Software Engineering

Norbert Pataki

October 18, 2007

Software Development Process Models

- Huge projects, the software is a product.
- cost, deadlines, quality, etc.
- the development process must be controlled!
- Waterfall
- Spiral
- V–Model
- IBM RUP
- XP

Waterfall model - 1970

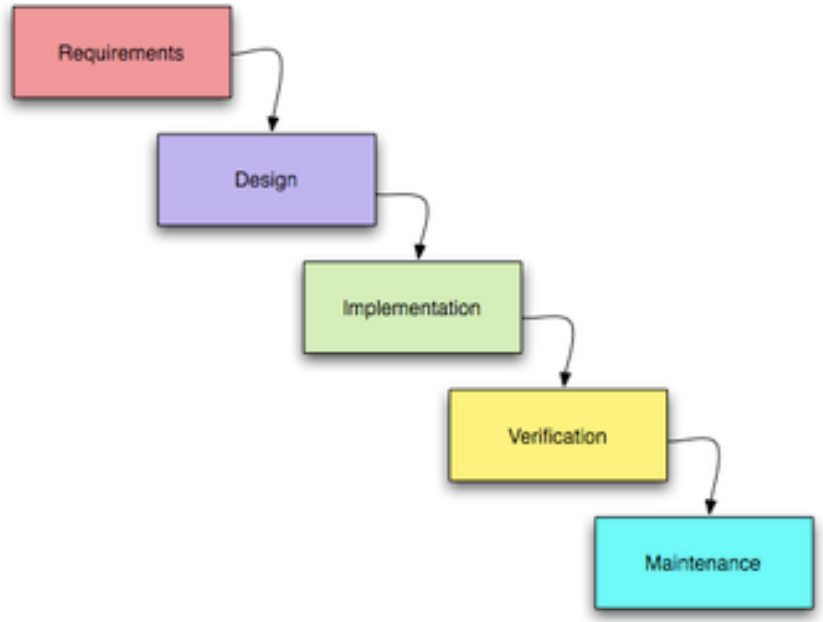
- sequential software development model

Phases:

- requirements analysis
- software design
- implementation
- testing (validating)
- integration
- maintenance

Waterfall model

phases proceeded by step-by-step



Waterfall model

Advantages:

- it can save time and effort because making sure that requirements and design are absolutely correct before implementation
- Waterfall model places emphasis on documentation (requirements documents, design documents, source code)
- The programmers must work well according to the complete design, ensuring that the integration of the system proceeds smoothly
- The result is “highly reliable system”
- The waterfall model is simple to comply with it

Waterfall model

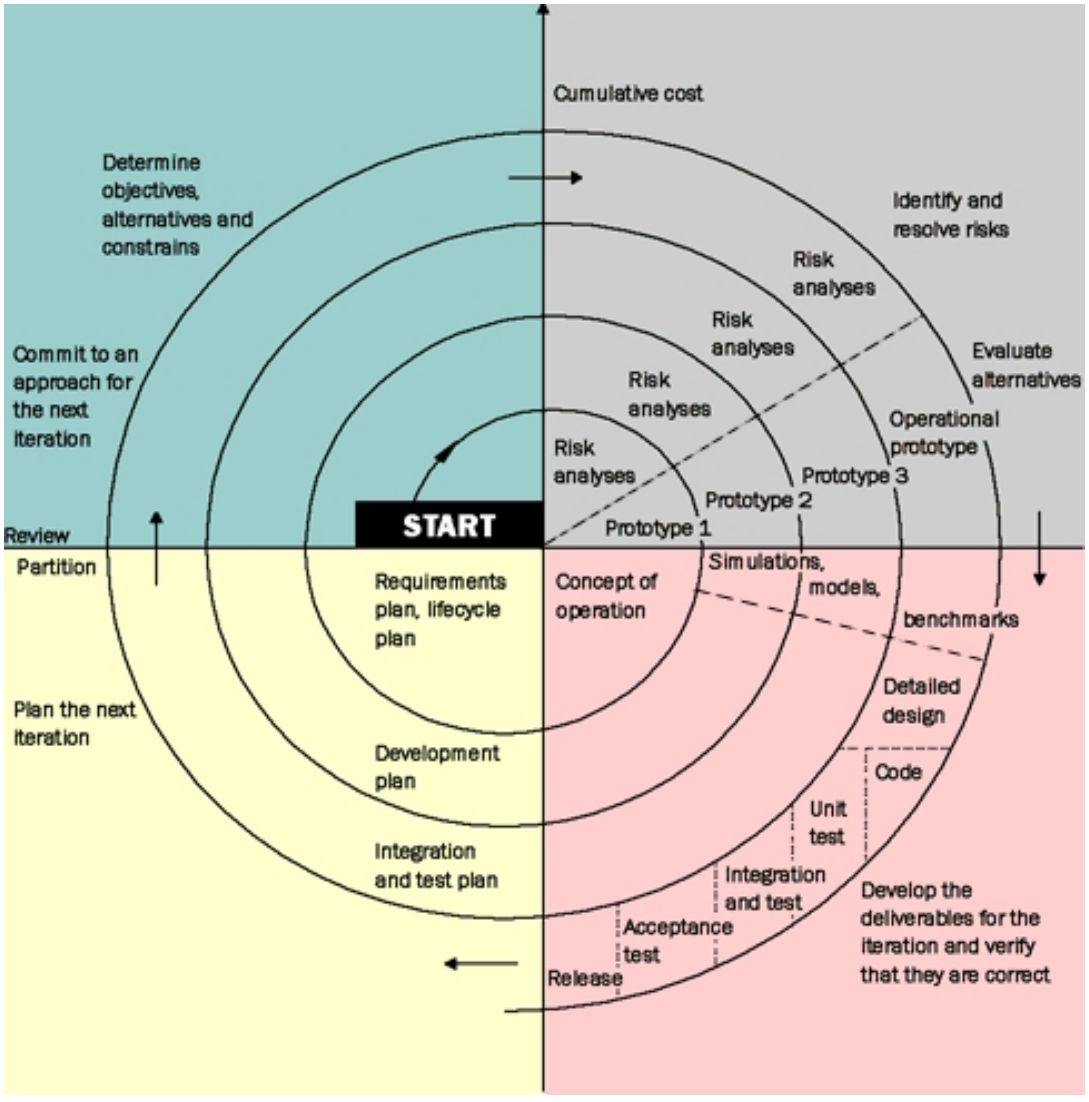
Disadvantages:

- Not possible to get one phase of a software product's lifecycle absolutely completed without any bugs before moving on to the next phases and learning from them. Feedback is needed.
- Validation can expect the repeatation of the entire lifecycle.
- Hard to arrange the project's work.

Spiral model - 1988

- Barry Boehm defined the spiral (lifecycle) model in 1988
- For large, expensive and complicated projects
- Combines the prototyping and waterfall model
- Iterative model

Spiral model



Spiral model

Advantages:

- The project is well-documented bacause documents are created in all phases.
- Validation is always done after implementation.
- Subphases, the project's structure and iterations are freely managable by the project manager.
- Estimates get more realistic as work progresses, because important issues are discovered earlier.

Spiral model

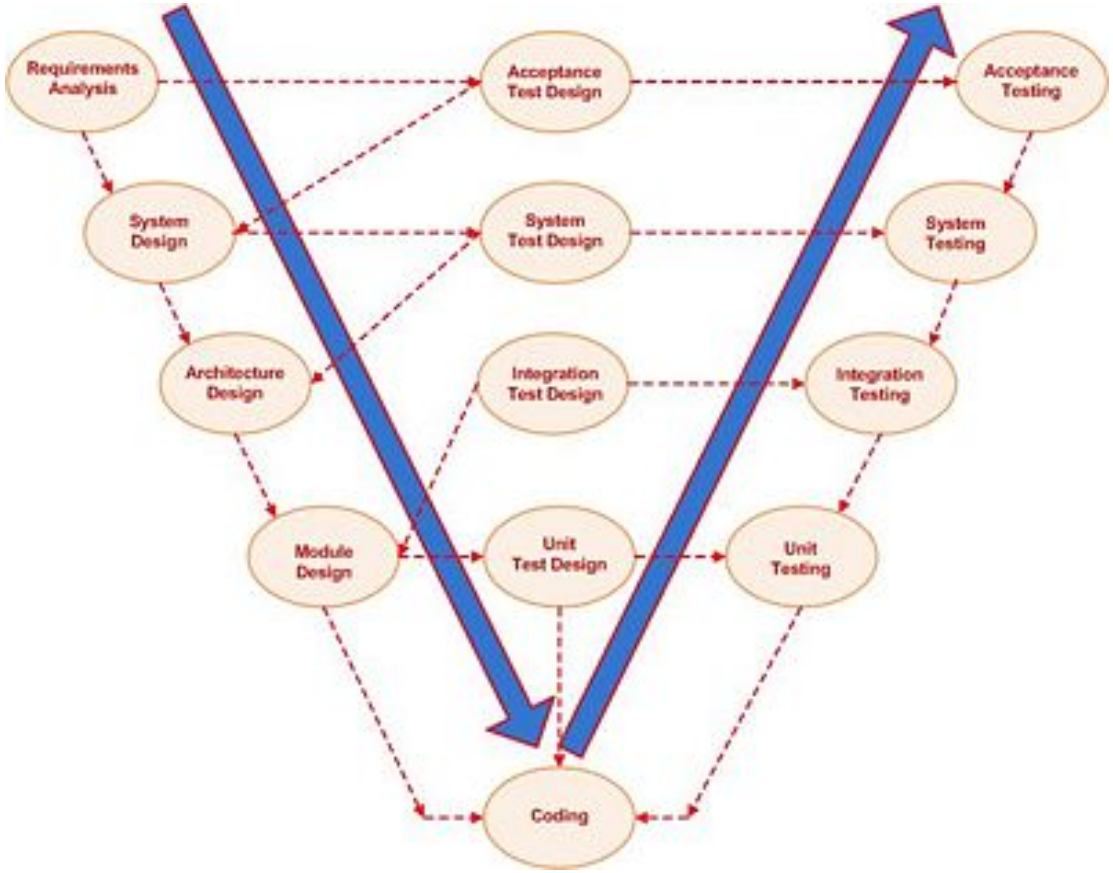
Disadvantages:

- The application of the spiral model is hard. These phases are hard to separate this way in a real project.
- The experts cannot be applied in a profitable way (for instance, concurrent work).
- The whole spiral model is uneconomical.

V-model

- Extension of the waterfall model
- “The result of the evolution of software testing”
- Developed by the German Ministry of Defense
- Standard in the German Army since 1992

V-model


Requirements Analysis
Acceptance Test Design
Acceptance Testing
System Design
System Test Design
System Testing
Architecture Design
Integration Test Design
Integration Testing
Module Design
Unit Test Design
Unit Testing
Coding

V-model

Advantages:

- The model deploys a well-structured method in which each phase can be implemented by the detailed documentation of the previous phase.
- Testing activities (like test designing) start at the beginning of the project well before coding and therefore saves a huge amount of the project time and money.

RUP

- stands for Rational Unified Process
- iterative software development process
- developed by the creator of the UML
- The first version was released in 1998

Why projects fail

- Ad hoc requirements management
- Ambigous and imprecise communication
- Fragile architecture
- Overwhelming complexity
- Undetected inconsistencies in requirements, design, and code
- Insufficient testing
- Subjective assessment of project status
- Failure to attack risks
- Uncontrolled change propagation
- Insufficient automation

6 Key Principles of Business-Driven Development

1. Adapt the process
2. Balance stakeholder priorities
3. Collaborate across teams
4. Demonstrate value iteratively
5. Elevate the level of abstraction
6. Focus continuously on quality

6 Key Principles of Business-Driven Development

- Adapt the process:
  - Size of a organization / project must be proper to its needs.
  - RUP provides pre-configured process templates for small, medium and large projects, which can be used for easier adoption.
- Balance stakeholder priorities:
  - Business goals and stakeholder needs ofter compete and conflict.
  - Must be balanced out the parties involved.

6 Key Principles of Business-Driven Development

- Collaborate across teams
  - Software engineering is a team effort.
  - Increasing demand of globally distributed development.
  - Modern communication tools should be used
  - Share: requirements, exchange of metrics, test results, release management and project plans.
- Demonstrate value iteratively
  - Incremental development
  - The progress can be measured.
  - Feedbacks from the previous iterations: situation can be changed.
  - The stakeholders can influence the shape of the development effort.
  - The combination of the iterative development and the focus on risks in RUP, allows projects an iterative risk-assessment.

6 Key Principles of Business-Driven Development

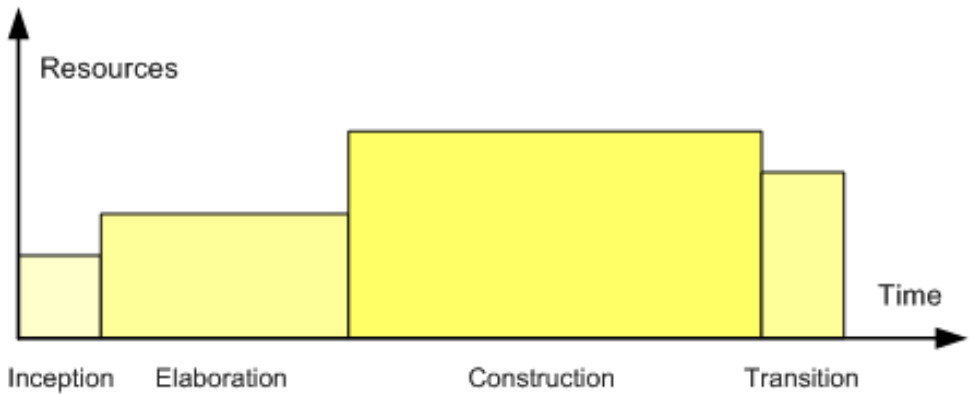
- Elevate the level of abstraction
  - Use reuseable software / tools (for example, design patterns, 4GL, etc.)
  - This can lead to custom-made software code.
  - A higher level of abstraction allows discussions on different architectural levels.
- Focus continuously on quality
  - Quality checks are not only at the end of each iteration
  - Quality checks must be a continuous ongoing activity, checks should be often performed in a daily rhythm supported by the entire team.
  - scripts and test automation can help to test more and more due to iterative development

Phases

A project has four phases:

1. Inception phase
2. Elaboration phase
3. Construction phase
4. Transition phase

A typical project profile showing the relative sizes of the four phases:



Inception phase

- The business case and financial forecast is established in this phase.
- Needed: basic use case model, project plan, initial risk assessment and project description (the core project requirements, constraints, key features, architecture opportunities)

Inception phase

In this phase the project is checked against the following criteria:

- Stakeholder concurrence on scope definition and cost / schedule estimates.
- Requirement understanding as evidenced by the fidelity of the primary use cases.
- Credibility of the cost / schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Establishing a baseline by which to compare actual expenditures versus planned expenditures.

Elaboration phase

- The project starts to take shape.
- Domain analysis is made.
- The architecture of the project gets its basic form.
- The most important analysis is the system architecture.

Elaboration phase

In this phase the project is checked against the following criteria:

- A use-case model in which the use-cases and the actors have been identified and most of the use-case descriptions are developed. The use-case model should be 80% complete.
- A description of the software architecture in a software system development process.
- An executable architecture that realizes architecturally significant use cases.
- Business case and risk list which are revised.
- A development plan for the overall project.
- Prototypes that demonstratively mitigate each identified technical risk.

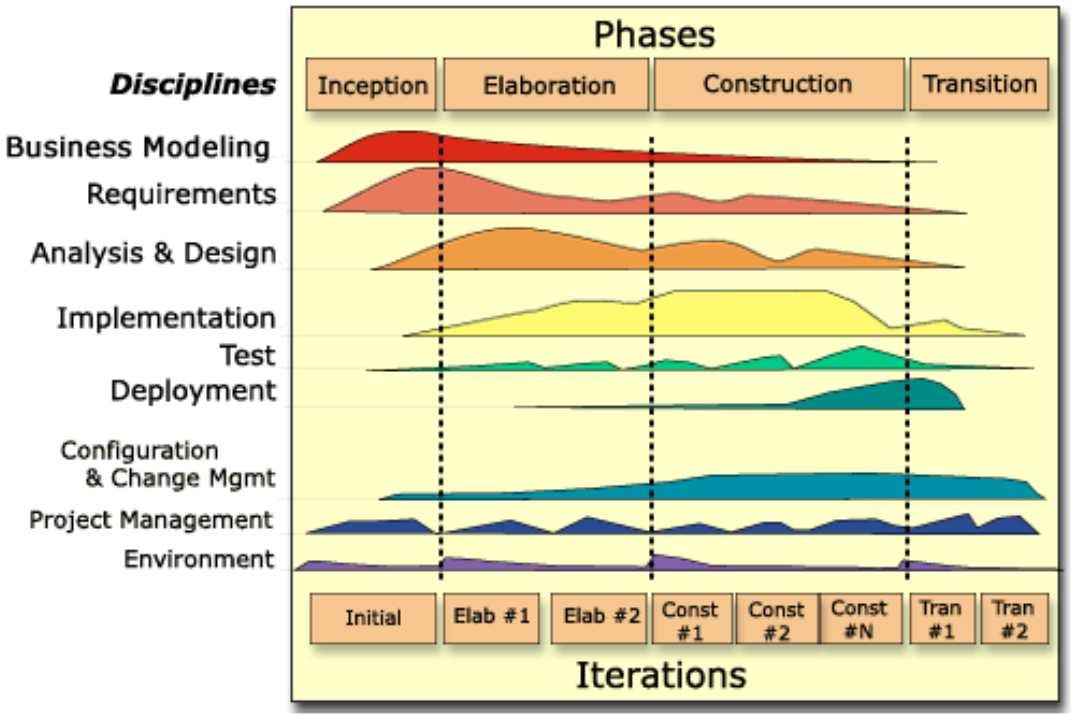
Construction phase

- The main focus is on the development of components.
- The bulk of the coding takes place in this phase.
- In larger projects more construction iterators can be completed and many demonstrable prototypes should be produced.

Transition phase

- The product has moved from the developers to the end user.
- Training of the end user is in this phase.
- Validate the product against the end users' expectations.
- CDs and reference manuals are manufactured in this phase.
- The product is tested against the quality level set in the Inception phase.

Phases


Phases
Disciplines
Inception
Elaboration
Construction
Transition
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration
& Change Mgmt
Project Management
Environment
Initial
Elab #1
Elab #2
Const #1
Const #2
Const #N
Tran #1
Tran #2
Iterations

Extreme Programming (XP)

- Lightweight model
- prescribing a set of daily stakeholder practices that embody and encourage particular XP values

XP values

- Communication: give all developers a shared view of the system which matches the view held by the users of the system.
- Simplicity: Extreme Programming encourages starting with the simplest solution. Extra functionality can then be added later. Always use the simplest solution that works.
- Courage:
  - always design and code for today and not for tomorrow.
  - feel comfortable with refactoring the code when necessary.
  - throw away existing, but obsolete code.

XP values

- Feedback
  - Feedback from the system: by writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.
  - Feedback from the customer: The functional tests (aka acceptance tests) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
  - Feedback from the team: When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.
- Respect

XP values

- Respect
  - team members respect each other
  - Members respect their work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.

Activities

- Coding: the code expresses ideas in a brief, clear way. Software cannot be without code.
- Testing: Code must pass all tests.
- Listening: Developers do not know anything about the business logic, therefore developers have to ask, and answers must be heard.
- Designing: The structure of the software must be planned.

Practicies

- Fine scale feedback
  - Pair programming: one code in computer is done by two programmers
  - Planning Game: it is a game between the customer and developers for well-specificated exercises.
  - Test Driven Development: unit tests before coding. The customer also writes special tests.
  - Whole team: one of the customers or end users should be the member of the development team.

Practicies

- Continuous process
  - Continuous Integration: The integration and building of the system can be executed more times a day.
  - Design Improvement: Refactoring for removing the repeatations and robustusness and simplification.
  - Small Releases: A release should be done within 1-2 months. A whole usable system's part must be released.
- Programmer welfare
  - Sustainable Pace: No overtime at all, or minimize it.

Practicies

- Shared understanding
  - Coding Standards: for consistent code.
  - Collective Code Ownership: everybody may change the code in the entire project.
  - Simple Design: the simplest plans are needed. Remove all the unnecessary elements from it.
  - System Metaphor: A brief description controls the attributes of the system.

A lifecycle of an XP project

1. Inception
2. Designing
3. Iterations before the first release
4. Production
5. Maintenance
6. Finishing