Managing Software Development

Software Development Lifecycles

Presenter: David Root
(Material developed with Tony Lattanze)

Session Objectives

- Software development lifecycles
  - Defined
  - Difference from “process”
  - Compare to development variables
  - Common Lifecycles

So…

Why should we care about this subject?

(E.D. Hirsch Cultural Literacy?)

What is a Life Cycle?

- Websters (1892):
  - “The series of stages in form and functional activity through which an organism passes between successive recurrences of a specified primary stage.”
- Reifer (1997): (product)
  - “Period of time that begins when a software product is conceived and ends when the product is retired from use.”

What is a Life Cycle?

Tony Lattanze

- The *software lifecycle* is the *cradle to grave* existence of a software product or software intensive system
    - includes initial development, repairs, and enhancement, and decommission
- Management of the entire lifecycle of a software intensive system requires a deeper knowledge than basic in-the-small development intuition and experience

More on What…

- Lifecycle models attempt to generalize the software development process into steps with associated activities and/or artifacts.
    - They model how a project is planned, controlled, and monitored from inception to completion.
- Lifecycle models provide a starting point for defining what we will do.
- But, what is the end point of a project?

So…What is a Process?

(remember this for the process lectures)

- A process is a sequence of steps performed for a given purpose.

Websters:

"a series of actions or operations conducing to an end."

The concept of software process is rarely presented in undergraduate education.

Process ≠ Lifecycle

- Software process is not the same as life cycle models.
  - ☐ process refers to the specific steps used in a specific organization to build systems
  - ☐ indicates the specific activities that must be undertaken and artifacts that must be produced
  - ☐ *process definitions* include more detail than provided lifecycle models
- Software processes are sometimes defined in the context of a lifecycle model.

So, what is important?

- What you call “it” isn’t.
- What stakeholders understand is.

Life Cycles

- Ad Hoc
- Classic (waterfall)
- Prototype
- RAD
- Incremental
- Spiral
- WinWin
- V model
- Chaos

Concurrent COTS 4th Gen

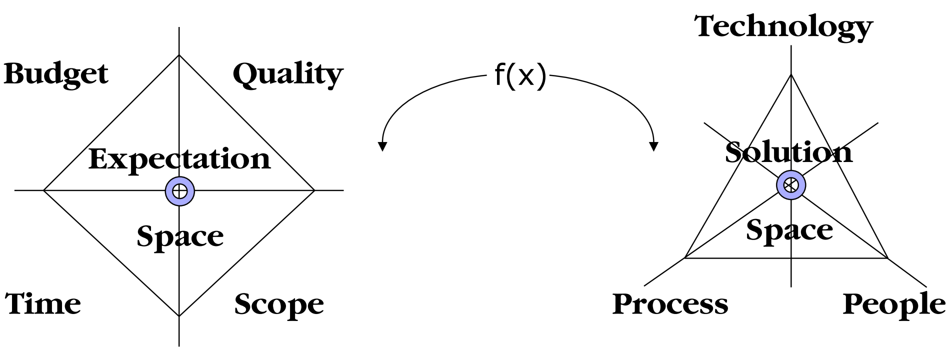
Be very careful here

- Is this just semantics?
- Are there standard definitions?
- How should approach this with a new project?
- Remember, we tend to think linearly, sequentially. Is this a problem?

Define, communicate, define, communicate..

Remember this when looking at SDLC's

Customer's view **Developer's view**



Budget
Quality
Expectation
Space
Time
Scope

f(x)

Technology
Solution
Space
Process
People

f(x) = f(Planning, Process, People, Product, ?.....)

Also need to look at with respect to:

- Stakeholders
  - Backgrounds, domain expertise
  - Commitment to project
- Environments
  - Business / market
  - Cultures
- Moral, legal constraints

So, when looking at projects

Need to ask:

What SDLC would *define* my project best?

(The project drives the lifecycle, not the other way around)

- What criteria are important for the project?

Project criteria….

Ad Hoc
"Hobbyist"

- Legacy
- Code – Test – Code – Test………
  - Becomes a mess, chuck it, start over
- Design (high level) – Code – Test – Code – Test…..
  - (Reality was Code - Test – Code – Test – Document the resulting design)
- Lack of defined, formalized processes

Is this the same as "no process?"

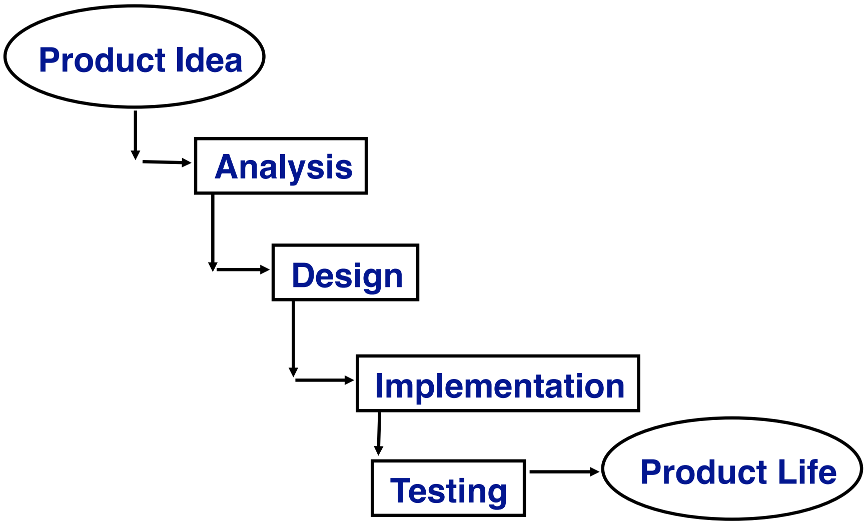
Waterfall Model

- First proposed in 1970 by W.W. Royce
- Development flows steadily through:
  - requirements analysis, design implementation, testing, integration, and maintenance.
- Royce advocated *iterations* of waterfalls adapting the results of the precedent waterfall.

Waterfall Model

- Technology had some influence on the viability of the waterfall model.
  - slow code, compile, and debug cycles
- Reflected the way that other engineering disciplines build things.
- Formed the basis of the earliest software process frameworks
- Waterfall is still used today (but no one will admit it). Has a bad reputation. Why?

Waterfall (linear) (Classic) Model Intent



Product Idea

Analysis

Design

Implementation

Testing

Product Life

©David Root, 2010, all rights reserved

Waterfall Problems

- Increasing use of resources?
- Oops
  - Go back to a previous step
  - Progressively more costly
- Downside
  - Cost
  - Time
  - Cascading Bugs
- Where appropriate?

©David Root, 2010, all rights reserved

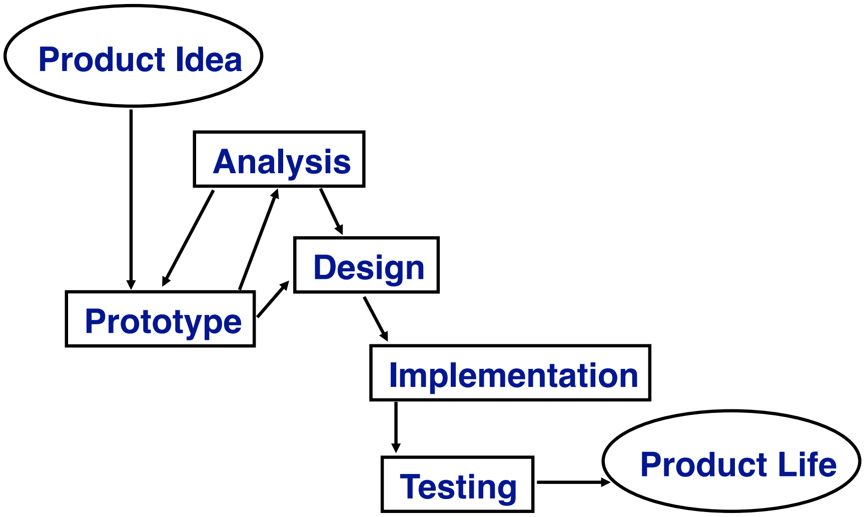
From Chris Kemerer……
Reality of Waterfall

1. Enthusiasm
2. Disillusionment
3. Panic & Hysteria
4. Search for the Guilty
5. Punishment of the Innocent
6. Praise & Honors for the non-participants

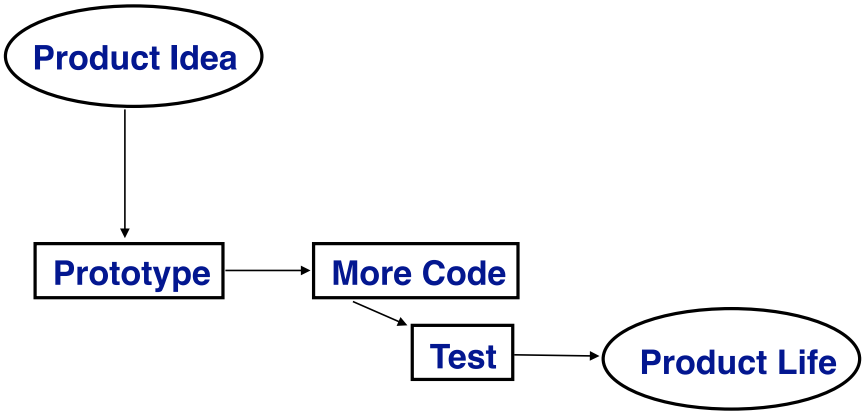
Prototypes

- Throw Away (Rapid)
  - ☐ Proof of concept – It can be done
  - ☐ End point unknown!
- Evolutionary
  - ☐ Keep something
  - ☐ Different than incremental?
  - ☐ The evolutionary development model can be distinguished from the prototyping model in that
    - a final product is typically specified
    - the product features are *evolved* overtime to some predetermined final state

The Rapid Prototype Model



A Common Misuse of the Rapid Prototype Model



What are the problems with the prototype lifecycle?

When would you use it:

Weaknesses:

Incremental Model

(One of the most misused definitions)

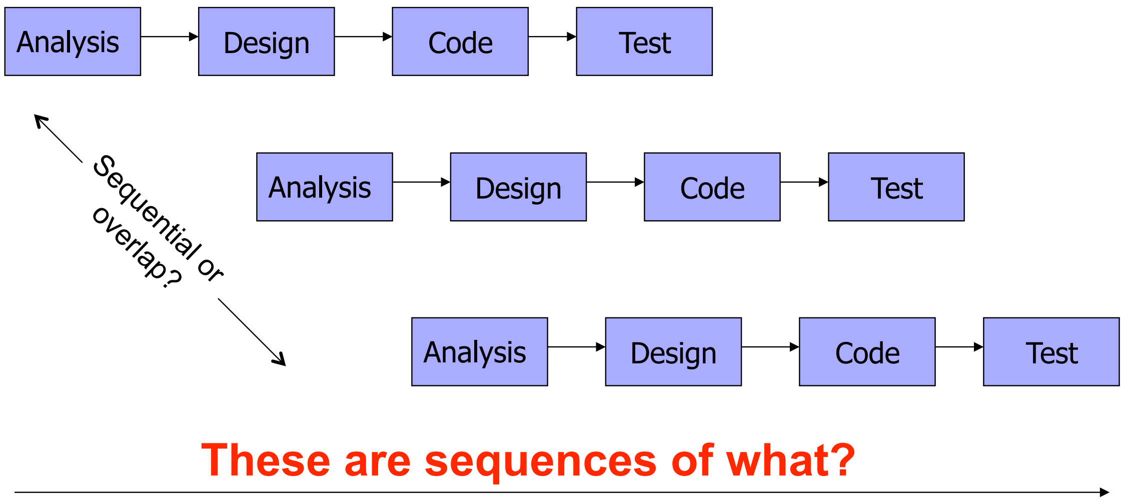
- The incremental model prescribes *developing* and *delivering* the product in *planned* increments.
  - ☐ The product is *designed* to be *delivered* in *increments*.
  - ☐ Each increments provides (in theory) more functionality than the previous increment.
- Reality: Projects called incremental really do increments in Waterfall phases…..

However, it is used:

- Almost all developments…or at least the term
- Anything done in pieces
  - Agile – are these planned in advance
  - No knowing the next step till you do an increment.
- Be very careful to define what you "*mean*" as incremental

Incremental Model

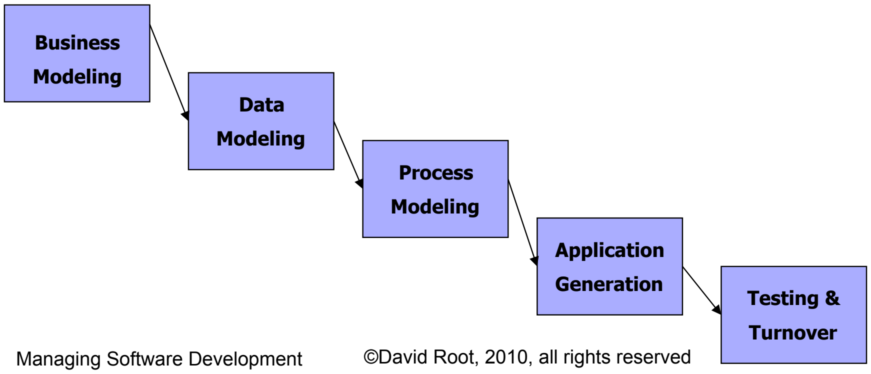
(what "blocks" are missing?)



These are sequences of what?

Rapid Application Development (RAD)

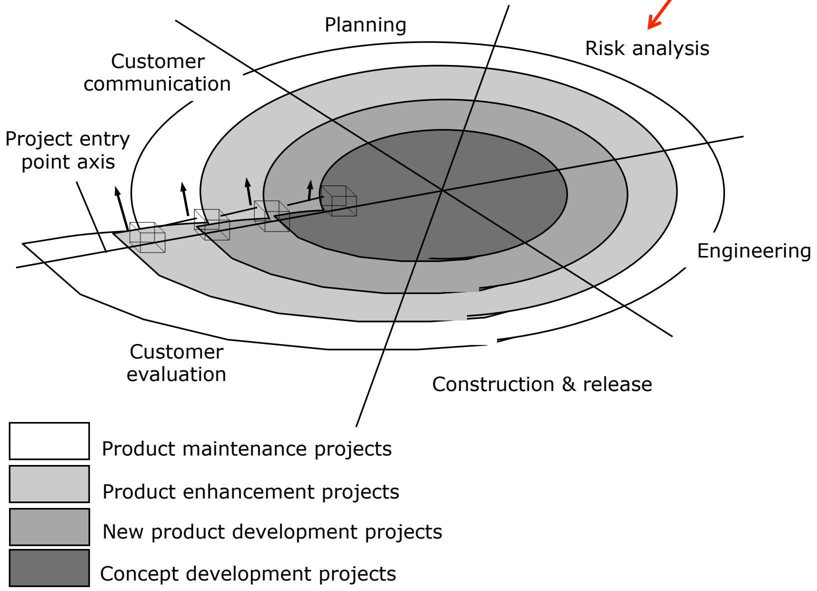
- Incremental
- <u>60-90 days</u> per release
- Information Systems
- 4th Generation Techniques



Spiral Model

- The spiral model
  - First defined by Barry Boehm
  - combines elements of:
    - evolutionary, incremental, and prototyping models
  - First model to explain
    - why iteration matters
    - How iteration could be used effectively
  - the term *spiral* refers to successive iterations outward from a central starting point.

Spiral Model



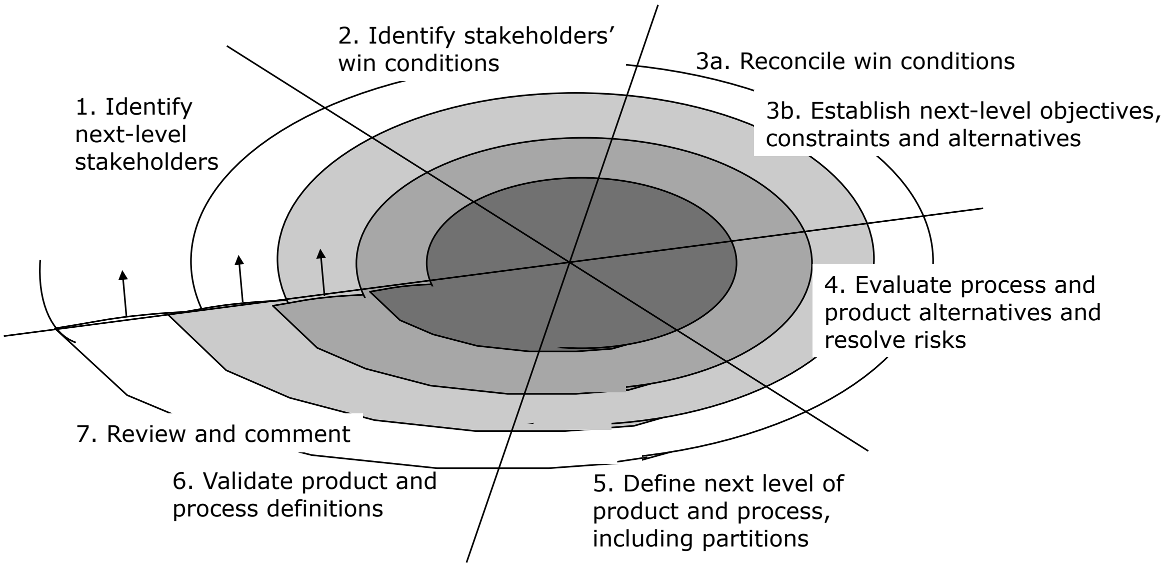
Note

Roger S. Pressman's "Software Engineering, a Practitioners Approach"

Spiral Model

- The goal is to
  - identify risk
  - focus on it early.
- In theory, risk is reduced in outer spirals a the product becomes more refined.
- Each spiral
  - starts with design goals
  - ends with the client reviewing the progress thus far and future direction
  - was originally prescribed to last up to 2 years

WINWIN Spiral



2. Identify stakeholders' win conditions

3a. Reconcile win conditions

1. Identify next-level stakeholders

3b. Establish next-level objectives, constraints and alternatives

4. Evaluate process and product alternatives and resolve risks

7. Review and comment

6. Validate product and process definitions

5. Define next level of product and process, including partitions

Roger S. Pressman's Software Engineering, a Practitioners Approach

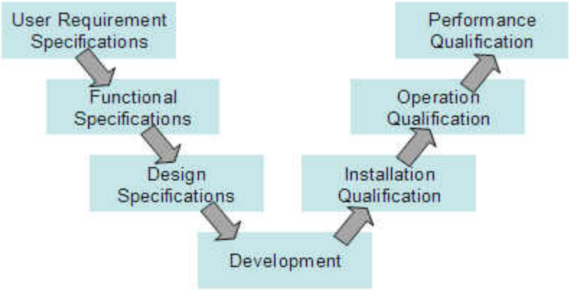
---

V Model

- Often used in system engineering environments to represent the system development lifecycle.
  - summarizes the main steps taken to build *systems not specifically software*
  - describes appropriate deliverables corresponding with each step in the model.

V Model…

- The left side of the V represents the specification stream where the system specifications are defined.
- The right side of the V represents the testing stream where the systems is being tested against the specifications defined on the left side.
- The bottom of the V where the tails meet, represents the development stream.



User Requirement Specifications

Functional Specifications

Design Specifications

Development

Installation Qualification

Operation Qualification

Performance Qualification

Chaos Model

- Extends the spiral and waterfall model defined by L.B.S. Raccoon.
  - espouses the notion that the lifecycle must address all levels of a project, from the larger system to the individual lines of code
  - The whole project, system, modules, functions and each line of code must by defined, implemented, and integrated holistically.

Chaos Model…

- Chaos Theory underlies the fundamental concepts of the Chaos Model including:
  - Software projects are non-linear systems exhibiting random motion (linear systems are rare in nature)
  - Non-linear systems can be more than the sum of their parts.
    - To characterize the behavior of a non-linear system one needs principles to study the system as a whole and not just its parts in isolation (i.e. it is senseless to study architecture design in isolation).

Chaos Model

- *Chaos strategy* resembles the way that programmers work toward the end of a project:
  - when they have a list of bugs to fix and features to create
  - usually someone prioritizes the remaining tasks
  - programmers fix them one at a time
- Chaos strategy states that this is the only valid way to do the work.

Chaos Model

- Key points of *chaos strategy* include
  - ☐ *Issues* are incomplete programming tasks.
  - ☐ *Resolving an issue* means to bring it to *stability*.
    - Resolve the most important issues first.
    - The *most important issues* will be a combination of *big*, *urgent*, and *robust*, where
      - ☐ *Big issues* provide value to users as working functionality.
      - ☐ *Urgent issues* are time sensitive and would otherwise hold up other work if not completed sooner rather than later.
      - ☐ *Robust issues* are trusted and tested.
  - ☐ Work and schedules are derived from *big, urgent,* and *robust issues*.

©David Root, 2010, all rights reserved

Others…

Presenter: David Root
(Material developed with Tony Lattanze)

©David Root, 2010, all rights reserved

Components

- COTS
- Cycle
    - Identify Possible ones
    - Check Library
    - Use (if they exist)
    - Build new ones (if they don't
    - Put new ones in Library
- Problems with COTS?

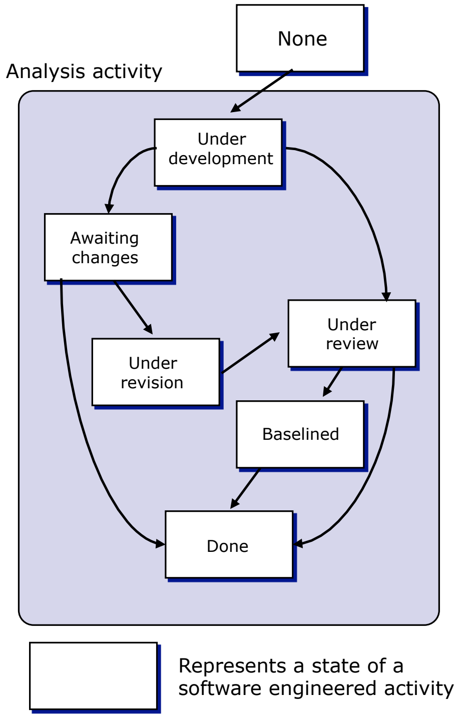
SEI process models for COTS

- PECA
    - Plan the evaluation – stakeholders, goals, constraints, timeframe
    - Establish criteria – measurable, not abstract
    - Collect data based on criteria
    - Analyze – careful of first fit compared to best fit
- Cure
    - COTS Usage Risk Evaluation

Concurrent

- Complementary applications
  - High Interdependence with Modules
- State Charts
- Triggers for transition
- Examples
  - Client – Server
  - OBUS

Concurrent Development Model



Roger S. Pressman’s Software Engineering, a Practioners Approach

Are these different?

- Different names for traditional?
- Does it matter?
- What do you as project managers need to take away from this?

Current State of the Art

- Iterative, cyclic development (or so stated)
- Agile Processes?
- Software is grown rather than birthed whole
- Short cycles
- Small teams
- Component development
- More integration vice new development?

When looking at a new project

DO NOT make your project fit a SDLC!!!

- INSTEAD, find the right SDLC and tailor it to your project (if it can be).
- Your organization may drive this
  - But any lifecycle, process should be seen as a tool to assist development, not an end in and of it self.

Summary

- Need to define & understand SDLC’s
- Variables / criteria that impact selection
  - Resources, time, scope & quality
- Advantages/disadvantages of each

Questions

Presenter: David Root

(Material developed with Tony Lattanze)

©David Root, 2010, all rights reserved