Life Cycle Models

Part #3

1

2.9 Other Life-Cycle Models

- The following life-cycle models are presented and compared:
  - Code-and-fix life-cycle model
  - Waterfall life-cycle model
  - Rapid prototyping life-cycle model
  - Open-source life-cycle model
  - Agile processes
  - Synchronize-and-stabilize life-cycle model
  - Spiral life-cycle model

2

Thoughts

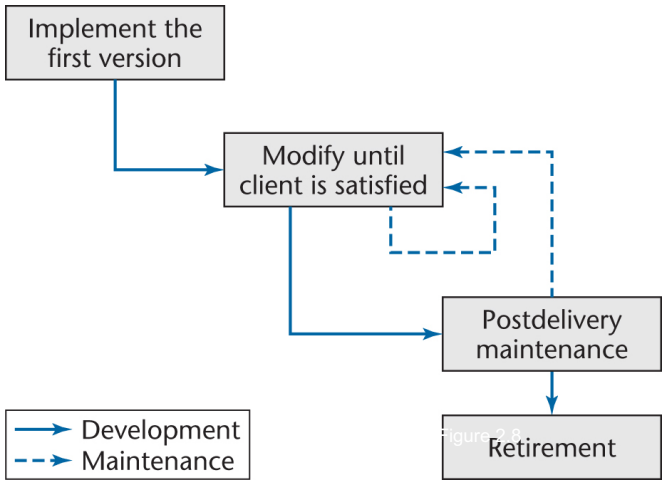
Don't assign good vs. evil traits to anything in software.

Don't assign good vs. evil to Life Cycle Models.

3

2.9.1 Code-and-Fix Model

- No design
- No specifications
  - Maintenance nightmare



4

Code-and-Fix Model (contd)

- The easiest way to develop software
- Extremely, Extremely popular
- The most expensive way (most times)
  - But nobody believes that

5

2.9.2 Waterfall Model

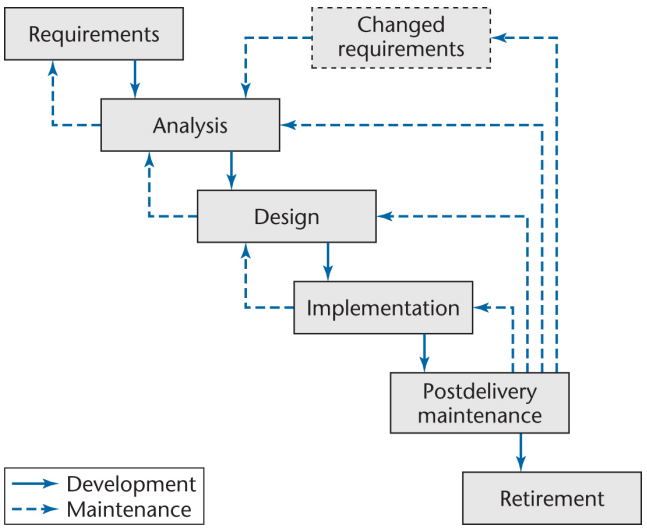


Figure 2.9

6

2.9.2 Waterfall Model

Characterized by

- Feedback loops
- Documentation-driven

Advantages

- Documentation
- Maintenance is easier

Disadvantages

- Specification document
  - Joe and Jane Johnson
  - Mark Marberry

7

2.9.3 Rapid Prototyping Model

- Linear model
- "Rapid"

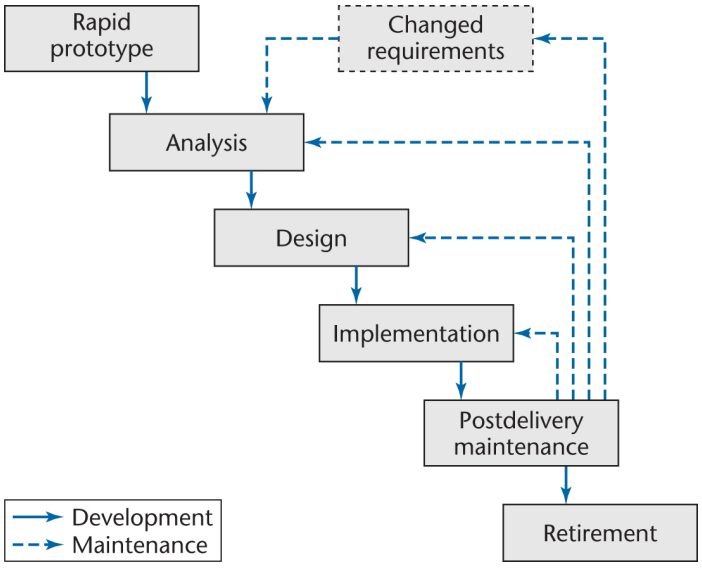


Figure 2.10

8

Strengths and Pitfalls of Rapid Proto

Strengths

- Early functionality
- Mitigate risk.
  - You can identify your technical debt early (and often)
- Darn fun!

Pitfalls

- Demo's of prototypes often imply completeness!
- Prototypes sometimes never go away or impose an architecture.
- Human emotions

9

2.9.4 Open-Source Life-Cycle Model

Two informal phases

- First, one individual builds an initial version
  - Made available via the Internet (e.g., SourceForge.net)
- Then, if there is sufficient interest in the project
  - The initial version is widely downloaded
  - Users become co-developers
  - The product is extended

Key point: IN many projects individuals generally work voluntarily on an open-source project in their spare time

10

Open-Source Life-Cycle Model (contd)

Postdelivery maintenance life-cycle model

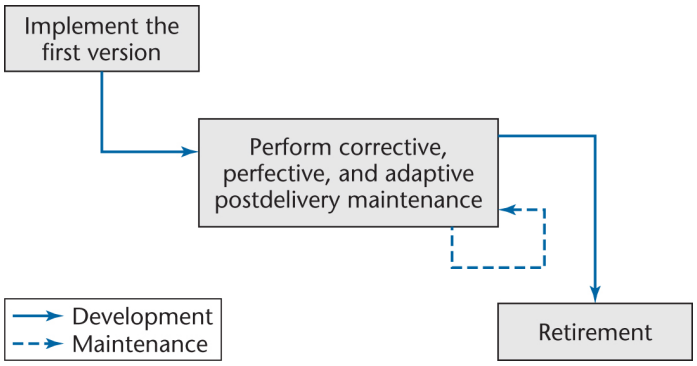


Figure 2.11

11

Open-Source Life-Cycle Model

- Closed-source software is maintained and tested by employees
  - Users can submit failure reports but never fault reports (the source code is not available)
- A lot of open-source software is generally maintained by unpaid volunteers
  - Users are strongly encouraged to submit defect reports, both failure reports and fault reports

12

Open-Source Life-Cycle Model

- An initial working version is produced when using
  - The rapid-prototyping model;
  - The code-and-fix model; and
  - The open-source life-cycle model
- Then:
  - Rapid-prototyping model
    - The initial version is discarded
  - Code-and-fix model and open-source life-cycle model
    - The initial version becomes the target product

13

Open-Source Life-Cycle Model

- Consequently, in an open-source project, in many open source projects, there are no specifications and no design
- How have some open-source projects been so successful without specifications or designs?

14

Open-Source Life-Cycle Model

- About half of the open-source projects on the Web have not attracted a team to work on the project
- Even where work has started, the overwhelming preponderance will never be completed
- But when the open-source model has worked, it has sometimes been incredibly successful
  - The open-source products previously listed have been utilized on a regular basis by millions of users

15

2.9.5 Agile Processes

- Began as controversial approach
- *Stories* (features client wants)
  - Estimate duration and cost of each story
  - Select stories for next build
  - Each build is divided into tasks
  - Test cases for a task are drawn up first
- Pair programming
- Continuous integration of tasks

16

Unusual Features of XP

- The computers are put in the center of a large room lined with cubicles
- A client representative is always present
- Software professionals cannot work overtime for 2 successive weeks
- No specialization
- *Refactoring* (design modification)

17

Agile Processes

- XP is one of a number of new paradigms collectively referred to as *agile processes*
- Seventeen software developers (later dubbed the "Agile Alliance") met at a Utah ski resort for two days in February 2001 and produced the *Manifesto for Agile Software Development*
- The Agile Alliance did not prescribe a specific life-cycle model
  - Instead, they laid out a group of underlying principles

18

Agile Processes

- Agile processes are a collection of new paradigms characterized by
  - Less emphasis on analysis and design
  - Earlier implementation (working software is considered more important than documentation)
  - Responsiveness to change
  - Close collaboration with the client

19

Agile Processes

- A principle in the *Manifesto* is
  - Deliver working software frequently
  - Ideally every 2 or 3 weeks
- One way of achieving this is to use *timeboxing*
  - Used for many years as a time-management technique
- A specific amount of time is set aside for a task
  - Typically 3 weeks for each iteration
  - The team members then do the best job they can during that time

20

Agile Processes

- It gives the client confidence to know that a new version with additional functionality will arrive every 3 weeks
- The developers know that they will have 3 weeks (but no more) to deliver a new iteration
  - Without client interference of any kind
- If it is impossible to complete the entire task in the timebox, the work may be reduced ("descoped")
  - Agile processes demand fixed time, not fixed features

21

Agile Processes

- Another common feature of agile processes is *stand-up meetings*
  - Short meetings held at a regular time each day
  - Attendance is required
- Participants stand in a circle
  - They do not sit around a table
  - To ensure the meeting lasts no more than 15 minutes

22

Agile Processes

- At a stand-up meeting, each team member in turn answers five questions:
  - What have I done since yesterday's meeting?
  - What am I working on today?
  - What problems are preventing me from achieving this?
  - What have we forgotten?
  - What did I learn that I would like to share with the team?

23

Agile Processes

- Stand-up meetings and timeboxing are both
  - Successful management techniques
  - Now utilized within the context of agile processes
- Both techniques are instances of two basic principles that underlie all agile methods:
  - Communication; and
  - Satisfying the client's needs as quickly as possible

24

Evaluating Agile Processes

- Agile processes have had some successes with small-scale software development
  - However, medium- and large-scale software development are completely different
- The key decider: the impact of agile processes on post-delivery maintenance
  - Refactoring is an essential component of agile processes
  - Refactoring continues during maintenance
  - Will refactoring increase the cost of post-delivery maintenance, as indicated by preliminary research?

25

Evaluating Agile Processes

- Agile processes are good when requirements are vague or changing
- In 2000, Williams, Kessler, Cunningham, and Jeffries showed that **pair programming** leads to
  - The development of higher-quality code,
  - In a shorter time,
  - With greater job satisfaction

26

Evaluating Agile Processes

- The *Manifesto for Agile Software Development* claims that agile processes are superior to more disciplined processes like the Unified Process
- Skeptics respond that proponents of agile processes are little more than hackers
- However, there is a middle ground
  - It is possible to incorporate proven features of agile processes within the framework of disciplined processes

27

Evaluating Agile Processes

- In conclusion
  - Agile processes appear to be a useful approach to building software products when the client's requirements are evolving and not mission critical.
  - User Interface Projects?.
  - Also, some of the proven features of agile processes can be effectively utilized within the context of other life-cycle models

28

2.9.6 Synchronize-and Stabilize Model

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

29

Synchronize-and Stabilize Model

- At the end of the day — *synchronize* (test and debug)
- At the end of the build — *stabilize* (freeze the build)
- Components always work together
  - Get early insights into the operation of the product

30

2.9.7 Spiral Model

- Simplified form
  - Rapid prototyping model plus risk analysis preceding each phase

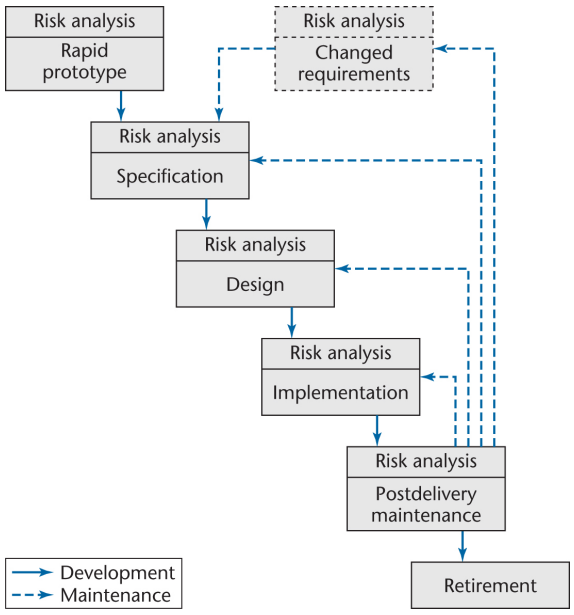


Figure 2.12

31

A Key Point of the Spiral Model

If all risks cannot be mitigated, the project is immediately terminated

32

Full Spiral Model

- Precede each phase by
  - Alternatives
  - Risk analysis
- Follow each phase by
  - Evaluation
  - Planning of the next phase
- Radial dimension: cumulative cost to date
- Angular dimension: progress through the spiral

33

Full Spiral Model (contd)

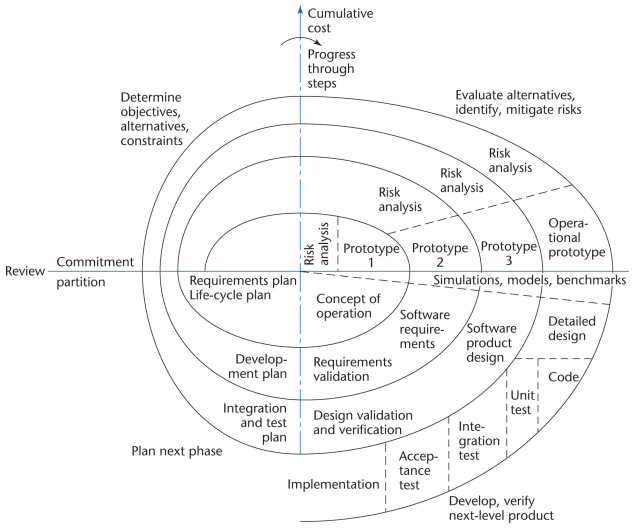


Figure 2.13

34

Analysis of the Spiral Model

- Strengths
  - It is easy to judge how much to test
  - No distinction is made between development and maintenance
- Weaknesses
  - For large-scale software only
  - For internal (in-house) software only

35

2.10 Comparison of Life-Cycle Models

- Different life-cycle models have been presented
  - Each with its own strengths and weaknesses
- Criteria for deciding on a model include:
  - The organization
  - Its management
  - The skills of the employees
  - The nature of the product
- Best suggestion
  - "Mix-and-match" life-cycle model

36

Conclusion

Best life cycle model depends on your particular project.

37