Software Process Models

Andy Podgurski

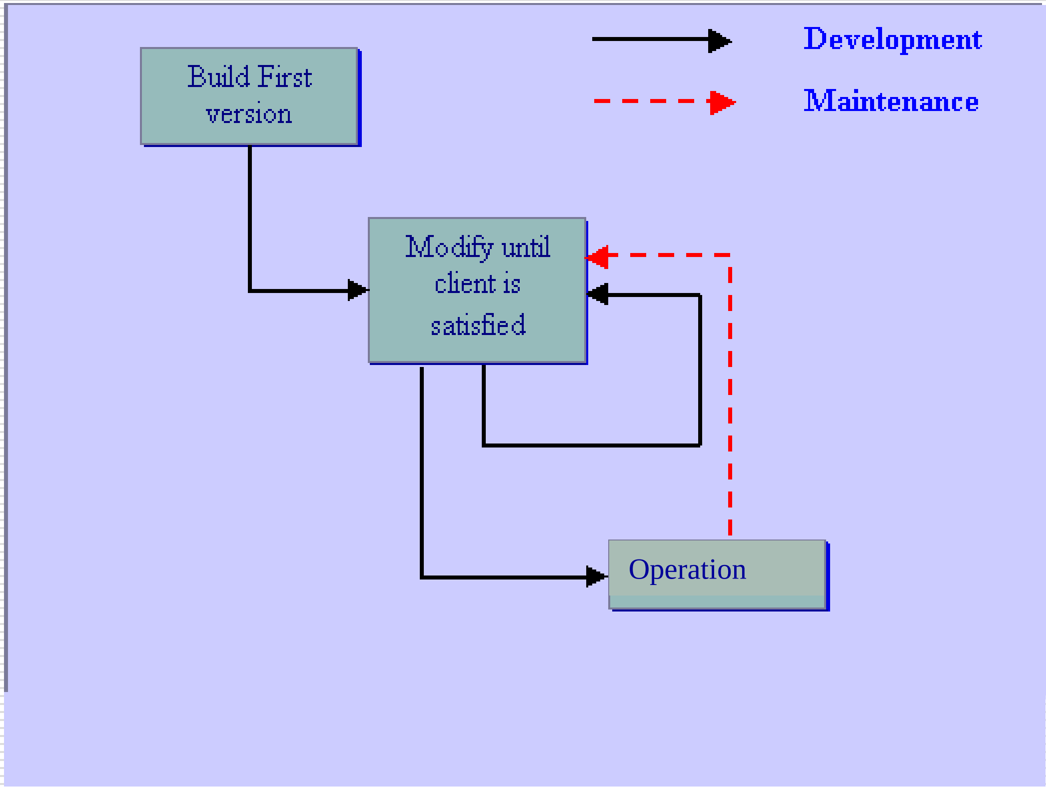
Electrical Engineering & Computer Science Dept.

Case Western Reserve University

Software Process Models

- Graphical models of the software development process
- Characterize *workflow*
- Have *descriptive* and *prescriptive* uses

Anti-model: Build & Fix



From *Object-Oriented and Classical Software Engineering* by Steven Schach

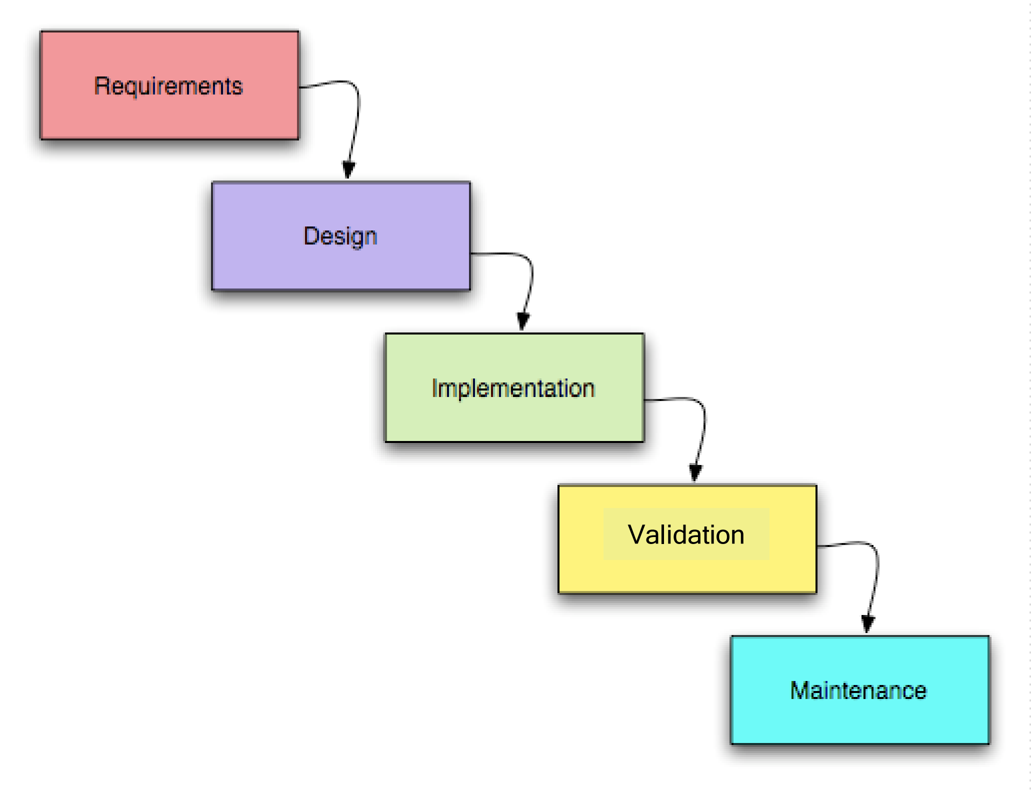
Build & Fix Model (2)

- Product is implemented without specification or design documentation
- It is *reworked* repeatedly until client is satisfied
- B&F works poorly for large products
- Maintenance is likely to be very difficult
  - *Why?*

Question

- How would you improve upon the Build & Fix model?

Basic Waterfall Model



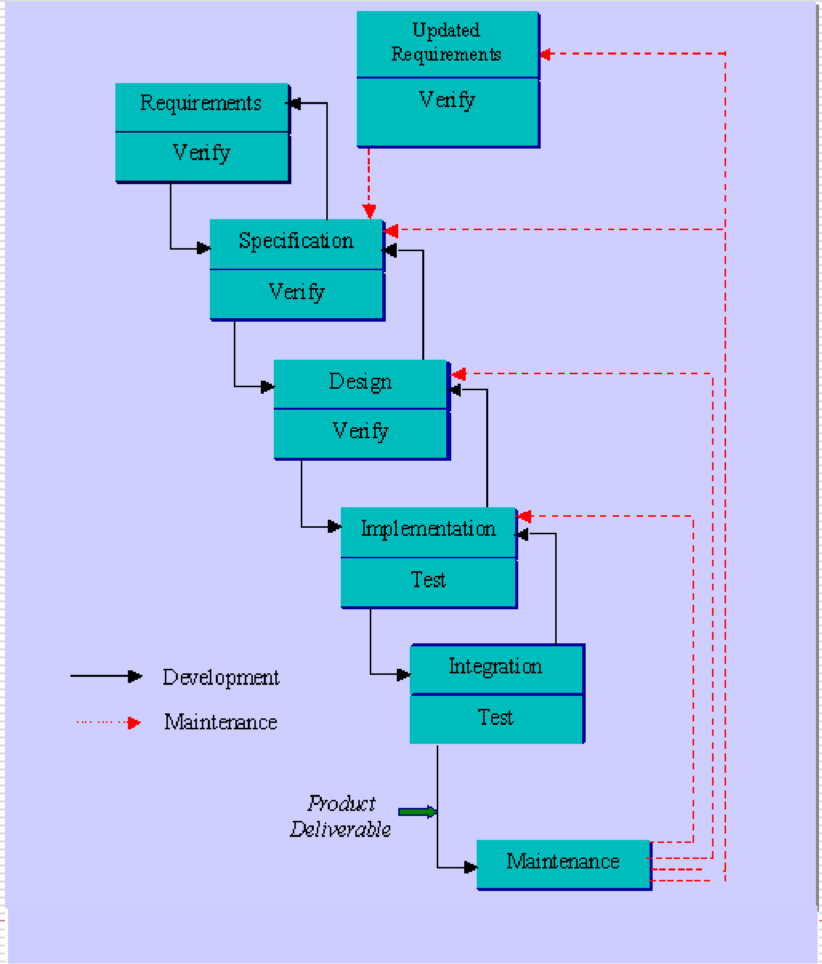
en.wikipedia.org/wiki/Image:Waterfall_model.png (modified)

Phases of Software Development

- **Requirements elicitation and analysis**: determine *what* the software should do
- **Requirements specification**: produce a *written specification* of what the software should do
- **Design**: abstractly describe the *structure and behavior* of a software system satisfying the requirements
- **Implementation**: *program* the components of the system
- **Integration**: *combine* the components into a working system
- **Deployment**: *deploy* the software to end users
- **Operations and maintenance**: *installation*, *support*, *repair*, *enhancement*, *adaptation*, and *evolution* of complete systems

Note **validation** should be associated with each phase.

Waterfall Model Augmented with Iteration



Note addition of *Integration* phase

From *Object-Oriented and Classical Software Engineering* by Steven Schach

Advantages of Waterfall Model

- *Disciplined* approach to development
- Careful *analysis and documentation* before coding can prevent costly problems later
- Documentation produced facilitates maintenance and training
  - Must be kept *up-to-date*

Question?

- Do you see any problems with the Waterfall Model?

Drawbacks of Waterfall Model

- It is difficult to convey *dynamic* appearance and behavior in a document
- Customers often know what they want and don't want *only when they see it*
- Requirements often *change* for other reasons as well
- Developers often *understand* the issues of one phase better during *later* phases
- It is difficult to *assess progress* until some things are implemented

Question

- How would you improve upon the Waterfall Model?

Incentive Mismatch

- Schrage* claims requirements create *perverse incentives* for clients to:
  - Avoid rigorous thinking about cost, change, priorities, and risk
  - Delegate hard design decisions to IT
- It is *inexpensive* for clients to generate many requirements.
- Developers are rewarded for *building to requirements*.
- They're not rewarded for *refining and removing* requirements.
- Shrage argues for *quick prototypes* based on *few* (20-25) requirements.
  - "Never go to a client meeting without a prototype."
  - This fosters *ongoing client interaction* in development.
  - Clients are also *less likely to reject their own work*.

[*IEEE Software*, 2004]

Prototyping

- Prototype is *incomplete model* of eventual system
  - Developed *rapidly* based on initial requirements
  - Provided to *users* for evaluation
- Aids *refinement* and *validation* of requirements
  - Especially helpful with *look-and-feel* and *user interactions*
  - Can also be used to validate an internal design
    - E.g., to assess performance or capacity

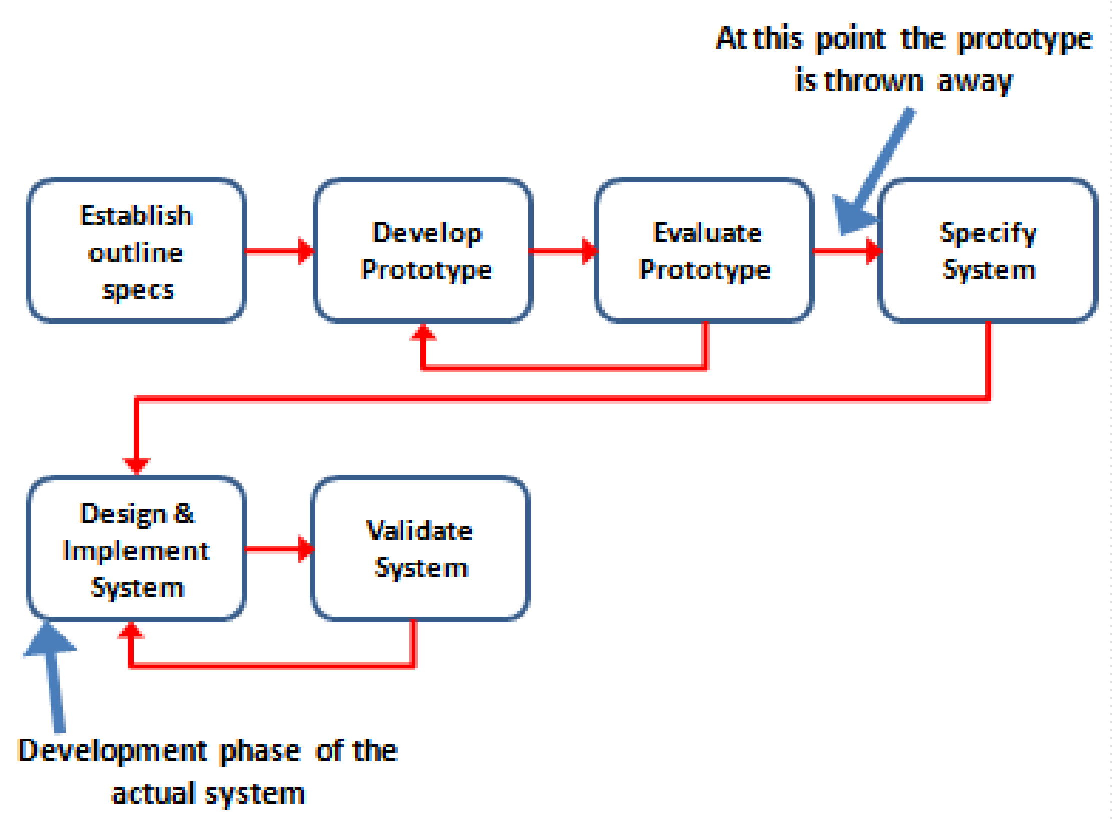
Prototyping cont.

- Focus should be on areas of *greatest risk* to project
- To produce prototype rapidly:
  - Functionality can be *omitted*
  - Non-functional constraints can be *ignored* (e.g., efficiency)
  - Existing components can be *reused*
  - "Rapid development" tools and languages can be exploited

Types of Prototyping

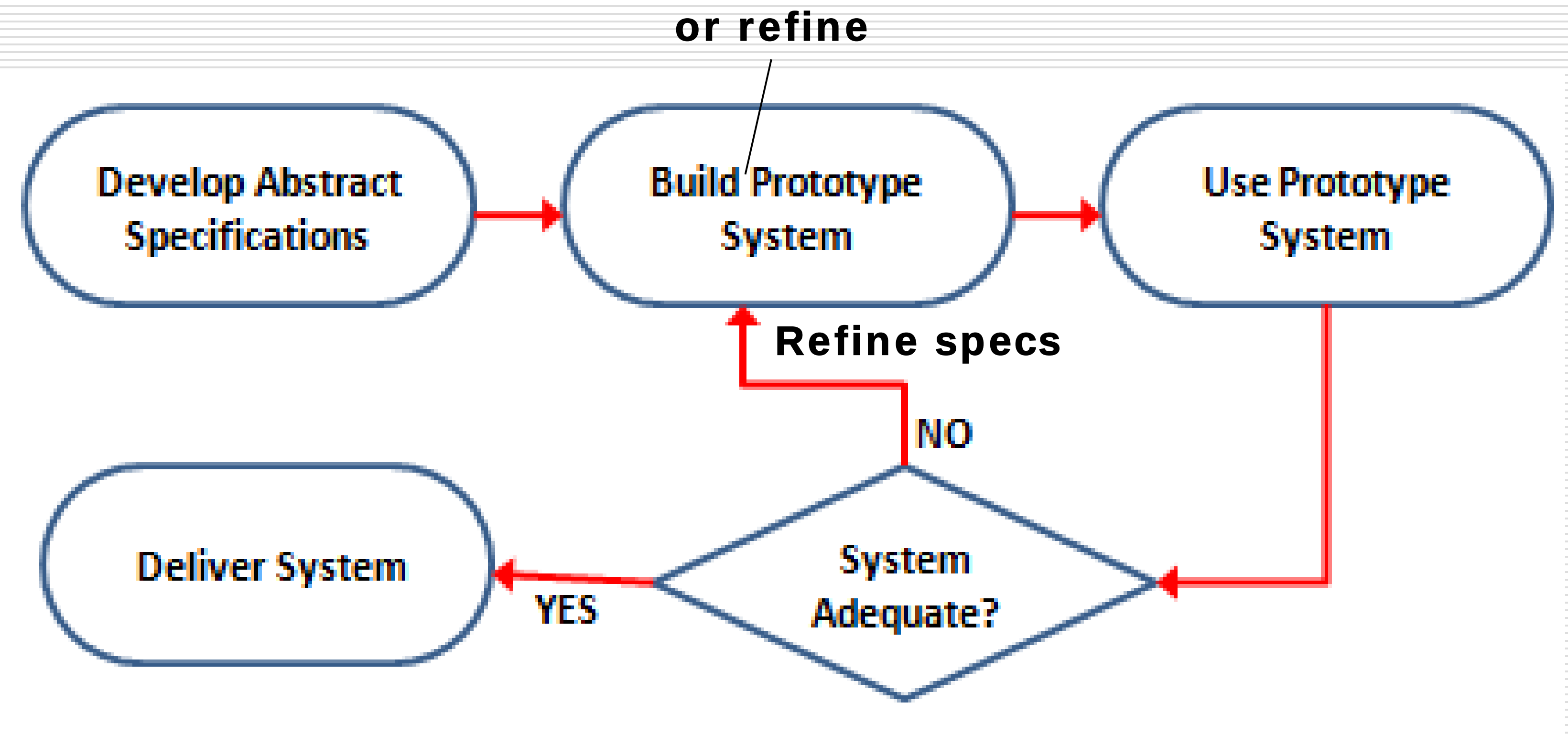
- **Throwaway** – prototype is not built upon
- **Evolutionary** – prototype is iteratively *refined and extended* to obtain final system
  - *Refactoring* (restructuring) is necessary to make and keep design coherent
    - See www.cs.unc.edu/~stotts/723/refactor/chap1.html

Throwaway Prototyping



mahaerss.wordpress.com/about/2-software-engineering-methods/

Evolutionary Prototyping



From *Software Engineering* by Ian Sommerville, 2000

Comparison of Prototyping and Conventional Development

Characteristics	Throwaway prototyping	Evolutionary prototyping	Conventional development
Development approach	Quick and dirty; sloppy	Rigorous; not sloppy	Rigorous; not sloppy
What is built	Poorly understood parts	Well-understood parts first	Entire system
Design drivers	Development time	Ability to modify easily	Depends on project
Goal	Clarify poorly understood requirements and then throw away	Uncover unknown requirements and then evolve	Satisfy all requirements

Alan Davis, IEEE Software, Sept. 1992.

Operational Prototyping [Davis, 1992]

- This *combines throwaway and evolutionary prototyping* to achieve rapid results with stability.
- An evolutionary prototype is constructed and made into a *baseline* using conventional methods
  - Only *well-understood requirements* are implemented in the baseline.
- Copies of the baseline are sent to *multiple customer sites* along with a *trained prototyper*.
  - At each site, the prototyper *observes* use of the system.
  - He/she logs problems and feature requests.

Operational Prototyping cont.

- After the observation period, the prototyper *constructs a throwaway prototype on top of the baseline system*.
- The user now uses and evaluates the new system.
- If new changes aren't effective, the prototyper *removes* them.
- If the user likes the changes, the prototyper sends *change requests* to the development team.
- Based on the change requests from all the sites, the development team produces a *new evolutionary prototype* using conventional methods.

Risks of Prototyping

- Possible neglect of up-front analysis
- Users may misunderstand purpose of prototype
- Accommodating users may lead to “feature creep”
- Excessive effort may be required
- Possible contractual difficulties

Questions:

- How does the Internet facilitate prototyping?
- Do you see any other potential drawbacks to prototyping?

A/B Testing

- An is an *experimental comparison* of 2 versions of a webpage or app to determine which is better
  - Objective evaluation criteria must be specified
- The versions are assigned *at random* to different users, persistently
- The users' interactions with the site are *monitored* and *key metrics* computed
- Uses techniques for *design and analysis of experiments*

Example: Evaluating Possible Bing Feature [Kohavi & Longbotham]

- Feature allows advertisers to provide links to the target site
  - Criterion: increasing average revenue without degrading user engagement

Figure 1: Ads with site link experiment. Treatment (bottom) has site links. The difference might not be obvious at first but it is worth tens of millions of dollars

Structure of Online Experiment
[Kohavi & Longbotham]

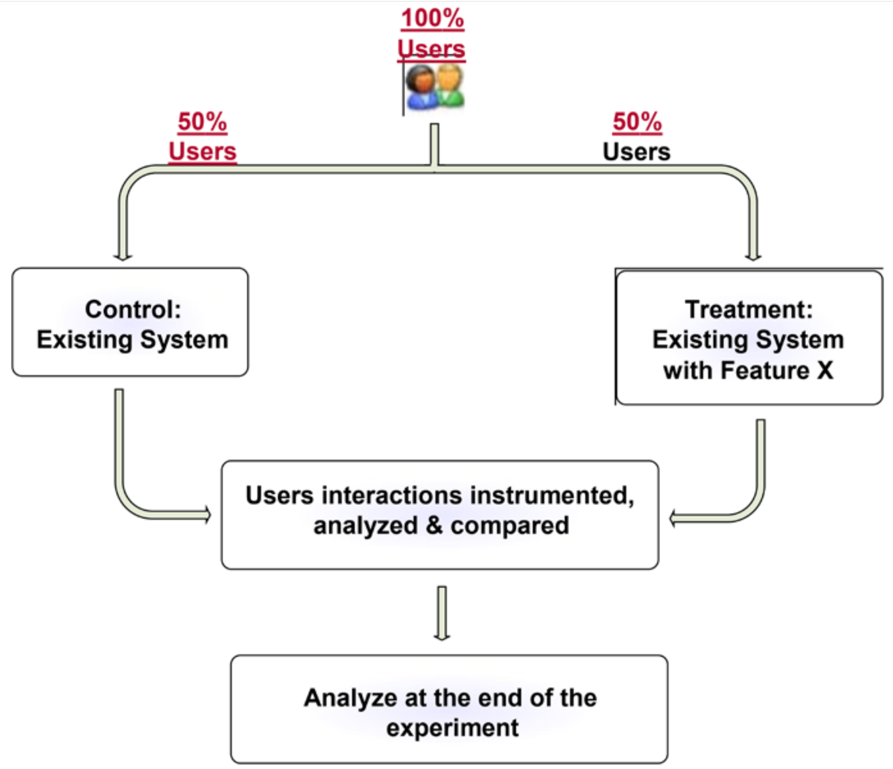


Figure 1 High-level structure of an online experiment

https://link.springer.com/referenceworkentry/10.1007%2F978-1-4899-7687-1_891

Opportunistic Programming [Brandt, CHI 2009]

- Programmers *"prototype, ideate, and discover"*
- Emphasizes *speed and ease* of development
- Involves *web foraging* and *just-in-time learning* for
  - Ideas, examples APIs, code, technical details, etc.

Characteristics of Opportunistic Programming [Brandt, WEUSE 08]

- Build from scratch using *high-level tools*
- Add new functionality via *copy-and-paste*
- Iterate *rapidly*
- Consider code *impermanent*
- Face unique *debugging challenges*

Question

- Do you see any potential problems with opportunistic programming?

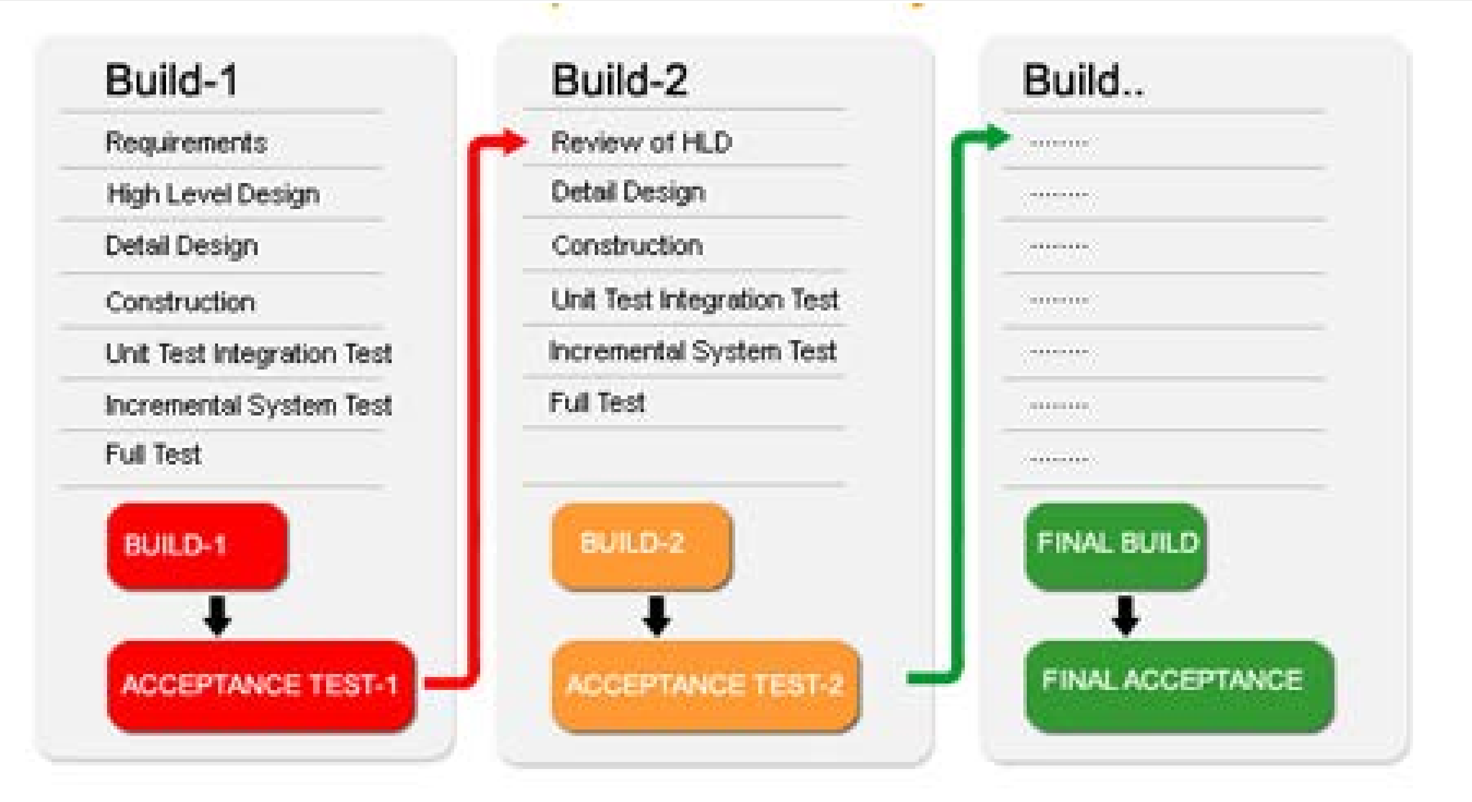
Opportunistic Programming Issues

- Rights to intellectual property
- Plagiarism
  - *Not permitted for EECS 393/493 projects!*
- Reliability, security, maintainability

Incremental Delivery

- System is developed and delivered in series of *increments* or *builds*
- Each increment provides a *subset* of the system functionality
- Services are allocate to increments based on *customer's priorities*
  - High risk features delivered early if needed by customer
- A *conventional* development process is applied to *each increment*

Incremental Delivery (2)



www.kirantechnologies.in/kt_files/images/methologies

Advantages of Incremental Delivery

- Client can *exploit* product functionality sooner
- Client can *adapt* to product gradually
- Developer gets earlier *feedback* than with waterfall model
- Requires *planning* for future enhancements
- Highest priority services get *most testing*

Question

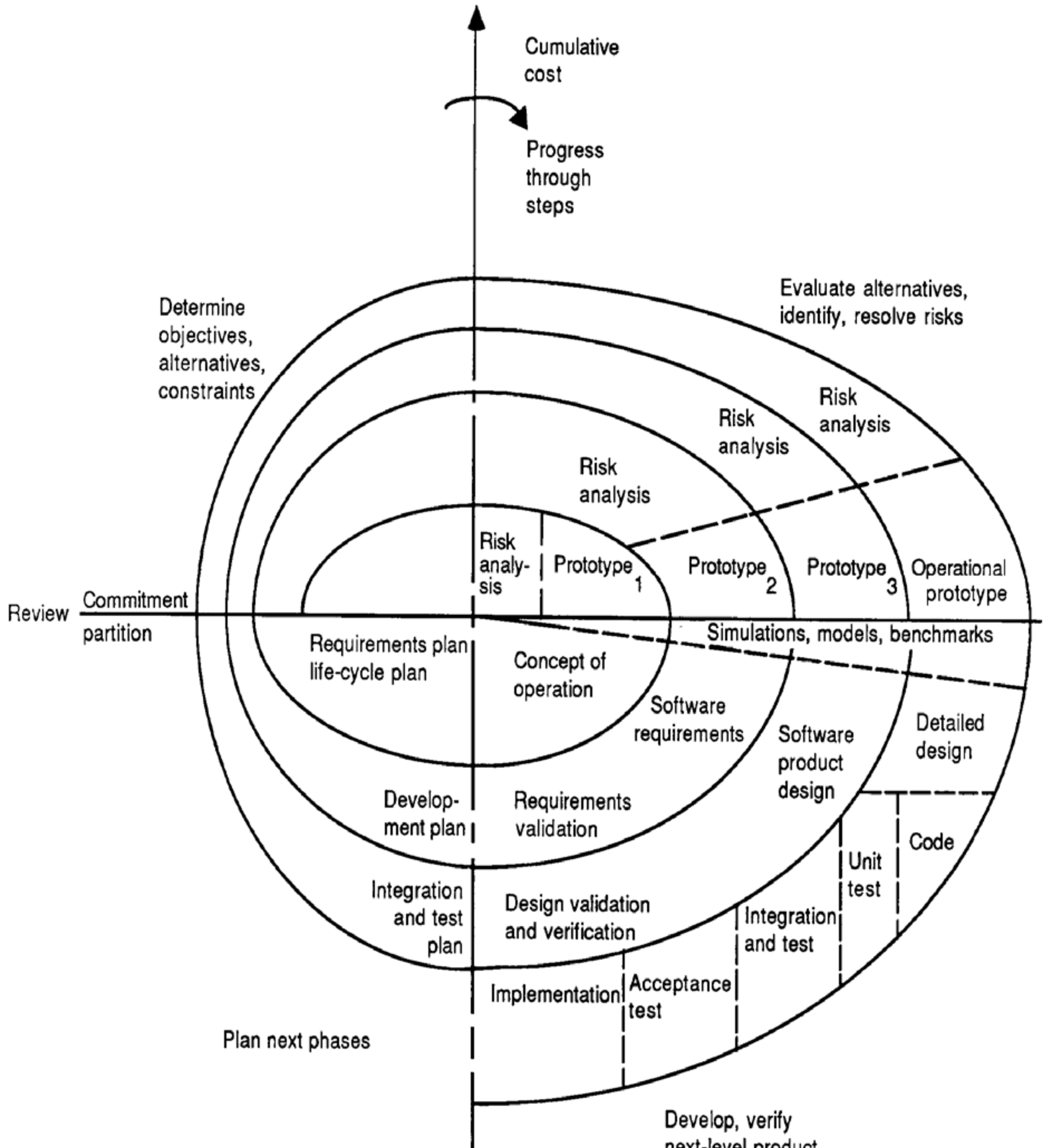
- Do you see any potential problems with incremental delivery?

Risks of Incremental Delivery

- It may be difficult to *integrate* later builds with early ones
  - Why?
- It can *degrade* into build-and-fix

Spiral Model [Boehm 1986]

- Assumes that *risk management* is a paramount issue in software development
- Development process is represented by a *spiral*
- Each cycle represents a phase with four parts:
  1. *Setting objectives*
  2. *Risk analysis and mitigation*
  3. *Development and validation*
  4. *Planning for next phase*



Boehm, IEEE Computer, 1988

Table 1. Spiral model usage: TRW Software Productivity System, Round 0.

Objectives	Significantly increase software productivity
Constraints	At reasonable cost Within context of TRW culture • Government contracts, high tech., people oriented, security
Alternatives	Management: Project organization, policies, planning, control Personnel: Staffing, incentives, training Technology: Tools, workstations, methods, reuse Facilities: Offices, communications
Risks	May be no high-leverage improvements Improvements may violate constraints
Risk resolution	Internal surveys Analyze cost model Analyze exceptional projects Literature search
Risk resolution results	Some alternatives infeasible • Single time-sharing system: Security Mix of alternatives can produce significant gains • Factor of two in five years Need further study to determine best mix
Plan for next phase	Six-person task force for six months More extensive surveys and analysis • Internal, external, economic Develop concept of operation, economic rationale
Commitment	Fund next phase

Boehm, IEEE
Computer, 1988

Table 2. Spiral model usage: TRW Software Productivity System, Round 1.

Objectives	Double software productivity in five years
Constraints	$10,000 per person investment Within context of TRW culture • Government contracts, high tech., people oriented, security Preference for TRW products
Alternatives	Office: Private/modular/. . . Communication: LAN/star/concentrators/. . . Terminals: Private/shared; smart/dumb Tools: SREM/PSL-PSA/. . .; PDL/SADT/. . . CPU: IBM/DEC/CDC/. . .
Risks	May miss high-leverage options TRW LAN price/performance Workstation cost
Risk resolution	Extensive external surveys, visits TRW LAN benchmarking Workstation price projections
Risk resolution results	Operations concept: Private offices, TRW LAN, personal terminals, VAX Begin with primarily dumb terminals; experiment with smart workstations Defer operating system, tools selection
Plan for next phase	Partition effort into software development environment (SDE), facilities, management Develop first-cut, prototype SDE • Design-to-cost: 15-person team for one year Plan for external usage
Commitment	Develop prototype SDE Commit an upcoming project to use SDE Commit the SDE to support the project Form representative steering group

Boehm, IEEE Computer, 1988

Table 3. Spiral model usage: TRW Software Productivity System, Round 2.

Objectives	User-friendly system Integrated software, office-automation tools Support all project personnel Support all life-cycle phases
Constraints	Customer-deliverable SDE ⇒ Portability Stable, reliable service
Alternatives	OS: VMS/AT&T Unix/Berkeley Unix/ISC Host-target/fully portable tool set Workstations: Zenith/LSI-11/. . .
Risks	Mismatch to user-project needs, priorities User-unfriendly system • 12-language syndrome; experts-only Unix performance, support Workstation/mainframe compatibility
Risk resolution	User-project surveys, requirements participation Survey of Unix-using organizations Workstation study
Risk resolution results	Top-level requirements specification Host-target with Unix host Unix-based workstations Build user-friendly front end for Unix Initial focus on tools to support early phases
Plan for next phase	Overall development plan • for tools: SREM, RTT, PDL, office automation tools • for front end: Support tools • for LAN: Equipment, facilities
Commitment	Proceed with plans

Boehm, IEEE Computer, 1988

Risk item	Risk management techniques
1. Personnel shortfalls	Staffing with top talent, job matching; teambuilding; morale building; cross-training; pre-scheduling key people
2. Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)
5. Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
10. Straining computer-science capabilities	Technical analysis; cost-benefit analysis; prototyping; reference checking

Boehm, IEEE Computer, 1988

Analysis of Spiral Model

- Best suited to projects with:
  - Large scale
  - High risk
  - Ample resources and time
  - Client and developers within same organization

Evolution of the Spiral Model

- Boehm later described it as a risk-based "process model generator"
  - Other models are *special cases* that fit the *risk patterns* of certain projects
- Recent revision: Incremental Commitment Spiral Model

Incremental Commitment Spiral Model [2014]