Lecture Notes for Lecture 2 of CS 5500 (Foundations of Software Engineering) for the Spring 2021 session at the Northeastern University San Francisco Bay Area Campuses.

Models of Software Engineering Processes

Philip Gust,
Clinical Instructor
Department of Computer Science

http://www.ccis.northeastern.edu/home/pgust/classes/cs5500/2021/Spring/index.html

Models of Software Engineering Processes

Review of Lecture 1

- In lecture 1, we began studying the foundations of software engineering to motivate the individual topics that we will discuss in more depth during the course.
- We discovered that software as an rigorous engineering discipline did not exist for at least half of the time during which people have be developing software.
- One of the biggest factors that drove software engineering as a discipline is the increasing complexity of software being developed, in response to growing customer expectations.

Models of Software Engineering Processes

Review of Lecture 1

- We learned that the base of software engineering is a *process framework* that rests on a bedrock of quality, leading to more effective approaches.
- A process framework with a small set of *generic framework activities*, and a set of *umbrella activities* applied throughout a process can be adapted to meet any specific requirements.
- Finally, a set of software engineering principles helps guide the development and application of specific software engineering processes.

Models of Software Engineering Processes

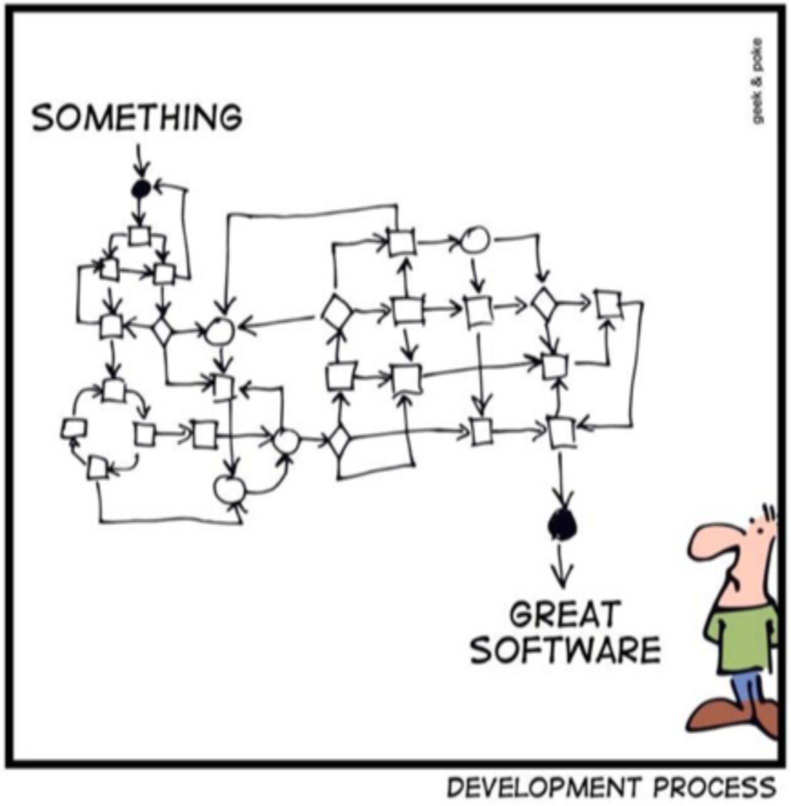
Introduction

- In this lecture we will begin looking at specific types of software engineering process models that have been developed to guide product development.
- For each process model, we will seek to
  - understand the underlying principles of the model
  - look at how it fits in to the generic software process framework
  - see how the process works
  - consider its relative strengths and weaknesses

Models of Software Engineering Processes

Key activities of a software process

- The generic software engineering process that we studied in the previous lecture provides a framework for specific process models.
- Key generic activities include:
    - Communication
    - Planning
    - Modeling
    - Construction
    - Deployment



Models of Software Engineering Processes

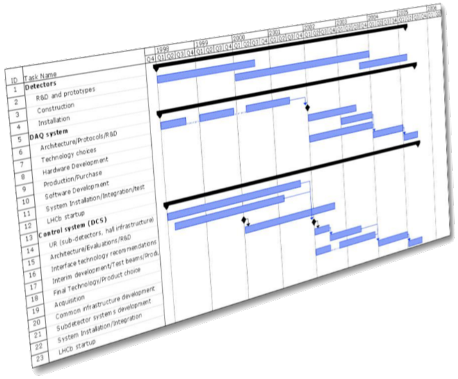
Key activities of a software process

- Communication (inception/feasibility)
  - Before any technical work commences, communicate and collaborate to understand customer’s and stakeholders’ objectives and gather requirements.
  - Steps include:
    - identify target customers and markets
    - solicit customer and stakeholder input
    - perform SWOT analysis of competition (strengths, weaknesses, opportunity, threats)

Models of Software Engineering Processes

Key activities of a software process

- Planning
  - Create a software project plan that describes the technical tasks, the risks, the resources, the work products, and a work schedule
  - The plan identifies:
    - high-level activities
    - work items
    - schedules
    - resources

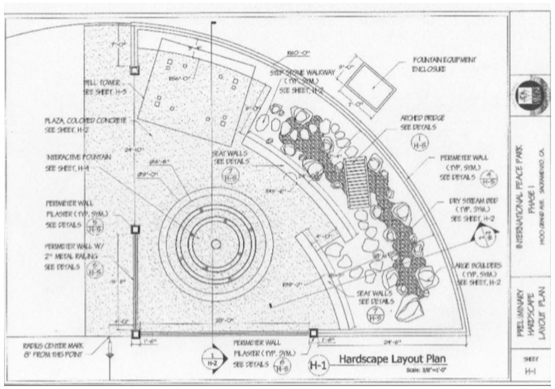


  - The plan is a guideline, not gospel. It may change: maybe a lot!
  - Plan for configuration management now, before any artifacts are created.

Models of Software Engineering Processes

Key activities of a software process

- Modeling (architecture and design)
  - Create a preliminary sketch and refine it in greater detail to understand the requirements and the design.
    - **Architecture** defines the system and components needed in the systems and their interfaces/interactions. It also considers how to achieve the User Experience.
    - **Design** defines how the software will actually be built
  - Outputs include:
    - Software Architecture Document (SAD)
    - Software Design Specification (SDS)


BELL TOWER SEE SHEET H-5
PLAZA, COLORED CONCRETE SEE SHEET H-2
INTERACTIVE FOUNTAIN SEE SHEET H-4
PERIMETER WALL PLASTER (TYP. SYM.) SEE DETAILS
PERIMETER WALL W/ 2" METAL RAILING SEE DETAILS
RADIUS CENTER MARK 8' FROM THIS POINT
STEP STONE WALKWAY (TYP. SYM.) SEE SHEET H-2
FOUNTAIN EQUIPMENT ENCLOSURE
ARCHED BRIDGE SEE DETAILS
SEAT WALLS SEE DETAILS
PERIMETER WALL (TYP. SYM.) SEE DETAILS
DRY STREAM BED (TYP. SYM.) SEE SHEET H-2
LARGE BOULDERS (TYP. SYM.) SEE SHEET H-2
SEAT WALLS SEE DETAILS
PERIMETER WALL PLASTER (TYP. SYM.) SEE DETAILS
H-1 Hardscape Layout Plan
Scale: 3/8"=1'-0"
INTERNATIONAL PEACE PARK PHASE I
PRELIMINARY HARDSCAPE LAYOUT PLAN
SHEET H-1

Models of Software Engineering Processes

Key activities of a software process

- Construction (development and validation)
  - Generate code (either manual or automated) and test to uncover errors in the code.
  - The emphasis is on delivering the design specified in previous activity
  - Validation includes
    - unit testing
    - integration testing
    - system testing
    - user experience testing
  - Output is a tested candidate for deployment.

Models of Software Engineering Processes

Key activities of a software process

- Deployment (delivery and maintenance)
  - Deliver software to the customer, who evaluates the delivered product and provides feedback.
  - Following acceptance, this phase involves updates because:
    - defects are discovered
    - customer requests enhancements
    - improvements are desired (e.g. performance and reliability)
  - Output is a new release of the system.

Models of Software Engineering Processes

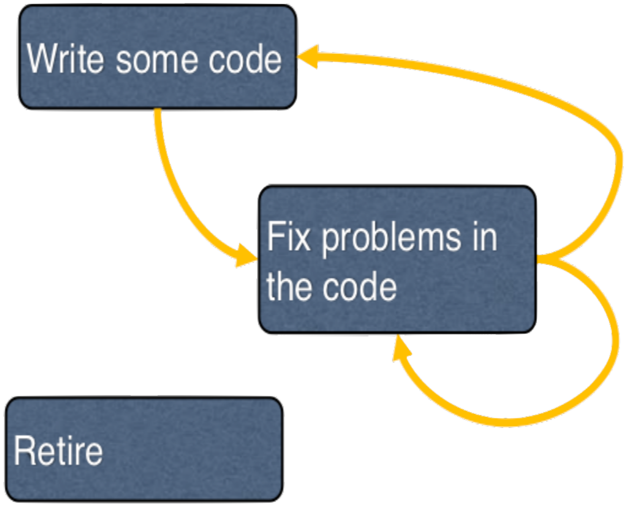
Key activities of a software process

- Summary
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment
- Process models based on a generic process framework differ by the order and frequency of these key framework activities.

Models of Software Engineering Processes

Software process models

- Code and fix (c. 1950s – 1960s)
  - The oldest process model is relatively simple and unstructured, involving writing code, then fixing problems that arise once the code is deployed.
  - The model lacks most of the phases described by the abstract process framework, including:
    - communication
    - planning
    - modeling
    - testing (construction)



Models of Software Engineering Processes

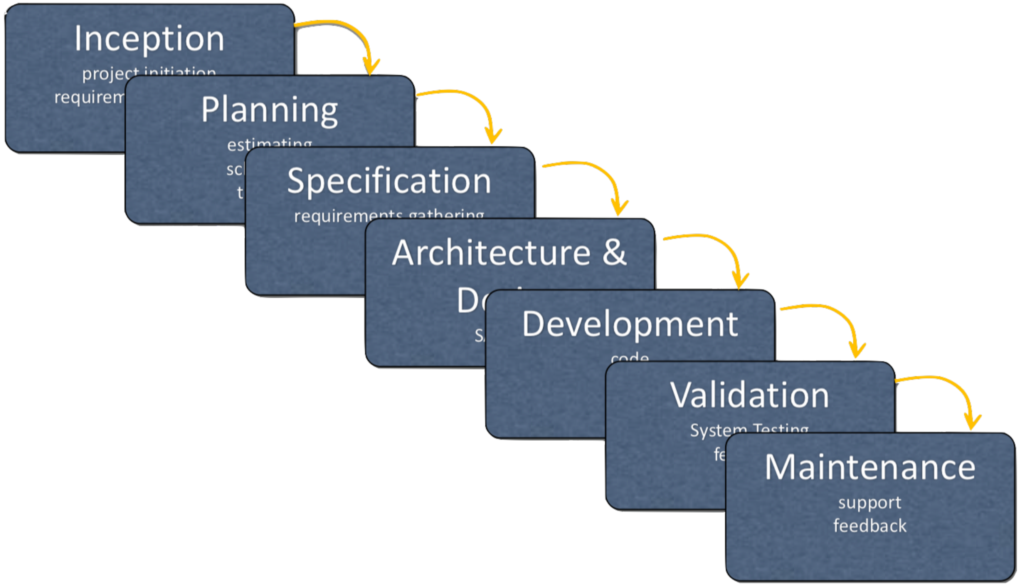
Software process models

- Code and fix (c. 1950s – 1960s)
  - Issues with the model include:
    - No process steps – no specs, docs, tests...
    - No separation of concerns – no teamwork
    - No way to deal with complexity
    - After a number of fixes, the code tends to become so poorly structured that subsequent fixes were very expensive.
    - Often, even well-designed software was a poor match to users' needs that it was either rejected outright or expensively redeveloped.
    - Code was expensive to fix because of poor preparation for testing and modification.

Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
  - The waterfall model organizes the activities of the generic process framework into linear, sequential phases, where each depends on the deliverables of the previous one.



Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
  - The first formal description is often cited as a 1970 article by Wilson W. Royce, director at the Lockheed Software Technology Center in Austin, Texas:
    - "Managing the Development of Large Software Systems"), Proceedings of IEEE WESCON, 26 (August 1970): p. 1–9
  - The article did not use the term “waterfall” but presented the model as an example of a flawed, non-working model.

Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
  - The earliest use of the term "waterfall" may have been in a 1976 paper by Thomas Bell and T.A. Thayer,
    - “Software requirements: Are they really a problem?”. Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press, 1976.

Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
    - In 1985, the United States Department of Defense (mistakenly) captured this approach in DOD-STD-2167A, their standards for working with software development contractors.
    - The standard said: the contractor shall implement a software development cycle that includes the following phases:
        - Preliminary Design
        - Detailed Design
        - Coding and Unit Testing
        - Integration
        - Testing

Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
  - Benefits of the model include:
    - Time spent early in model can reduce costs at later stages.
      - 20%-40% in first 2 phases, 30%-40% coding, rest for testing
    - Medium-to-large projects included detailed procedures and controls to simplify project management and tracking.
    - Strong emphasis on documentation (requirements, design) as well as code.
      - ensures knowledge not lost if team members leave
      - easier to on-board new team members
      - simplifies later maintenance phase

Models of Software Engineering Processes

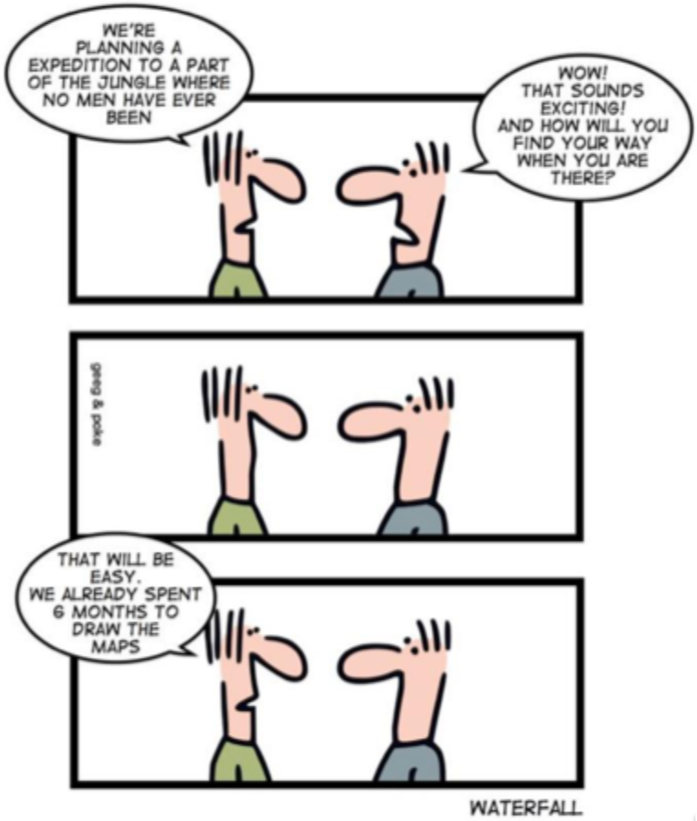
Software process models

- Waterfall model (c. 1970s – 1980s)
    - Issues with the model include:
        - Requirements unknown before seeing working software.
        - Designers not fully aware of future difficulties with new design.
        - Problems discovered late are difficult to fix.
        - Difficult to maintain strict separation between systems analysis and implementation.
            - exposes issues and edge-cases designers did not consider
    - Organizations like U.S. Department of Defense now have a stated preference against waterfall-type methodologies.
        - MIL-STD-498 encourages evolutionary acquisition, and Iterative and Incremental Development.

Models of Software Engineering Processes

Software process models

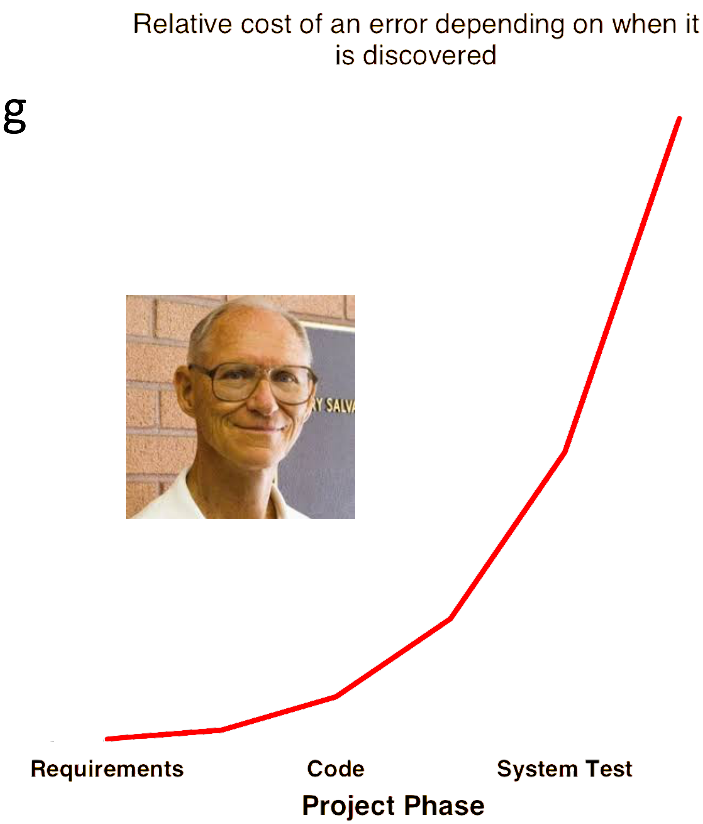
- Waterfall model (c. 1970s – 1980s)


WE'RE PLANNING A EXPEDITION TO A PART OF THE JUNGLE WHERE NO MEN HAVE EVER BEEN
WOW! THAT SOUNDS EXCITING! AND HOW WILL YOU FIND YOUR WAY WHEN YOU ARE THERE?
geek & poke
THAT WILL BE EASY. WE ALREADY SPENT 6 MONTHS TO DRAW THE MAPS
WATERFALL

Models of Software Engineering Processes

Software process models

- Waterfall model (c. 1970s – 1980s)
  - Boehm’s first law and its effect:
    - Errors are most frequent during the requirements and design
    - Correcting errors is more expensive the later they are removed.

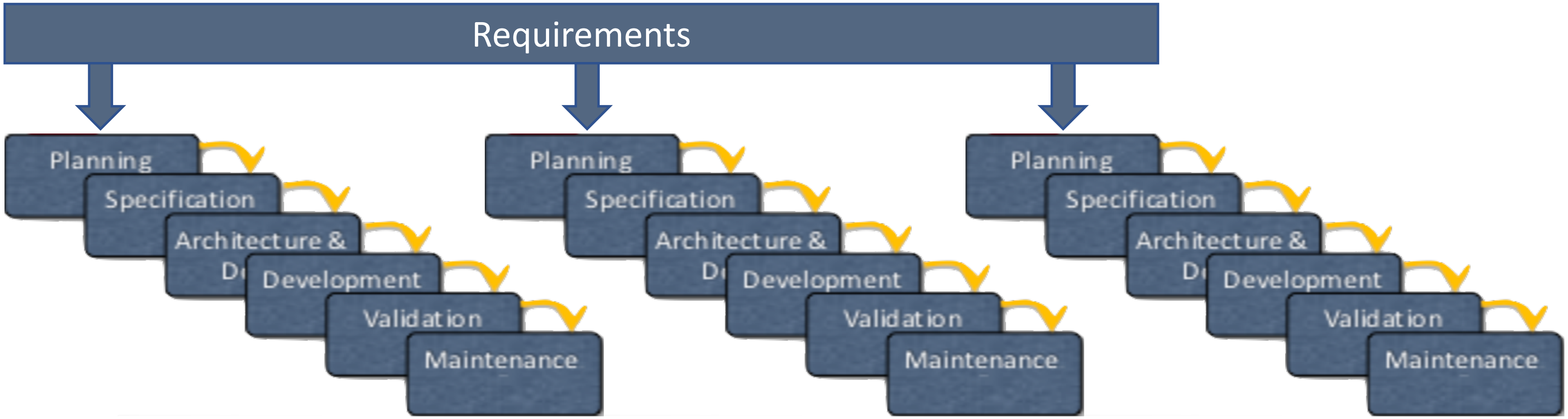


Boehm, Barry W., McClean, R.K., Urfrig, D.B.: “Some Experience with Automated Aids to the Design of Large-Scale Reliable Software.” *IEEE Trans on Software Engineering* 1, 1 (1975), 125–133

Models of Software Engineering Processes

Software process models

- Incremental model (c. 1970s – 1980s)
  - An improvement that Wilson Royce proposed to the original waterfall model is the *incremental model*.
  - The assumption is that requirements can be segmented into an incremental series of products, each developed somewhat independently.



Models of Software Engineering Processes

Software process models

- Incremental model (c. 1970s – 1980s)
  - Additional benefits with the model include:
    - Less cost and time required to make the first delivery
    - Less risk incurred to develop the smaller systems represented by increments
    - User requirement changes may decrease because of the quicker time to first release
    - Incremental funding is allowed; only one or two increments need to be funded when the project starts.

Models of Software Engineering Processes

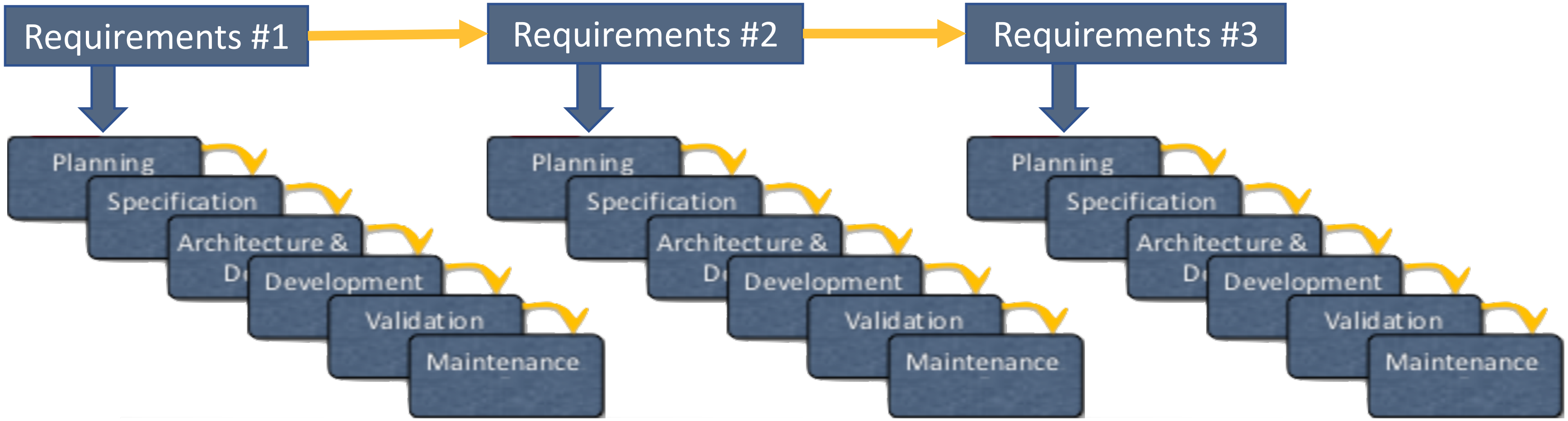
Software process models

- Incremental model (c. 1970s – 1980s)
  - Additional issues with the model include:
    - Fielding initial increments may destabilize later increments through unplanned levels of user change requests.
    - If requirements are not as stable or complete as thought earlier, increments might be withdrawn from service, reworked, and re-released.
    - Managing resulting costs, schedule, and configuration complexity may exceed capabilities of an organization.

Models of Software Engineering Processes

Software process models

- Evolutionary model (c. 1970s – 1980s)
  - The next logical step in life cycle development is the *evolutionary model*, which explicitly extends the incremental model to the requirements phase.
  - The first build increment is used to refine the requirements for a second build increment, and the second for the third.



Models of Software Engineering Processes

Software process models

- Evolutionary model (c. 1970s – 1980s)
  - Benefits with the model include:
    - The model can be used when the requirements cannot or will not be specified.
    - The user can experiment with the system to improve the requirements.
    - Greater customer/stakeholder involvement is required than the waterfall model.

Models of Software Engineering Processes

Software process models

- Evolutionary model (c. 1970s – 1980s)
  - Disadvantages of the model include:
    - Use of the model is exploratory in nature and therefore constitutes a high-risk endeavor; strong management is required.
    - This model is used as an excuse for hacking to avoid documenting the requirements or design even if they are well understood.
    - Customers/stakeholders do not understand the nature of the approach and can be disappointed when the results are unsatisfactory.

Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping (1980s – 1990s)
    - Each unknown is a risk to a project
    - Especially at the project’s beginning, there are many unknown to deal with, such as:
        - How should customers interact with the system (UI/UX)?
        - What features do costumers or stakeholders need?
        - Will this design hit the performance benchmarks?
        - How does some piece of technology work?

Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping (1980s – 1990s)
    - Prototyping is one technique for managing risk by creating a partial implementation to identify and resolve concerns of understanding.
    - Prototyping is useful because it helps avoid rework and possible future roadblocks.

Models of Software Engineering Processes

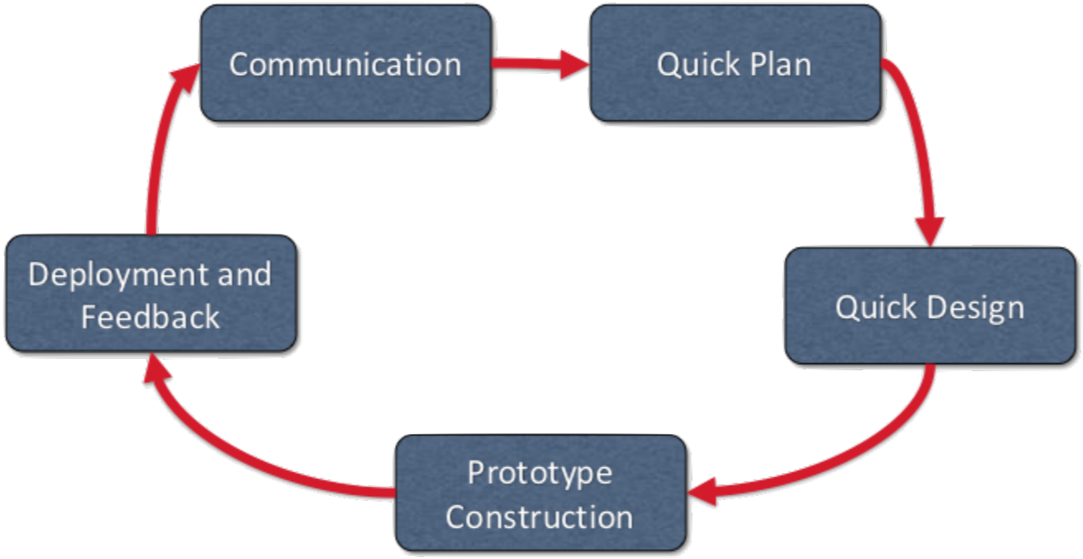
Software process models

- Evolutionary models and prototyping (1980s – 1990s)
    - Boehm’s second law describes the benefit of prototyping:
        - Prototyping significantly reduces requirement and design errors, especially for user interfaces.
            - (Perhaps prototyping as part of the requirement and design phases amounts to spending time thinking about requirements.)
        - Boehm, B.W., Gray, T.E., Seewaldt, T.: “Prototyping Versus Specifying: A Multiproject Experiment.” *IEEE Trans on Software Engineering* 10, 3 (1984), 290–302.

Models of Software Engineering Processes

Software process models

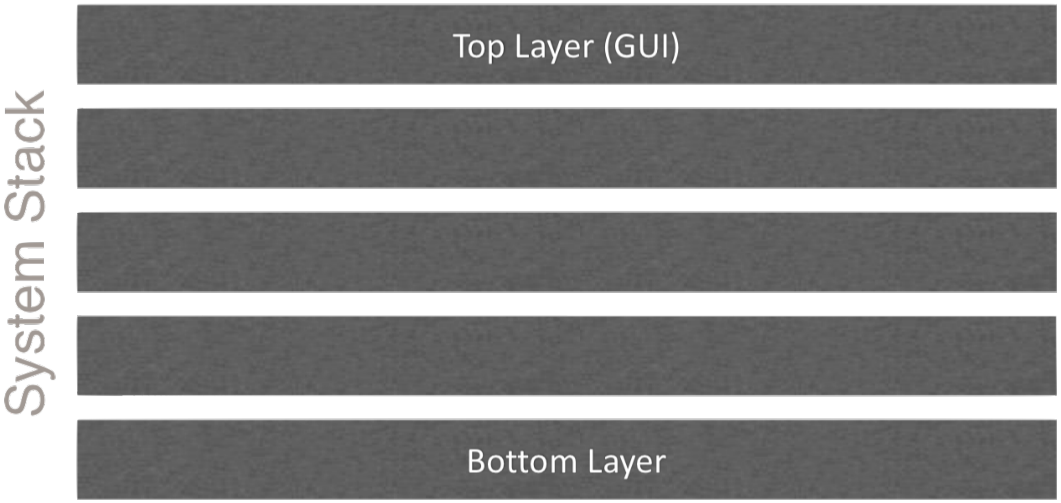
- Evolutionary models and prototyping (1980s – 1990s)
  - An evolutionary model can make use of quick planning, design, and prototyping.



Models of Software Engineering Processes

Software process models

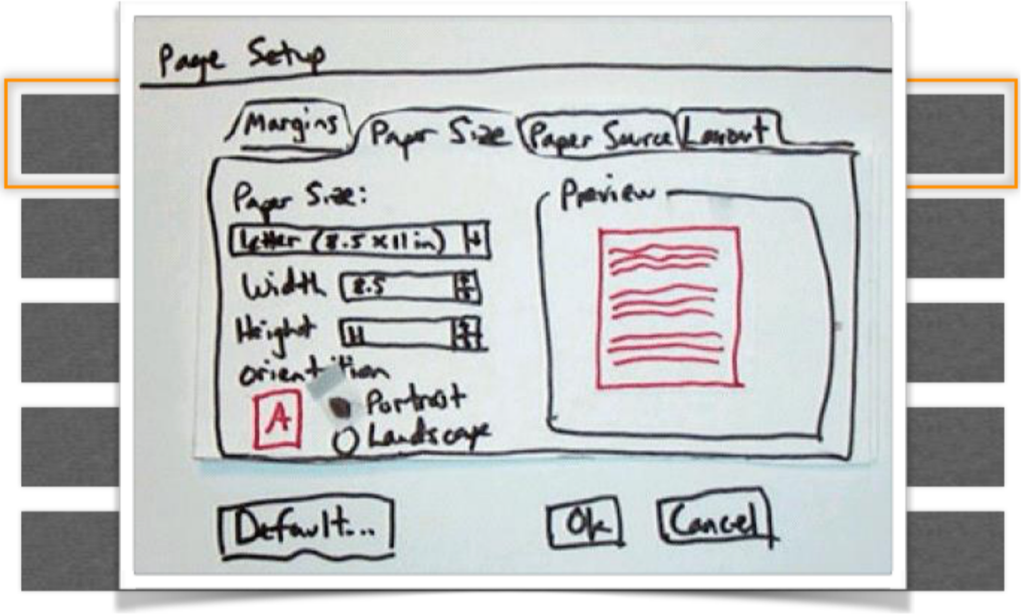
- Evolutionary models and prototyping (1980s – 1990s)
  - Software consist of stacked layers of software, from a bottom layer for the back-end, to the top layer GUI.
  - A prototype can cut the stack horizontally or vertically.



Models of Software Engineering Processes

Software process models

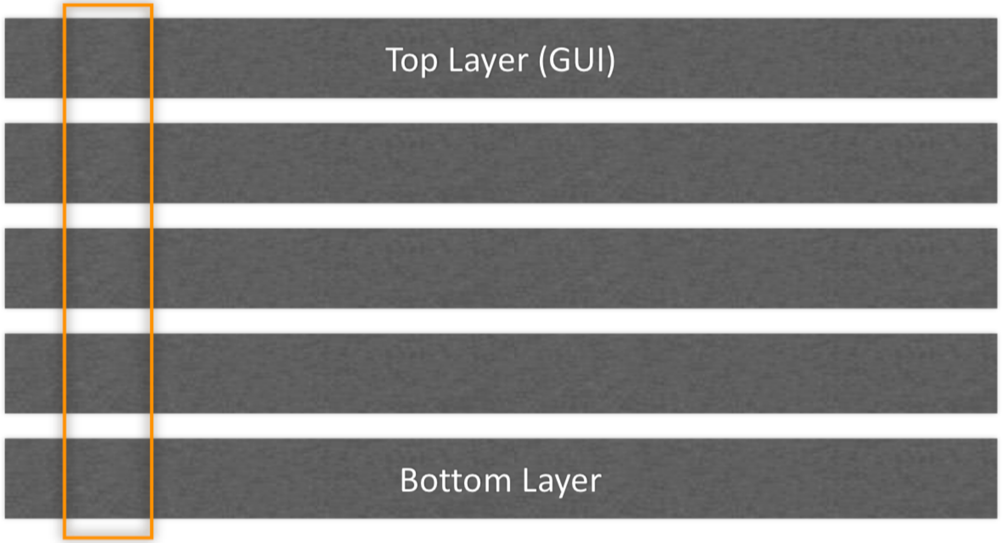
- Evolutionary models and prototyping (1980s – 1990s)
  - A horizontal prototype focuses on just one application layers. Here is a paper prototype of just the GUI layer.



Models of Software Engineering Processes

Software process models

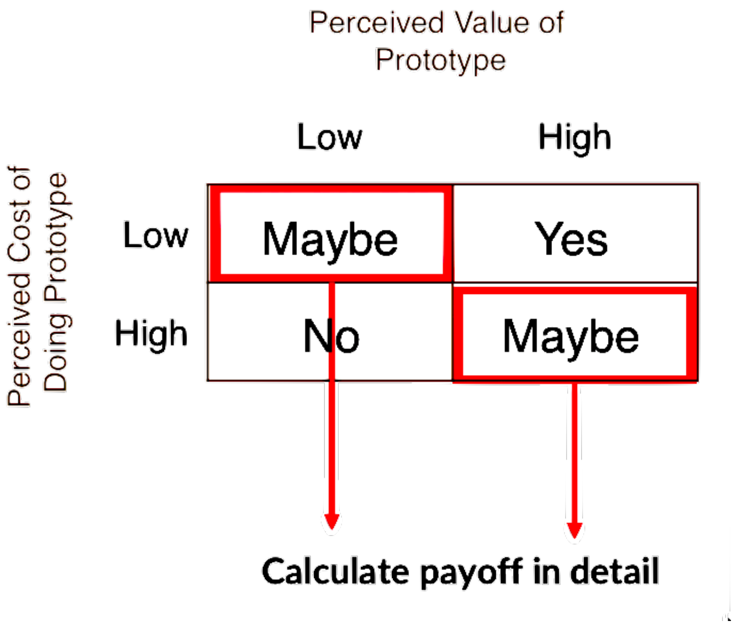
- Evolutionary models and prototyping (1980s – 1990s)
  - A vertical prototype focuses on a limited subset of functionality across all the application layers.



Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping (1980s – 1990s)
  - Is it worthwhile building a prototype? It depends on the payoff for the expected cost and the perceived value.



Perceived Value of Prototype

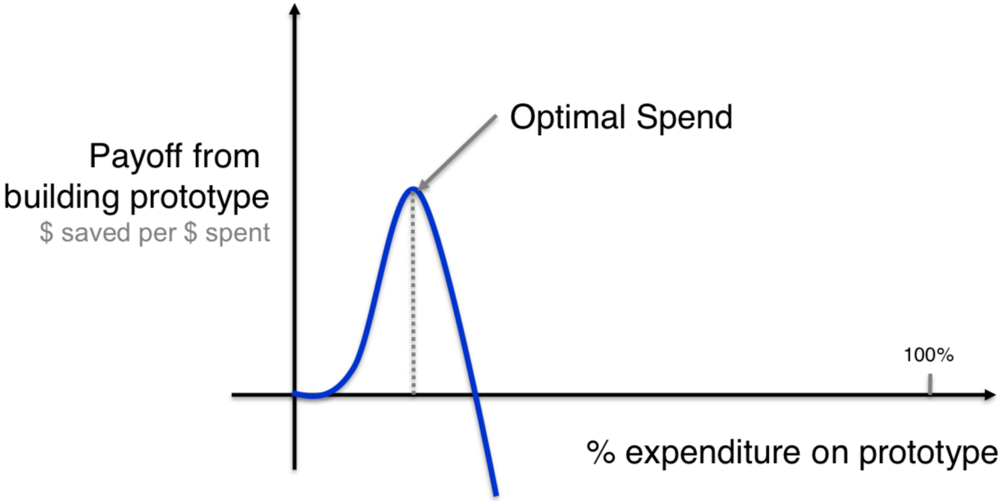
Perceived Cost of Doing Prototype	Low	High
Low	Maybe	Yes
High	No	Maybe

Calculate payoff in detail

Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping (1980s – 1990s)
  - It is necessary to perform a calculation to determine the payoff for the amount of effort to build a prototype.



Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping (1980s – 1990s)
    - Exact calculations can be difficult, so they may only be estimates. Typical estimates for the calculation may include:
        - Cost to construct the prototype
        - Percentage of the end feature’s implementation that will be reused from the prototype
        - Gross benefit from the effort
            - avoiding unneeded requirements
            - retiring some of the software risk
            - avoiding rework from late feedback

Models of Software Engineering Processes

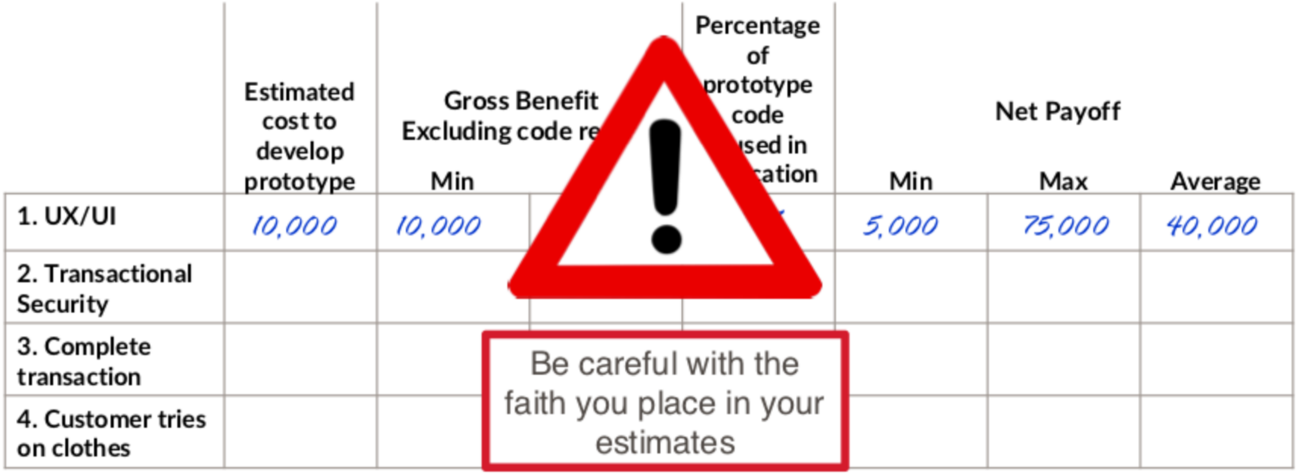
Software process models

- Evolutionary models and prototyping (1980s – 1990s)
  - Example: on-line clothes selling app
    - Four parts under consideration for prototyping:
      - UI/UX
      - transaction security
      - completing the transaction
      - customer trying on the clothes

Models of Software Engineering Processes

Software process models

- Evolutionary models and prototyping
  - Example: on-line clothes selling app
    - Payoff calculation:



	Estimated cost to develop prototype	Gross Benefit Excluding code re		Percentage of prototype code ..sed in ..cation	Net Payoff		
		Min			Min	Max	Average
1. UX/UI	10,000	10,000			5,000	75,000	40,000
2. Transactional Security							
3. Complete transaction							
4. Customer tries on clothes							

Be careful with the faith you place in your estimates

Payoff formulas:

Optimistic (max payoff) : assume highest benefits & lowest costs
Pessimistic (min payoff): assume lowest benefits & highest costs

Estimated payoff = [Gross Benefit] – [Estimated Costs]
= [Gross Benefit] – [cost to develop prototype]*(1 – [% reusable])

Models of Software Engineering Processes

Software process models

- Boehm’s spiral model (c. 1986)
  - A spiral model is a risk-driven software development process, based on unique risk patterns of a given project.
  - The spiral model guides a team to adopt elements of one or more process models, such as the incremental, waterfall, or evolutionary prototyping.

Models of Software Engineering Processes

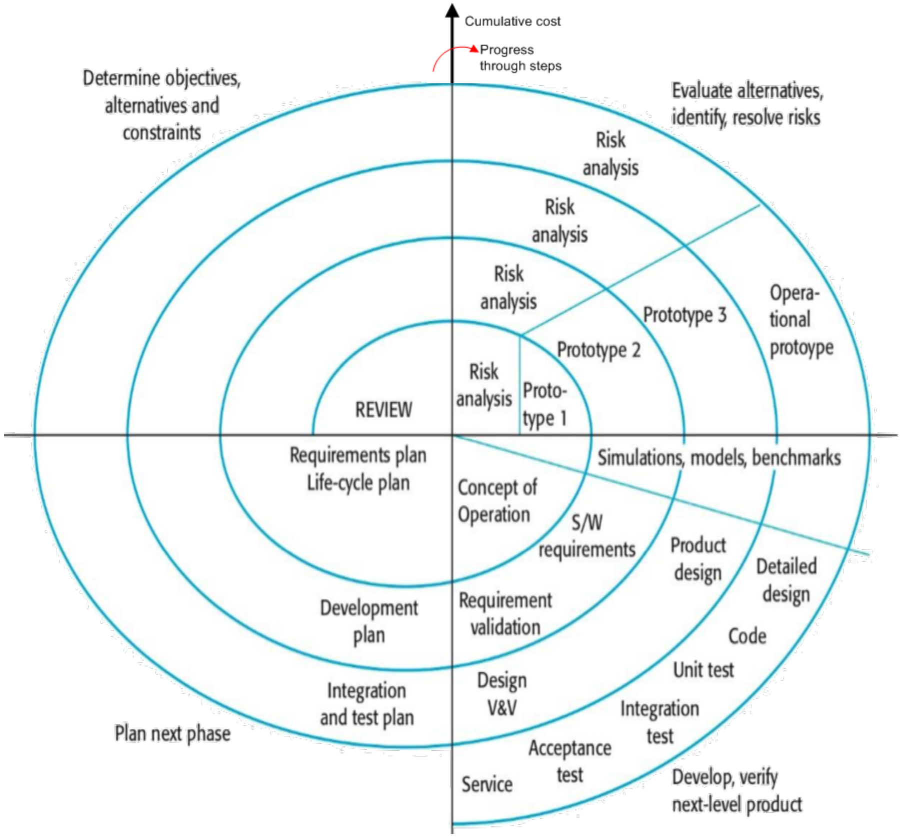
Software process models

- Boehm’s spiral model (c. 1986)
  - Barry Boehm first described the spiral model in his 1986 paper, "A Spiral Model of Software Development and Enhancement".
    - Boehm B, "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, August 1986
  - Boehm describes the spiral model as a "process model generator", where choices based on a project's risks generate an appropriate process model for the project.
  - Thus, the incremental, waterfall, prototyping, and other process models are special cases of the spiral model that fit the risk patterns of certain projects.

Models of Software Engineering Processes

Software process models

- Boehm’s spiral model (c. 1986)



Models of Software Engineering Processes

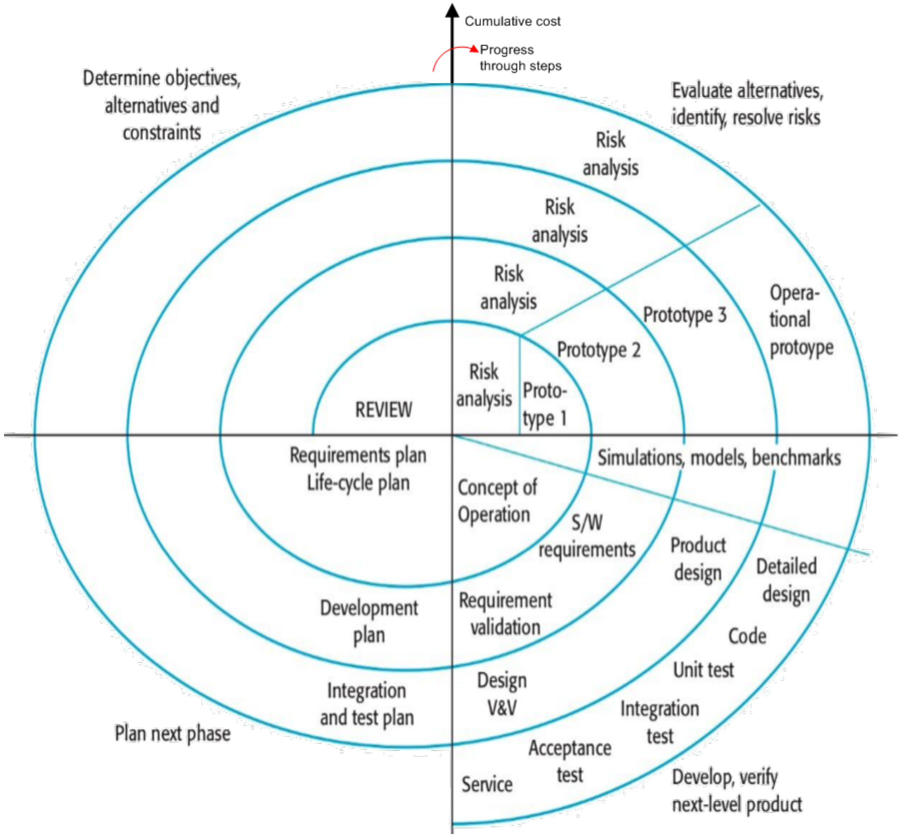
Software process models

- Boehm’s spiral model (c. 1986)
  - Each loop in the spiral is split into four sectors, corresponding to four basic activities that must occur in every cycle
    1. Consider the win conditions (objectives and constraints) of all success-criterial stakeholders.
    2. Identify and evaluate alternative approaches for satisfying the win conditions.
    3. Identify and resolve risks that stem from selected approach(es).
    4. Obtain approval from all success-criteria stakeholders, and commitment to pursue the next cycle.
  - Project cycles that omit or short-change any of these activities risk wasting effort by pursuing options that are unacceptable to key stakeholders, or are too risky.

Models of Software Engineering Processes

Software process models

- Boehm's spiral model (c. 1986)



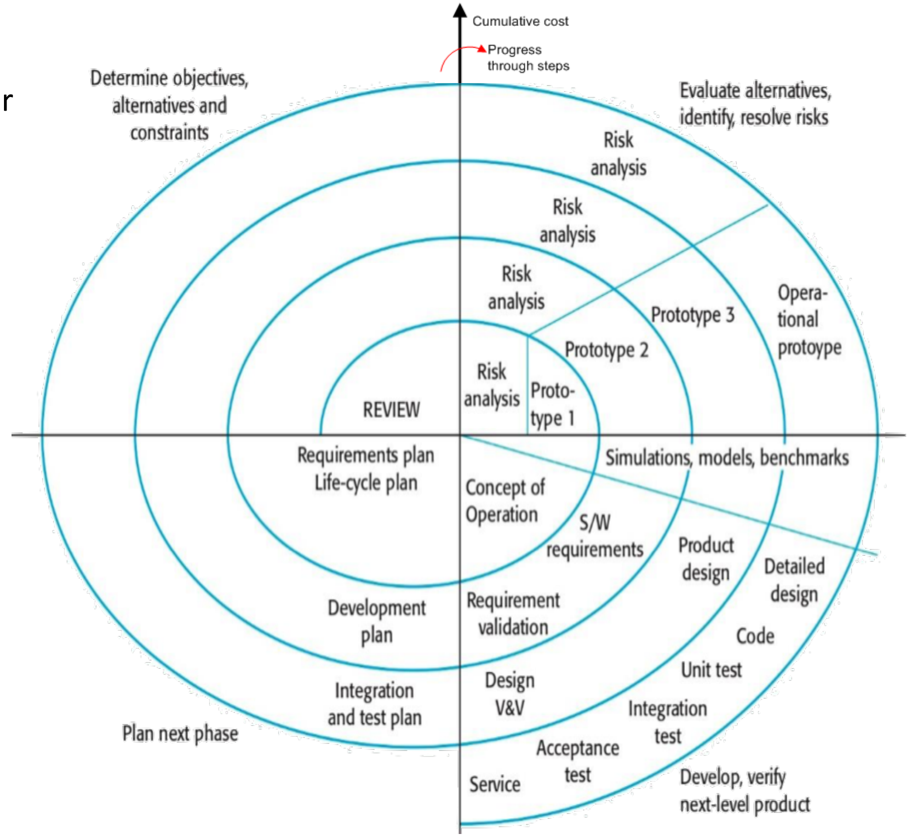
Models of Software Engineering Processes

Software process models

- Boehm's spiral model (c. 1986)

1. Determine Objectives

- Define specific objectives for that phase of the project.
- Identify constraints on the process and the product.
- Develop a detailed management plan. Identify project risks.
- Plan alternative strategies as back-ups, depending on these risks.



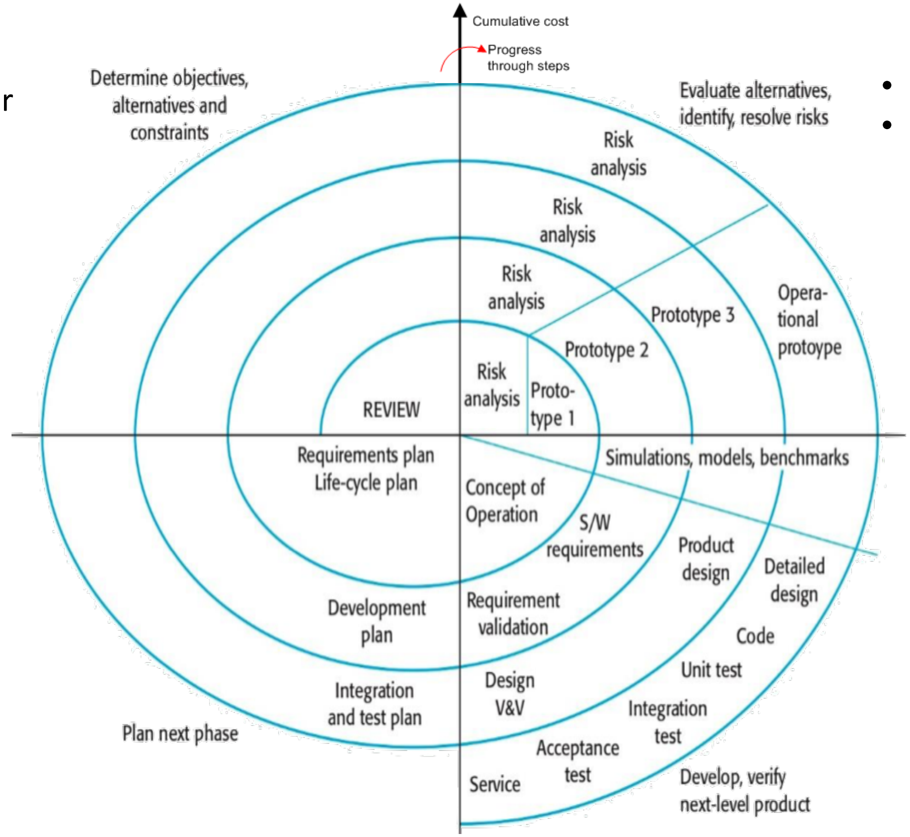
Models of Software Engineering Processes

Software process models

- Boehm's spiral model (c. 1986)

1. Determine Objectives

- Define specific objectives for that phase of the project.
- Identify constraints on the process and the product.
- Develop a detailed management plan. Identify project risks.
- Plan alternative strategies as back-ups, depending on these risks.



2. Assess Risk

- Assess risks in detail
- Take action to reduce risks

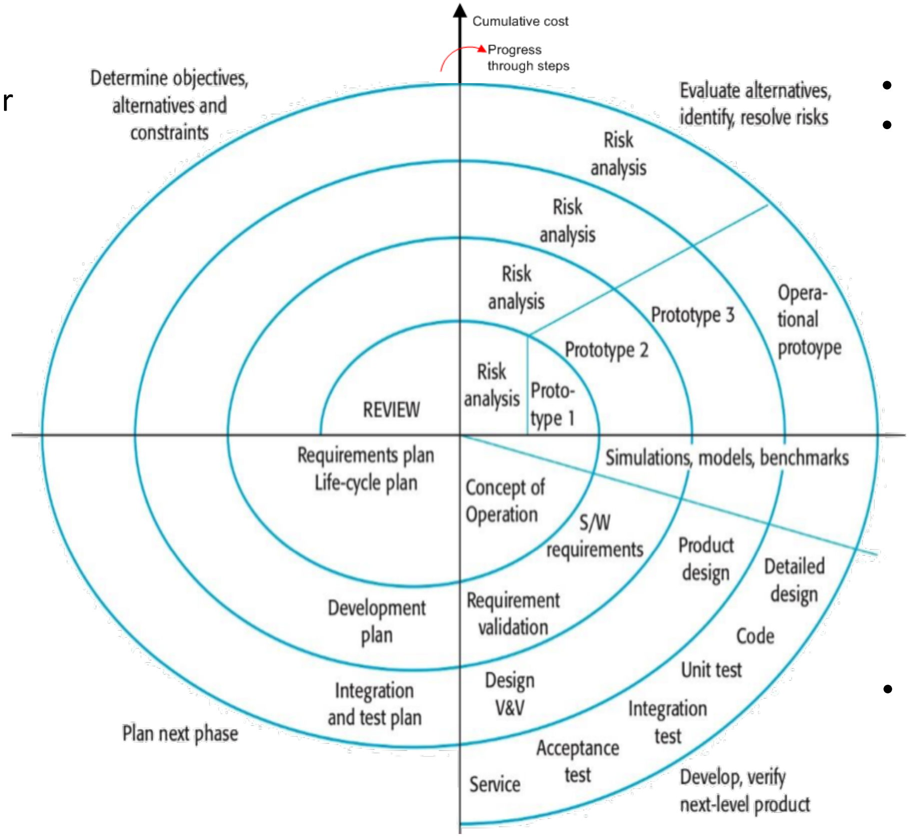
Models of Software Engineering Processes

Software process models

- Boehm's spiral model (c. 1986)

1. Determine Objectives

- Define specific objectives for that phase of the project.
- Identify constraints on the process and the product.
- Develop a detailed management plan. Identify project risks.
- Plan alternative strategies as back-ups, depending on these risks.



2. Assess Risk

- Assess risks in detail
- Take action to reduce risks

3. Develop and Verify

- Select development model appropriate for aims of the iteration.

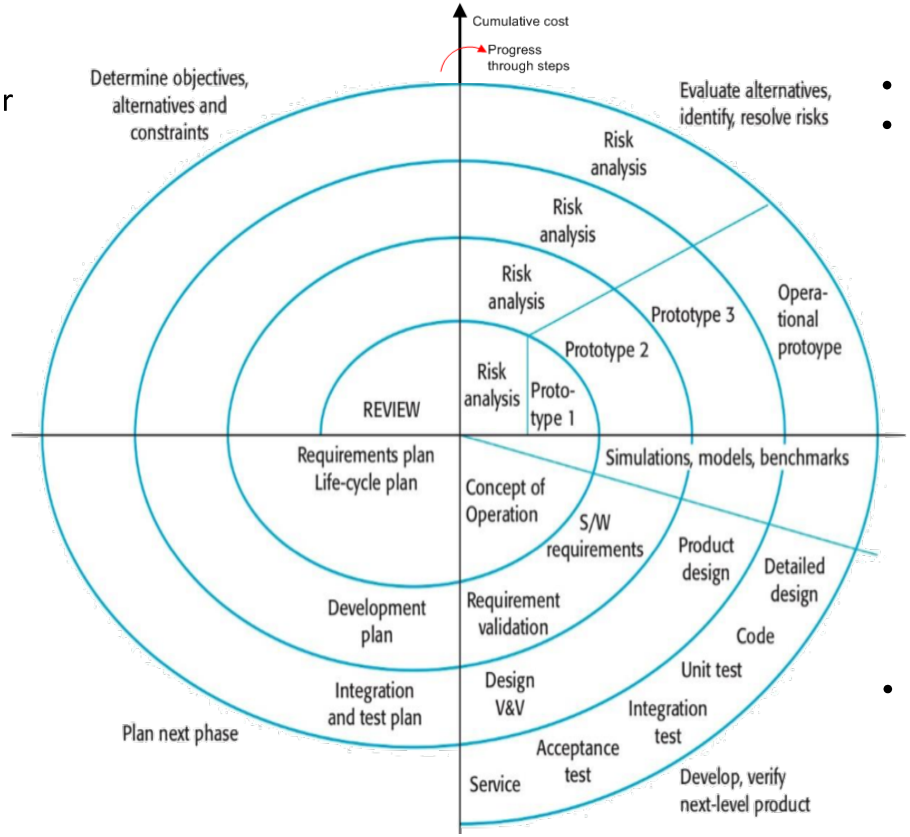
Models of Software Engineering Processes

Software process models

- Boehm's spiral model (c. 1986)

1. Determine Objectives

- Define specific objectives for that phase of the project.
- Identify constraints on the process and the product.
- Develop a detailed management plan. Identify project risks.
- Plan alternative strategies as back-ups, depending on these risks.



2. Assess Risk

- Assess risks in detail
- Take action to reduce risks

4. Plan Next Phase

- Review the project.
- Make a decision whether to continue with another iteration of the spiral.

3. Develop and Verify

- Select development model appropriate for aims of the iteration.

Models of Software Engineering Processes

Software process models

- Boehm’s spiral model (c. 1986)
  - Boehm identifies misconceptions from over-simplification in the original spiral model diagram. The most dangerous of these misconceptions are that:
    - the spiral is simply a sequence of waterfall increments
    - all project activities follow a single spiral sequence
    - that every activity in the diagram must be be performed, and in the order shown.

Models of Software Engineering Processes

Software process models

- Boehm’s spiral model (c. 1986)
    - Benefits of the model include:
        - Risks are managed continuously throughout the process
        - Software evolves as the project progresses
        - Planning is built into the process
        - System grows as a series of evolutionary releases
        - Reflects iterative nature of development
    - Issues with the model include:
        - It is fairly complicated
        - May be overkill for small projects, especially if risk analysis ends up being the dominant cost.

Models of Software Engineering Processes

Summary

- In this we began to study specific types of software engineering process models that have been developed to guide product development.
- For each process model, we sought to
  - understand the underlying principles of the model
  - look at how it fits in to the generic software process framework
  - see how the process works
  - consider its relative strengths and weaknesses

Models of Software Engineering Processes

Summary

- We looked at the following historical process models.
  - Code and fix (1950s – 1960s)
    - no underlying engineering model
  - Waterfall model (1970s – 1980s)
    - original engineering-based model, proposed as “strawman”
  - Incremental models (1970s – 1980s)
    - multiple waterfall cycles, fixed requirements
  - Evolutionary models (1970s – 1980s)
    - multiple waterfall cycles, evolving requirements
  - Evolutionary models with prototyping (1980s – 1990s)
    - Cyclic model includes continuous refinement
  - Spiral framework (c. 1986)
    - Evolutionary model sought to minimize risk at each turn