well-defined
- Inputs and Outputs of each phase defined

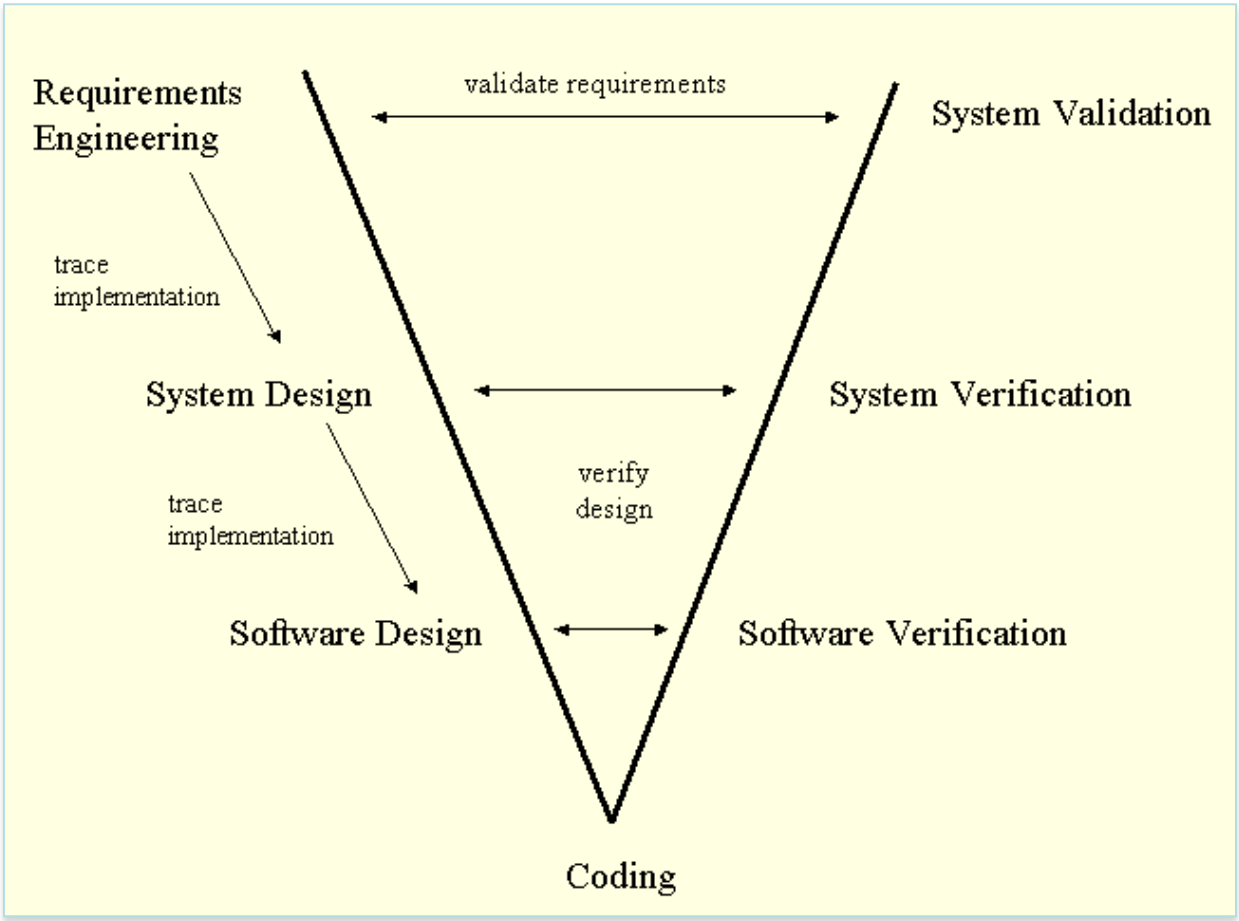
- Cons:
  - Inflexible partitioning of the project into distinct stages
  - Difficult to respond to changing customer requirements
  - Difficult to accommodate change of any kind
  - Document-centric
- When to use:
  - This model is only appropriate when the requirements are well-understood and truly “frozen”

V-MODEL

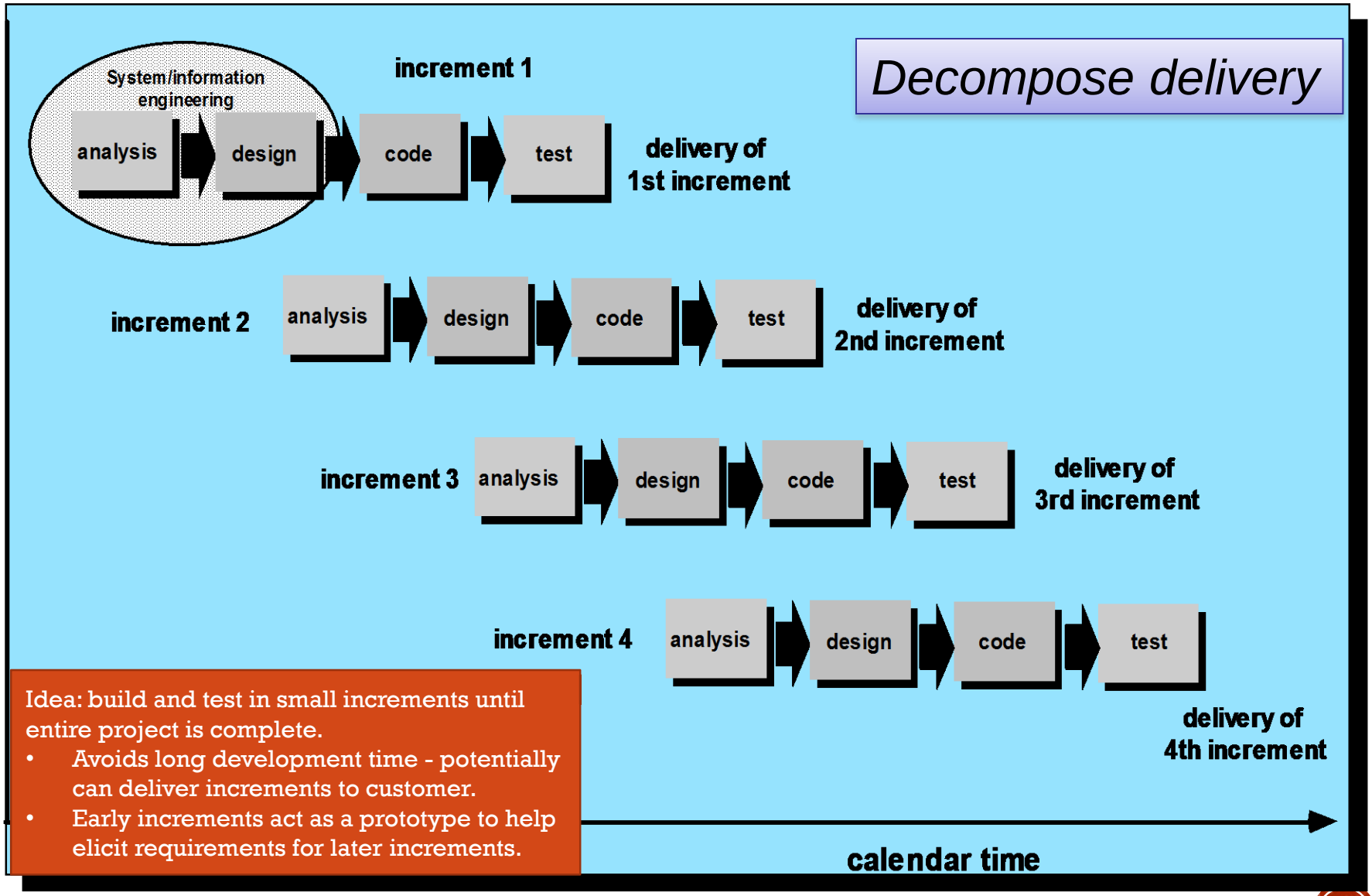
- An extension of the waterfall model with roots in the system engineering process.

Note:

- Down left side is a "waterfall"
- Right-side has parallel validation activities
- Recommended practice now is to "chain" or "cycle" many V-models



PROCESS PRINCIPLE: INCREMENTS

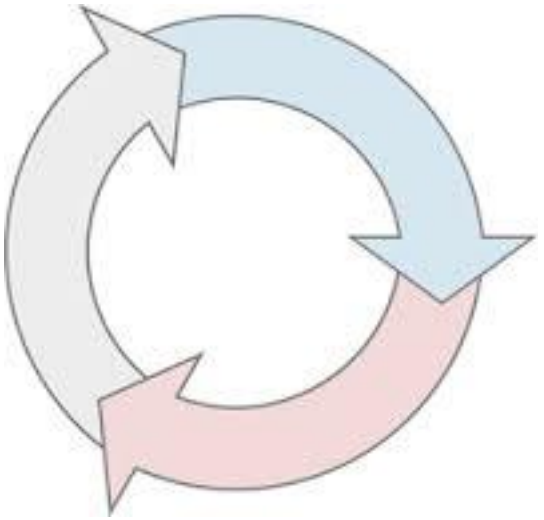


Idea: build and test in small increments until entire project is complete.

- Avoids long development time - potentially can deliver increments to customer.
- Early increments act as a prototype to help elicit requirements for later increments.

PROCESS PRINCIPLE: ITERATION

- Idea: Design, Develop (Prototype), and Test. Then, repeat, refining the project until it meets requirements.
- Assumption:
  - Requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems
- Iterative process models are considered the current "best practice" for software methodologies
- Approaches that incorporate iteration
  - Synchronize and Stabilize
  - Spiral development
  - RUP
  - XP

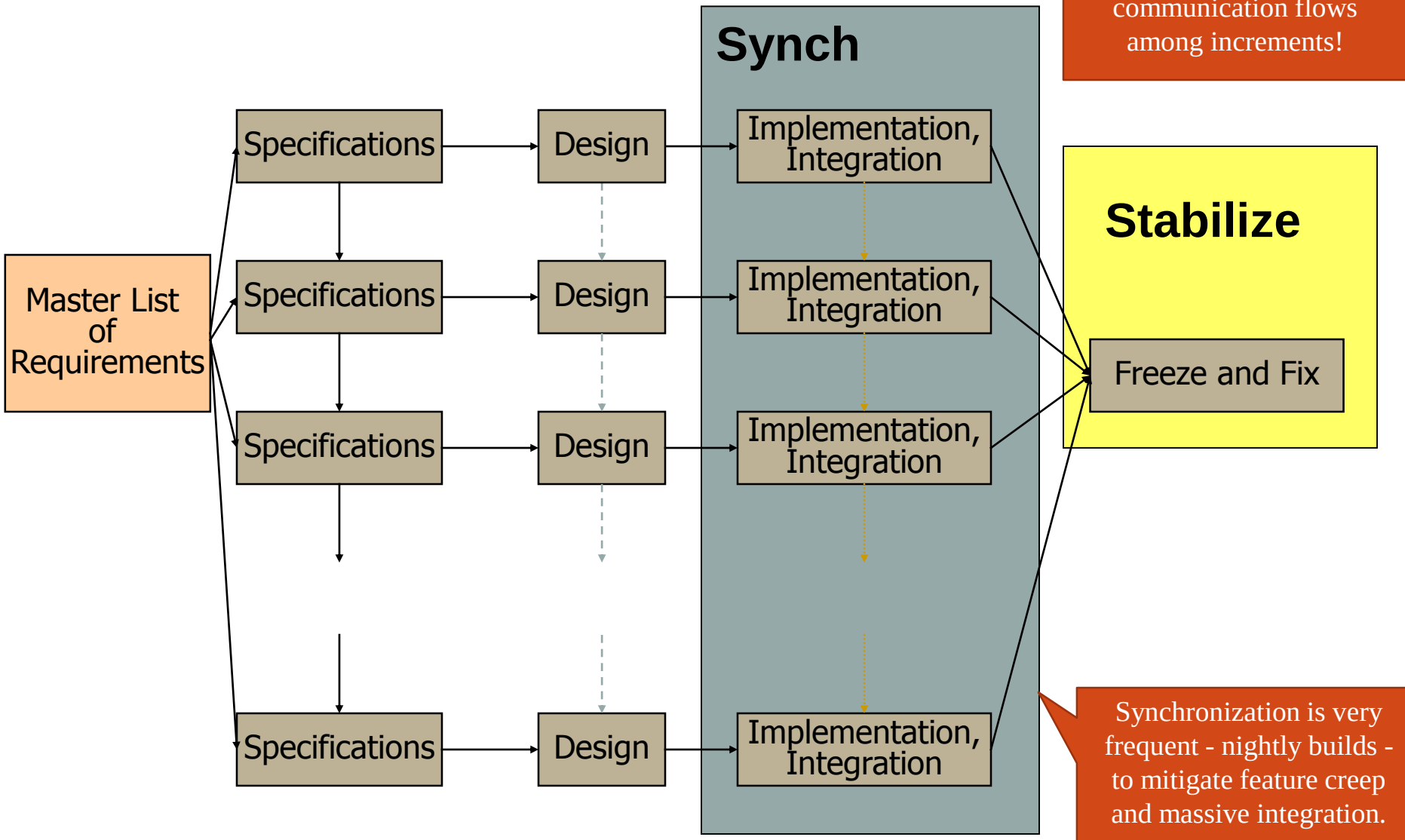


ITERATIVE & INCREMENTAL: PROS & CONS

- Pros:
  - Project broken into smaller, more manageable pieces
    - Easier to clarify requirements
    - Small teams may be used to address each increment
- Cons:
  - Poor system architecture
    - "narrow" prioritizing by customer
    - Each increment evolves independently
  - Localizing requirements creep to a particular increment
  - "Big-bang" integration
    - What happens when the increments don't line up?
- When to use:
  - When top-level requirements are properly partitioned
  - When a lot of outsourcing/COTS is used

SYNCHRONIZE AND STABILIZE

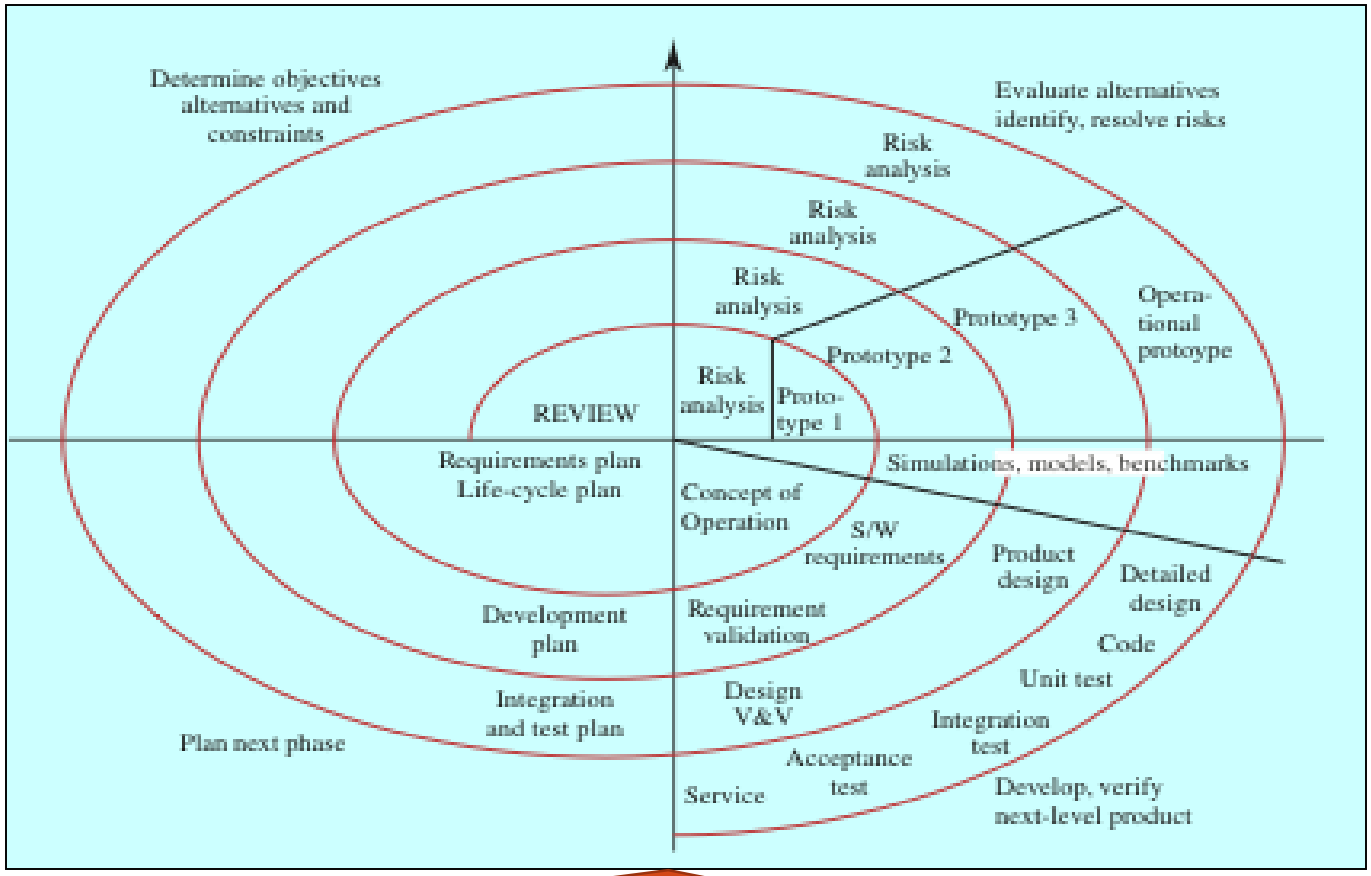
Overall idea is similar to increments. One key focus: communication flows among increments!



Synchronization is very frequent - nightly builds - to mitigate feature creep and massive integration.

Specification team Design team Implementation/integration team

SPIRAL MODEL OF THE SOFTWARE PROCESS



- Risk-driven
- Prototyping
- Expand and Converges

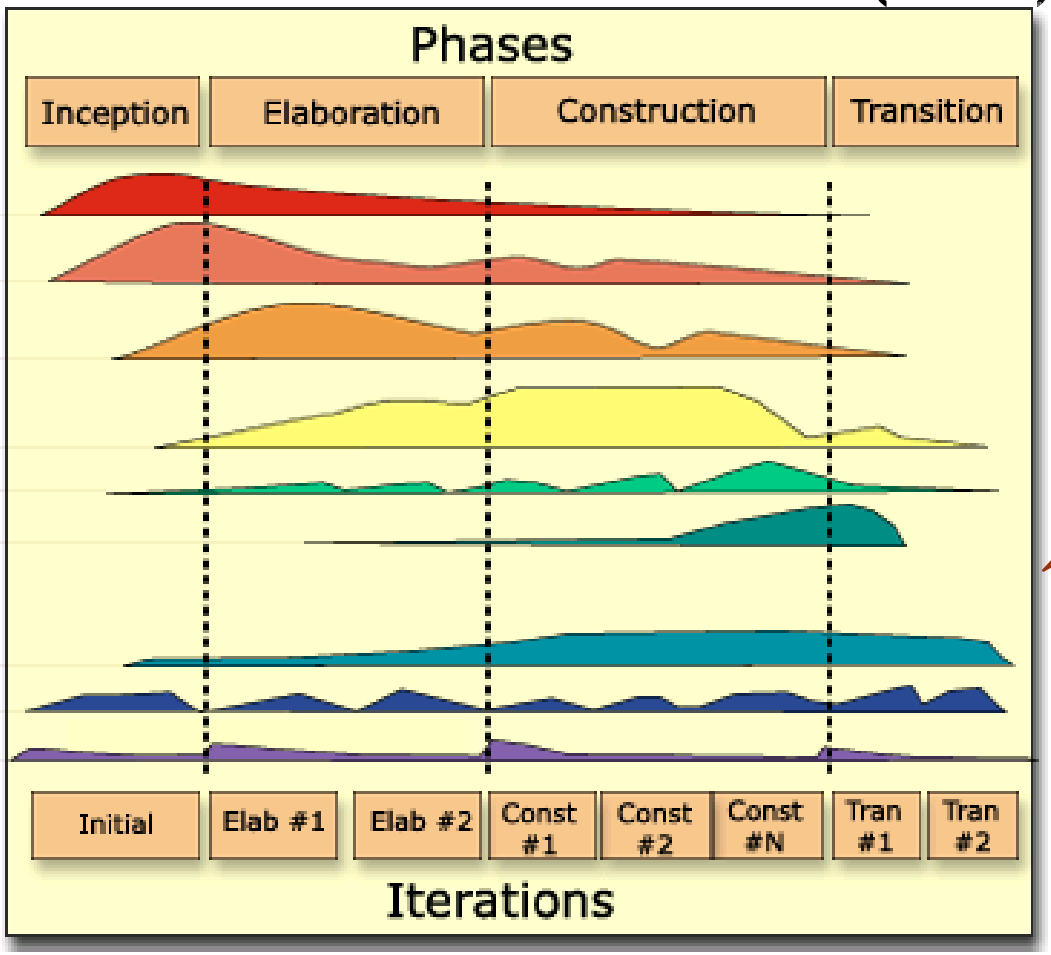
Pros:

- Each iteration should bring you closer to a finished product
- "Convergence"
- Risks are quickly identified and addressed
- Provides a distinct communication point between engineers and managers
- Free to use any specific model in each "cycle"

Con:

- What happens if prototypes diverge? May spiral out of control!

RATIONAL UNIFIED PROCESS (RUP)

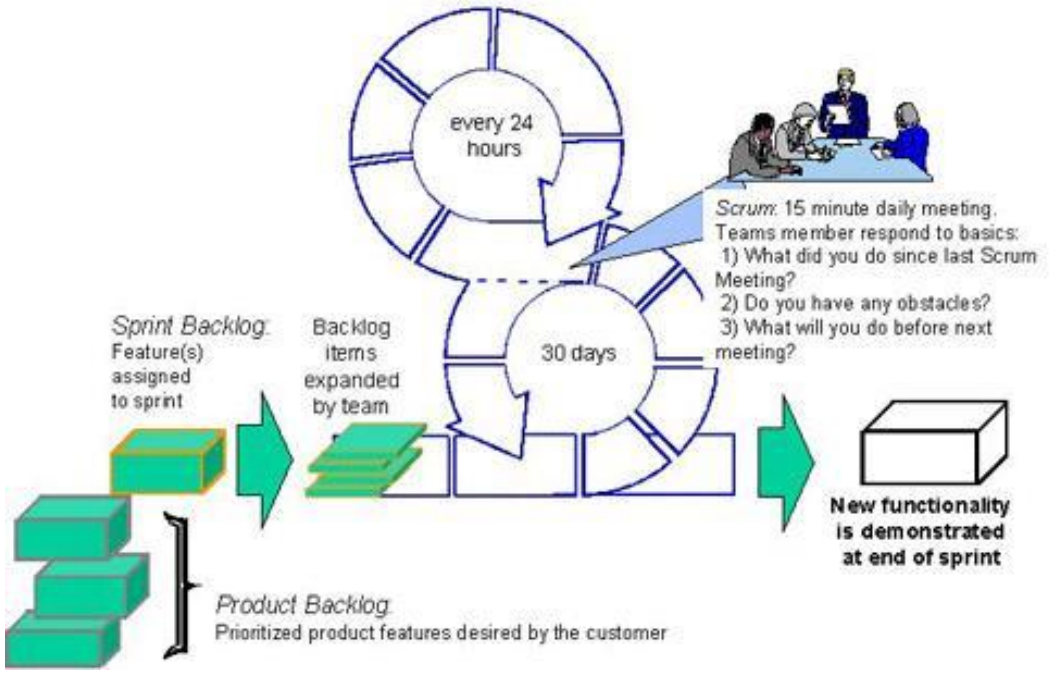


IBM provides many documents online that detail RUP in depth.

For us, the key idea is realizing that phases overlap.

- Pre-Agile popular model, still is widely adopted
- Explicitly identifies Workflows and Transition Phase
- Over-commercialized; accused of being waterfall-ish

AGILE EXAMPLES: SCRUM, EXTREME PROGRAMMING

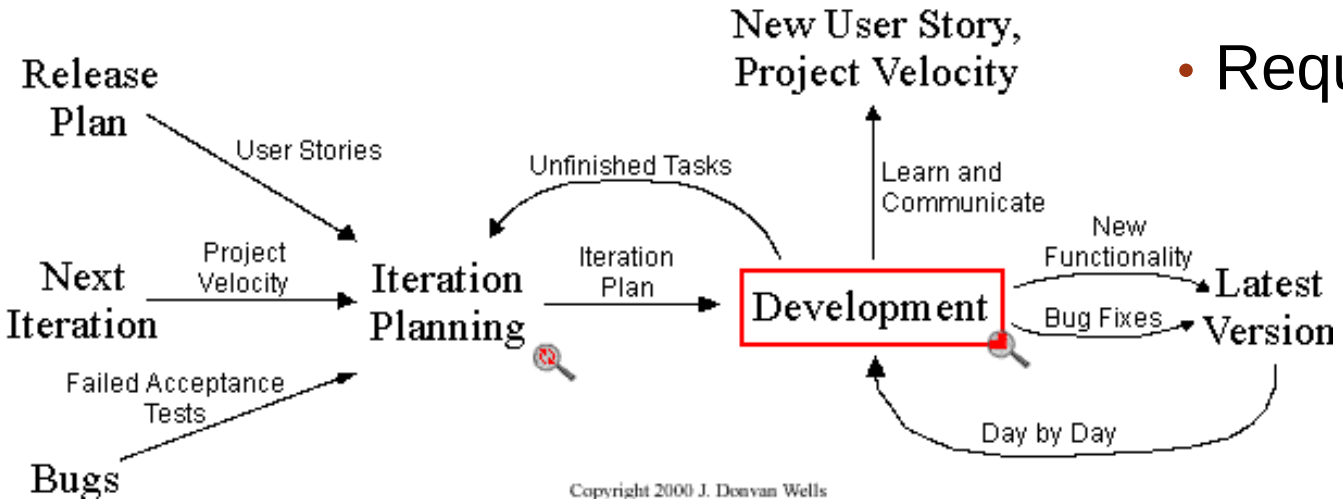


Pros:

- Adapts to change
- People-centric
- Fast & light
- Fosters dev buy-in

Cons:

- Management buy-in
- Pseudo-agile
- Requires good people!



A COMPARISON OF LIFE CYCLE MODELS

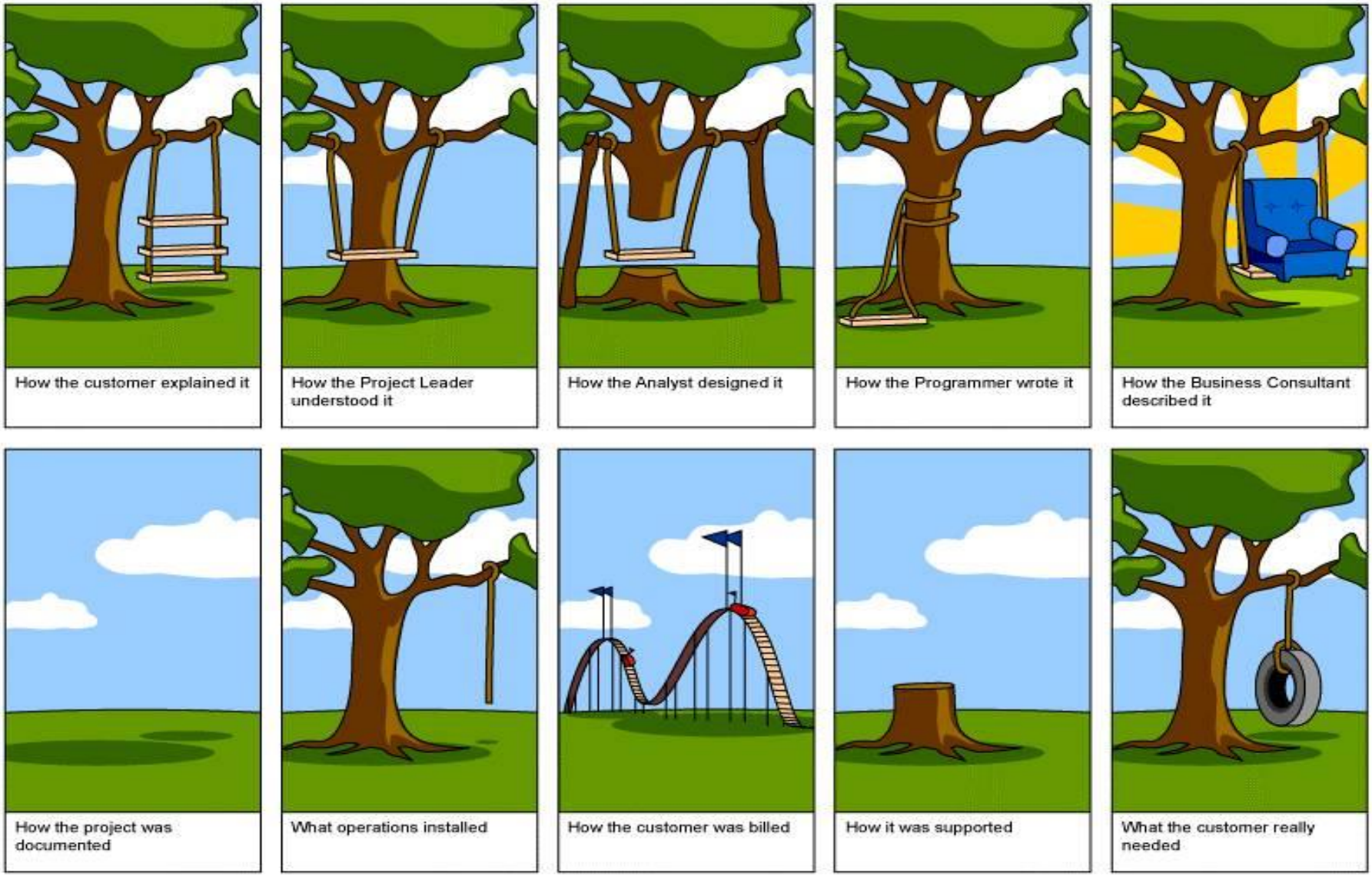
Model	Strengths	Weaknesses
Build-and-Fix	Fine for small programs that do not require much maintenance	Totally unsatisfactorily for nontrivial programs
Waterfall	Disciplined approach Document driven	Delivered product may not meet client's needs
Incremental	Manageable increments User sees end functionality sooner	Architecture mismatch Lack of communication
Synchronize & Stabilize	Requirements change is manageable Ensures components can be successfully integrated	Complex to manage Complex to architect
Spiral	Incorporates features of all the above models Global view always revisited	Can be used only for large-scale products Engineers have to be competent at risk-analysis
RUP	Incorporates current process thinking Strong tool support Recognizes deployment	Commercially-driven All things to all people
Agile/Scrum/XP	Light process that "stays out of the way" Recognize and embrace change	Scalability is a question Relies on experienced developers

SOFTWARE DESIGN AND THE SDLC

SOFTWARE DESIGN

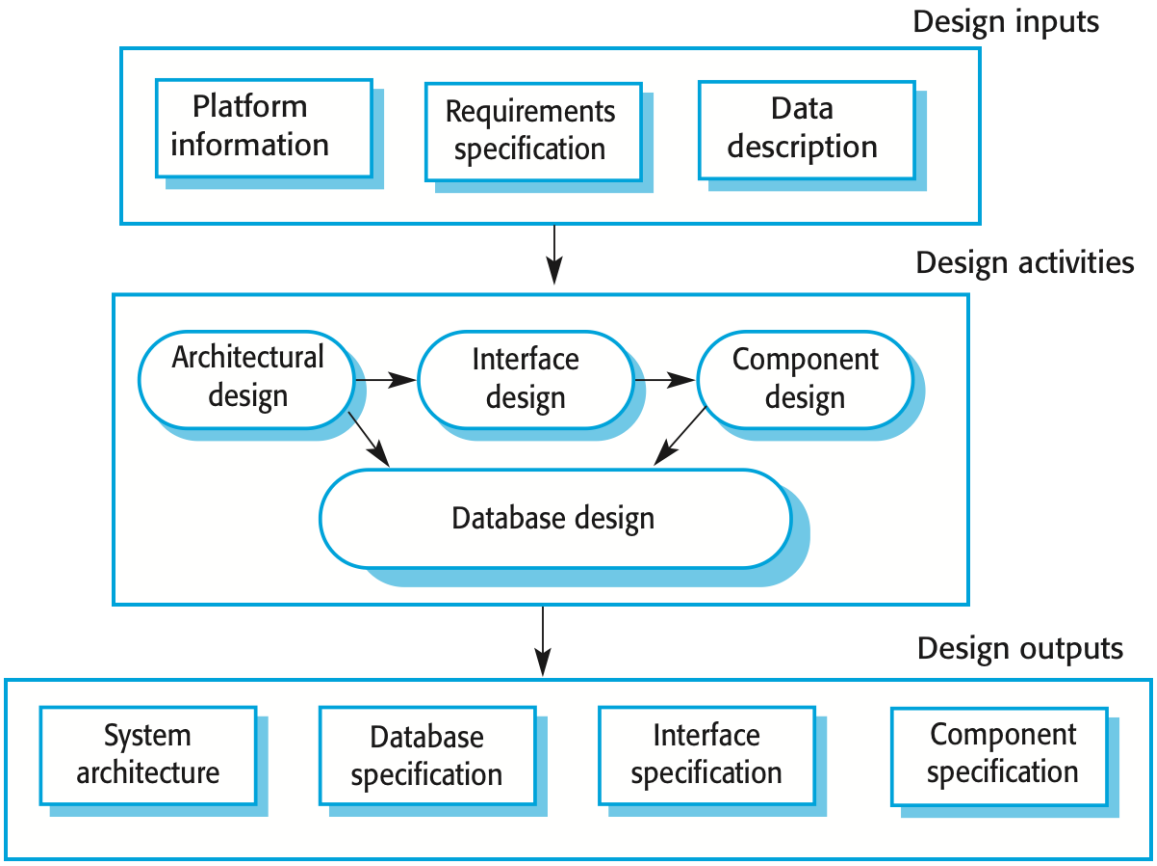
Communication is hard - there is often a disconnect…

http://projectcartoon.com/cartoon/2



General Software Design Process

- From your textbook (9e):
  - Somewhat ignores UI Design!



Architectural design, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed.

Interface design, where you define the interfaces between system components.

Component design, where you take each system component and design how it will operate.

Database design, where you design the system data structures and how these are to be represented in a database.

Software Design and the SDLC

- Design plays role in SDLC models covered earlier
  - Quite a bit in Waterfall and V
    - Model-Driven Architecture (MDA), systems engineering
  - Iterative & Incremental: recall each iteration can use a mini-lifecycle as it desires
  - Spiral: prototyping is a design activity
  - RUP perspective (architecture w/ iterative design)
  - Only to the extent needed in Agile!
- What about User Interface Design
  - UI Design is a "traditional design process"
  - Focuses on interactions and the end user
  - Has the advantage of being "visible"

SO, WHAT IS OUR PROCESS?

- We will leverage the *iterative*, *selective* and *refining* nature of a design process
  - Prototype solutions in the face of incomplete & inconsistent info
  - The prototype is the vehicle of communication – about your assumptions and understanding, not just UI specific
- What about Software Design?
  - We will address software design in 2 parts:
    - User Experience Design – Interaction Design and UI Design
    - Software System Design – Modeling object structures & communication with UML
- Modeling and Prototyping will be a key activity for both!
  - On the UX side, we will *model* end users and *prototype* UIs make tangible our designs and get feedback
  - On the System side, we will *model* in UML and prototype to explore technology feasibility of a solution path