CS342 Software Engineering

Dr. Ismail Hababeh
German Jordanian University
Lecture 3

SOFTWARE LIFE-CYCLE MODELS

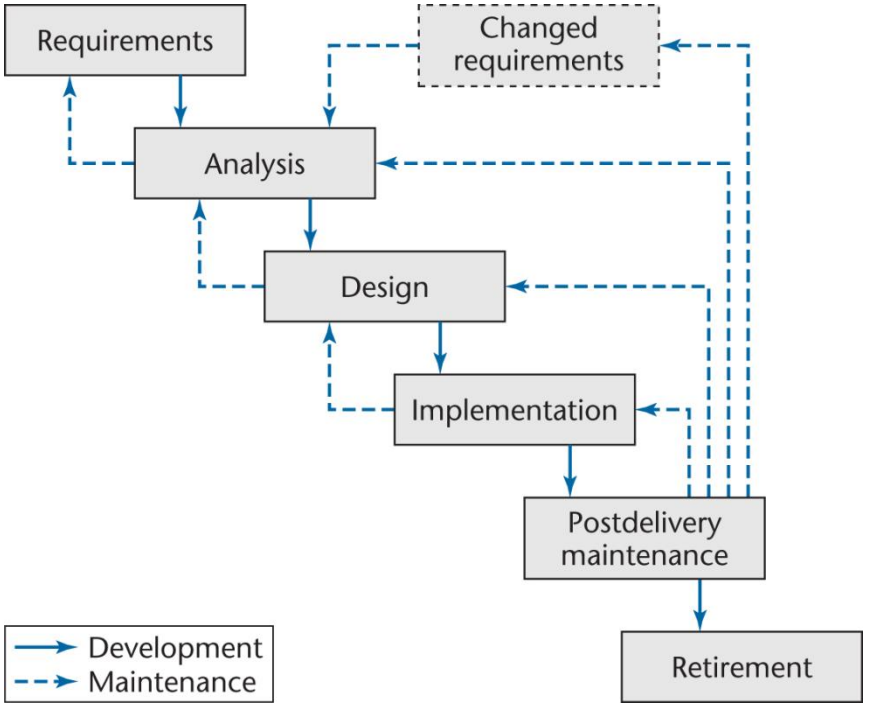
Adapted from Software Engineering, by Dr. Paul E. Young & slides by Dr. Mohammad Daoud

Current Software Life-Cycle Models

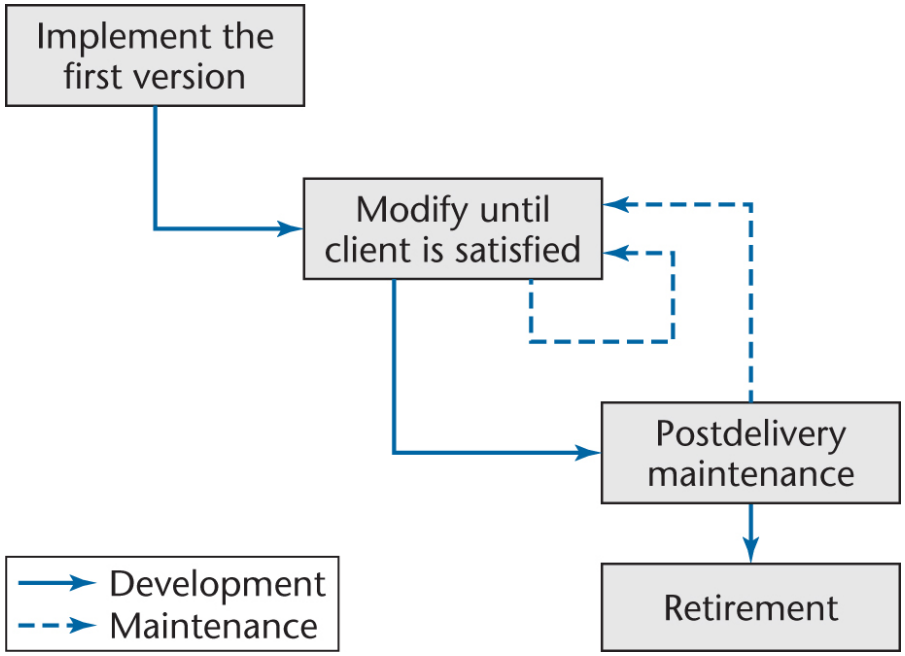
- The following software life-cycle models are well known in the software engineering field:
  - Waterfall
  - Code-and-fix
  - Rapid prototyping
  - Open-source
  - Agile processes
  - Synchronize-and-stabilize
  - Spiral
  - Unified Processes

Waterfall Model

- Characterized by
  - Feedback loops
  - Documentation-driven
- Advantages
  - Documentation
  - Maintenance is easier
- Disadvantages
  - Its hard for the client to understand the complex specification document



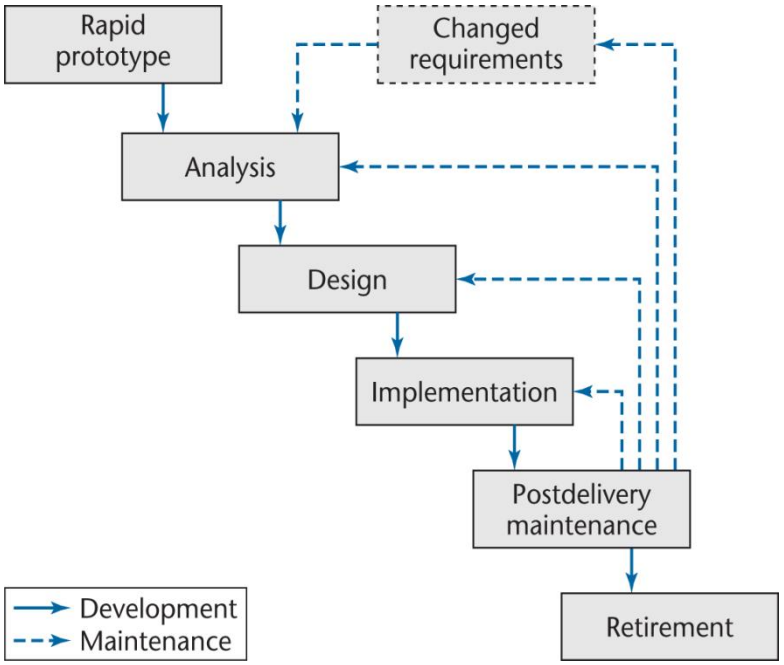
Code-and-Fix Model



- The easiest way to develop software
- No design
- No specifications → Maintenance horrible
- The most expensive model

Rapid Prototyping Model

- Build a rapid prototype (first version) and ask the user to try it.
- Once the client is satisfied with the prototype, the developers can draw the specification document
- The feedback loops are less compared to the waterfall
- In the post-delivery maintenance, we can re-enter requirement, analysis, design, or implementation
- Rapid built: the prototype should be built rabidly



Rapid Prototyping

- Rapid built
  - Imperfections can be ignored
  - Shows only key functionality
  - Emphasis on only what the client sees
  - Error checking, file updating can be ignored
- Aim:
  - To provide the client with an understanding model of the product
- A rapid prototype is built for change
  - Languages for rapid prototyping include 4GLs

Rapid Prototyping - Human Factors

- The *client and intended users* must interact with the user interface
- Human-computer interface (HCI)
  - No command line
  - Point and click
  - Menus, icons, buttons

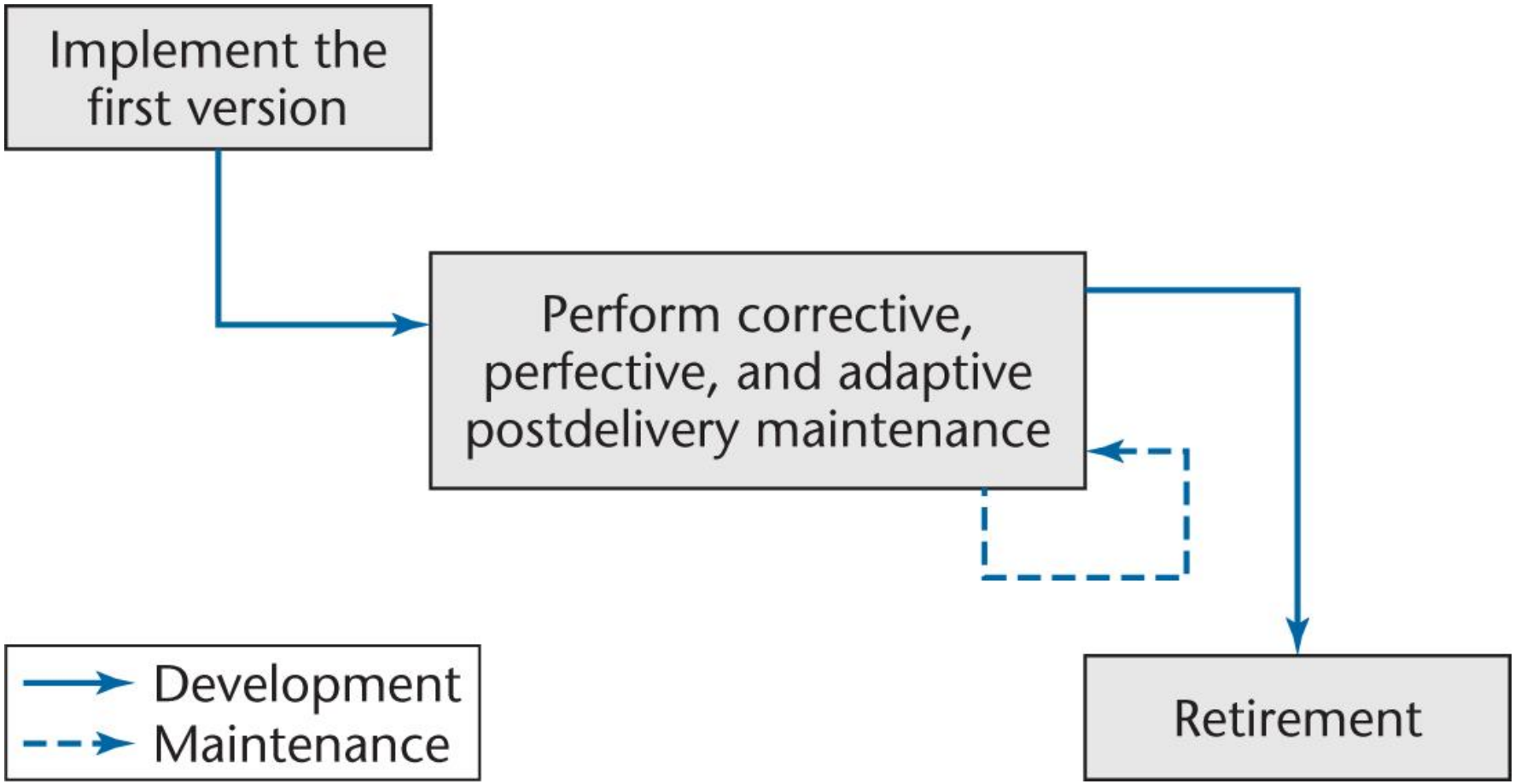
Rapid Prototyping - Human Factors

- Human factors that must be taken into consideration:
  - Avoid a lengthy sequence of menus
  - Uniformity of appearance is important
  - Allow the expertise level of an interface to be modified

Rapid Prototype - Reusing

- Reusing a rapid prototype is essentially code-and-fix
- Changes are made to a working product
  - Expensive
- Maintenance is hard without specification and design documents
- Real-time constraints are hard to meet
- We can safely reuse parts of a rapid prototype if
  - This is prearranged
  - Those parts pass SQA inspections

Open-Source Model



Open-Source Model

Two informal phases:

- First phase activities:
  - builds an initial version, then make it available via the Internet
  - If there is sufficient interest in the project and the initial version is widely downloaded, then
    - Users become co-developers
    - The product is extended

Open-Source Model

- Second phase activities:
  - Reporting and correcting defects
    - Corrective maintenance
  - Transfer the program to a new environment
    - Adaptive maintenance
  - Adding additional functionality
    - Perfective maintenance
  - Consists of post-delivery maintenance
    - The “co-developers” should rather be “co-maintainers”

Agile Model

- Based on the principle:

  Deliver working software frequently
- One way of achieving this is using *time-boxing*
- *time-boxing:*
  - Specific amount of time is set for a task
  - Used as a time-management technique
  - Typically, 3 weeks for each iteration

Agile Model

- It gives the client confidence to know that a new version with additional functionality will arrive, for example every 3 weeks
- The **developers team** know that they will have time limit to deliver a new iteration
- No client interference of any kind
- If it is impossible to complete the entire task in the timebox, the work may be reduced.
  - Agile processes demand fixed time, not fixed features

Agile Model

- Success with small-scale software development.
- Good when requirements are dynamic or changed.
- The Evaluation key: the impact of agile processes on post-delivery maintenance.
- Refactoring is an essential component of agile processes.
    - continues during maintenance
    - consumes large amount of the overall cost
  - The code is reorganized until the team is satisfied that the design runs all test cases.
  - No data on maintenance.

Synchronize and Stabilize Model

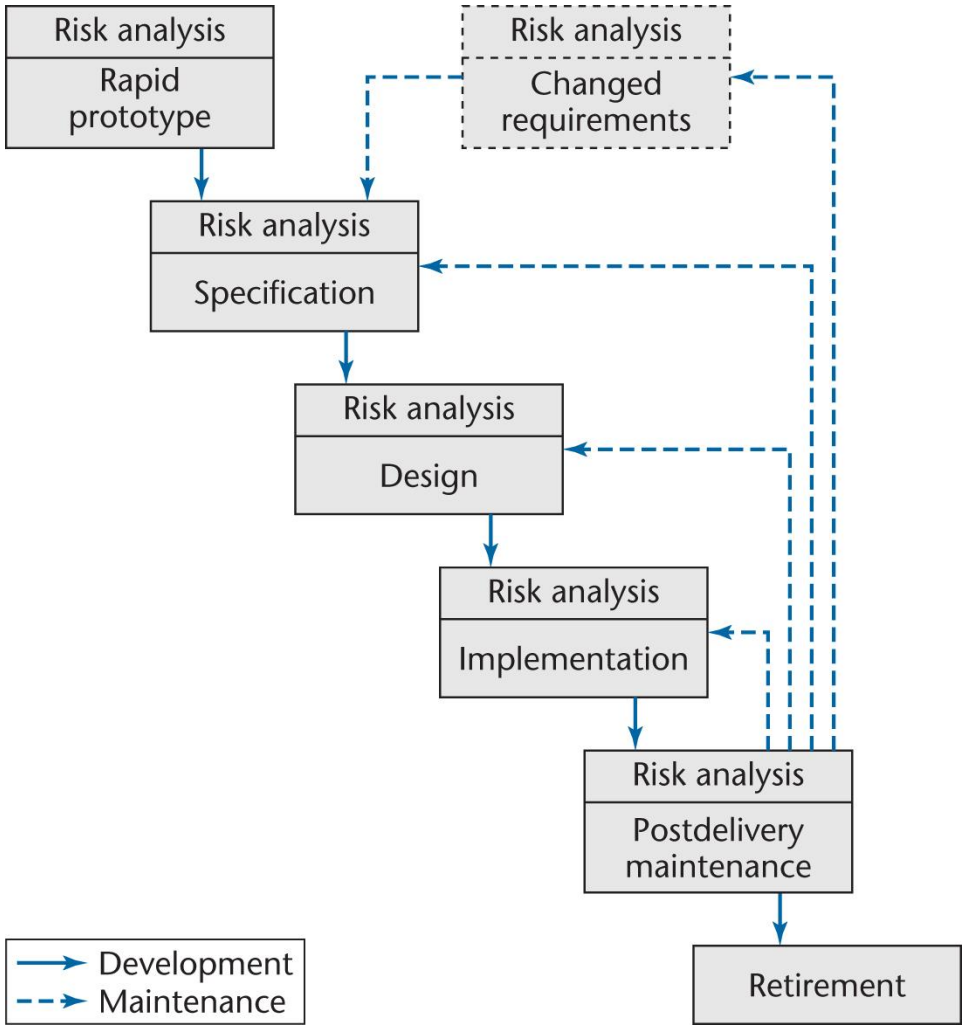
- Microsoft's life-cycle model
- Requirements analysis — interview potential customers
- Draw up specifications
- Divide project into 3 or 4 components
- Each component is carried out by small teams working in parallel

Synchronize and Stabilize Model

- At the end of the day, all teams *synchronize* (test and debug)
- At the end of the build, *stabilize* (fix all detected faults and freeze the build) $\longrightarrow$ no further changes will be made for the specifications.
- The repeated synchronization step:
  - Ensures that the various components always work together
  - Get early insights into the operation of the product

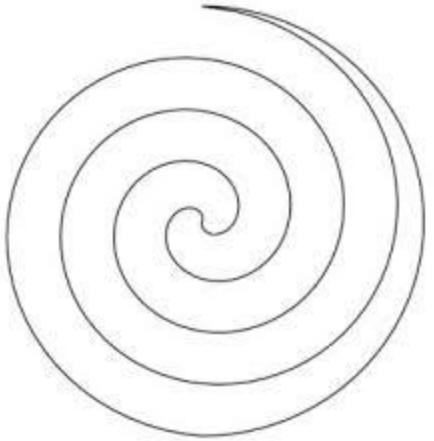
Spiral Model

- Simplified form of:
  - Rapid prototyping model + risk analysis preceding each phase
- If all risks cannot be solved, the project is terminated immediately

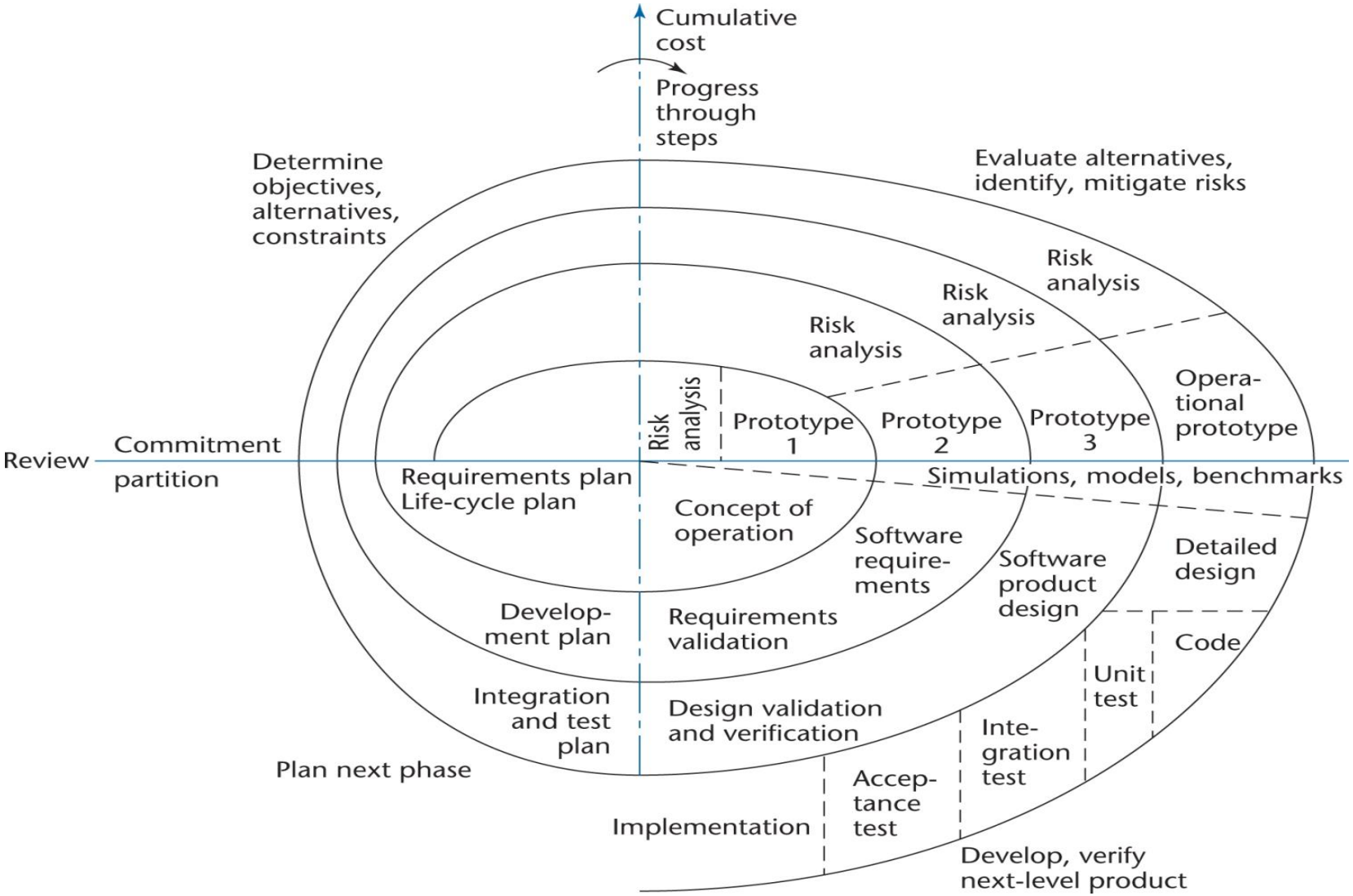


Spiral Model

- Each phase is preceded by
  - Risk analysis
  - Alternatives
- Each phase is followed by
  - Evaluation
  - Planning of the next phase
- Spiral model dimensions:
  - Radial: accumulative cost up to date
  - Angular: progress through the spiral
- Each spiral cycle corresponds to a phase



Spiral Model



Spiral Model

- Strengths
  - It is easy to evaluate how much to test (through risk analysis).
  - No distinction is made between development and maintenance (maintenance is just another cycle of the spiral).
- Weaknesses
  - Used for large-scale software only
  - Used for internal (in-house) software only

The Unified Process

- Three researchers, Booch, Jacobson, and Rumbaugh published a complete object-oriented analysis and design methodology that unified their separate methodologies.
- The name of this methodology is ***Unified Process***

The Unified Process

- The Unified Process is *not* a series of steps for constructing a software product because:
  - No such single methodology could exist
  - There is a wide variety of different types of software
- The Unified Process is an adaptable methodology
  - The methodology should be modified to develop a specific software product.

The Unified Process

- The Unified Process is a modeling technique
  - A *model* is a set of UML diagrams that represent various aspects for developing the software product.
- UML stands for *Unified Modeling Language*
  - UML is the tool that is used to represent (model) the target software product
  - UML is graphical (diagrams)
  - UML diagrams enable software engineers to communicate quickly and accurately

Comparison of Life-Cycle Models

Life-Cycle Model	Strengths	Weaknesses
Evolution-tree model (Section 2.2)	Closely models real-world software production Equivalent to the iterative-and-incremental model	
Iterative-and-incremental life-cycle model (Section 2.5)	Closely models real-world software production Underlies the Unified Process	
Code-and-fix life-cycle model (Section 2.9.1)	Fine for short programs that require no maintenance	Totally unsatisfactory for nontrivial programs
Waterfall life-cycle model (Section 2.9.2)	Disciplined approach Document driven	Delivered product may not meet client's needs
Rapid-prototyping life-cycle model (Section 2.9.3)	Ensures that the delivered product meets the client's needs	Not yet proven beyond all doubt
Open-source life-cycle model (Section 2.9.4)	Has worked extremely well in a small number of instances	Limited applicability Usually does not work
Agile processes (Section 2.9.5)	Work well when the client's requirements are vague	Appear to work on only small-scale projects
Synchronize-and-stabilize life-cycle model (Section 2.9.6)	Future users' needs are met Ensures that components can be successfully integrated	Has not been widely used other than at Microsoft
Spiral life-cycle model (Section 2.9.7)	Risk driven	Can be used for only large-scale, in-house products Developers have to be competent in risk analysis and risk resolution