Plan-based Software Development

Software Engineering

2004-2005

Marco Scotto (Marco.Scotto@unibz.it)
Development Models

- Introduction
  - The Waterfall Development Model
  - The V-Shaped Software Development Model
  - The Incremental Software Development Model
  - A synoptic view
Plan-based Development

- Addresses problems in posed by 3 beasts
- Based on the idea that the problems of software development are due to a lack of order
- Order requires finding what to do (planning) and then doing it
- Plans involve:
  - tasks,
  - the deliverables of those tasks, and
  - the dependencies among the tasks that are often related to those deliverables
- High level of adherence to process
Changes...

- A change in the process is intrinsically bad in a plan-based model
  - it causes an alteration of the plan and that might affect the overall architecture of the project.

- To eliminate the need for change, more effort is required by the software developer
  - better planning
  - a higher level of adherence to the development process
  - the automation of plan management...
Plan-based Development

- Waterfall model
- V-shaped model
- Incremental model

Software Engineering
Development Models

- Introduction
- The Waterfall Development Model
- The V-Shaped Software Development Model
- The Incremental Software Development Model
- A synoptic view
Waterfall development model

- Different phases of software development are defined first and then ordered in a strictly sequential way.

- Several instances of the waterfall software development model differ according to how the phases are organized.

- In some cases there may be more emphasis on:
  - customer communications
  - on the design document, or
  - on the system architecture
  - ...

Software Engineering 7
Waterfall model

- Properties and features
- Functional decomposition models
- Cowboy coding

Intention of producing organized systems backfires with resulting chaotic structures
Properties and features

- Product passed from stage to stage
- Division and specialization of labor
- Matches with sequential coordination mechanism
  - Strict end-to-start dependencies
  - Magnifies the problem of unclear deliverables
- Strict and detailed planning to tame the beasts
  - Uncertainty is hard to avoid
  - Irreversibility cannot be ignored if uncertainty exists
- Customers are not able to view the product development
Division and specialization of labor

- Aiming for perfection and economies of scale
- Specific professionals for specific stages
- Progresses from generic to detailed
- Strict and detailed planning to tame the beasts
- Clear and well defined deliverables as interfaces between stages
  - No human in deliverables \(\rightarrow\) more robust
- Simplified management & clear responsibilities
The 3 beasts

- Strict and detailed planning to tame the beasts
  - Uncertainty is hard to avoid
  - Irreversibility cannot be ignored if uncertainty exists

<table>
<thead>
<tr>
<th>Beast</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty</td>
<td>A very detailed planning phase in which all the details are sorted out.</td>
</tr>
<tr>
<td>Irreversibility</td>
<td>Not really managed within the process. Almost all decisions are considered irreversible.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Clear division of tasks among and within phases. Formal document interfaces.</td>
</tr>
</tbody>
</table>
Evaluation of the waterfall model

- First methodology to guide the development of a software system
- Pitfalls
  - It is not possible to foresee the appearance of all possible features of a software system
  - Work specialization may not work well in software nowadays
  - Lack of integration between phases
  - Deliverables often not fully understood by the next stage
    - Bugs or problems of deliverables are overlooked
    - Sequential dependency magnifies the problem
- Implications for Coding
  - Modifying the code regardless of what is written in the upstream deliverables
    - “Write Once Read Never” documents
Functional decomposition models

- Associated to waterfall models
- Similarities
  - Superimposing suitable structure to project
  - Belief in perfect planning and precise prediction
- Differences
  - Functional decomposition: divide and conquer
  - Waterfall: linear and easily decomposable structure
Cowboy coding

- Just patching the bugs and hacking in code
- Spaghetti code
  - Messy
  - Result of cowboy coding
- Everything is tied together
- Slight modification causes unpredictable changes
Alternatives to the Waterfall Model – needed? Why?

- Not possible to develop a unique formalism for defining requirements, analysis, design, and code
  - Unique model missing – unambiguous, complete, correct

- The product appears only at the very end and then it is shipped to the customer
  - The customer must proceed on an act of faith that the development is going in the right direction
Development Models

- Introduction
- The Waterfall Development Model
- The V-Shaped Software Development Model
- The Incremental Software Development Model
- A synoptic view
V-shaped model

- Develop correctness criteria for each stage
- Sequential (preferred) or common output mechanism
- Verification vs. validation
- Addressing the 3 beasts
- Some issues of waterfall model yet to be addressed
- Investment structure
The Structure of the V-Shaped Development Model

At the end of each phase there is the definition of a test of the system as it is perceived then...
Develop correctness criteria for each stage

- Correctness criteria = test case
- Two interfaces between contiguous phases
  - Deliverable
  - Test case
- Checking for errors at the end of development
  - Checking proceeds from more specific to less specific issues
Interfaces in the V-Shaped Model

In the V-shaped models there are two interfaces between contiguous phases:

- A document is created with the details of the product at the conclusion of each phase (requirement document, analysis document, design document, code)
  - The document triggers the flow down in the model, toward more specialization

- A document is created with the acceptance criteria for a more specific phase to the more generic phase – the test cases for customer acceptance, the test cases for system integration, and so on
  - This document ensures that the more detailed phase has satisfied the need of the less detailed phase
Coordination Mechanisms in the V-Shaped Model

- Verification vs. validation
  - Verification: checking against customer's original requirements
  - Validation: checking against previous phase

- Sequential (preferred) or common output mechanism
  - Strict end-to-start dependencies
Addressing the 3 beasts

- Uncertainty and irreversibility are considered with the introduction of test cases.
- An error in a phase implies rework of all the following phases.
Addressing the 3 beasts

<table>
<thead>
<tr>
<th>Beast</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty</td>
<td>There is very detailed planning phase in which all the details are sorted out. The possibility of misunderstanding and errors is taken into consideration via the insertion of checks against each previous phase.</td>
</tr>
<tr>
<td>Irreversibility</td>
<td>Irreversibility is now considered. Errors exist. Finding an error in a phase implies the invalidity of everything done after it, which, in principle, requires a complete rework.</td>
</tr>
<tr>
<td>Complexity</td>
<td>Complexity is handled analogously to the handling in waterfall model, with the addition of the checks of validity for the movements upstream.</td>
</tr>
</tbody>
</table>
Evaluation of the V-shaped Model

**Successes**
- The control of correctness beyond the scope of simple testing of the code and before the final deliverable to the customer

**Pitfalls**
- We cannot be sure that the different phases are properly interconnected
- All the development process is based on the same sequential concepts of planning, specialization of skills, and modularization of development
  - The structure of development is not intended for concepts that will evolve in the course of development
- The product appears only at the end; the customer cannot see it growing step-by-step
Development Models

- Introduction
- The Waterfall Development Model
- The V-Shaped Software Development Model
- The Incremental Software Development Model
- A synoptic view
Incremental model

- Based on first developing the overall system architecture and then building in increments subsystems fitting that architecture.
Activities in the incremental model

The incremental model is based on two kinds of activities:

- On the one hand:
  - the capture of requirements,
  - the analysis of the system,
  - the development of a product architecture, and
  - the division of the product architecture into a set of independent parts.

- The development of each independent part quite like a small V-shaped model.
New features

*Compared to waterfall and V-shaped models*

- Increments
- Architecture design phase
- Reuse library
Increments

- Subsystems; independent parts
- Independent development
  - Analogous to small V-shaped models
  - Mutual references and parallel progress via clear interfaces
  - No direct and unrestricted access to each other's part
- Can be shown to customers
- Require regular deliverable and system architecture as inputs
Upper portion of the incremental model

- Requirement capture
  - Requirement document
  - Test cases for customer acceptance

- Analysis
  - Analysis document
  - Test cases for system integration

- Arch. design
  - System architecture
  - Test cases for unit integration

- Unit Integration

- System integration

- Delivery to customer

Very generic

Level of detail

Very detailed

Calendar time

Software Engineering
Lower portion of the incremental model

- System architecture
- Analysis document
- Design document
- Design
- Coding
- Code
- Test cases for unit integration
- Unit Integration
- Test of Code
- Test cases for code

Software Engineering
The big picture

Software Engineering

Requirement capture

Test cases for customer acceptance

Very generic

Analysis

Test cases for system integration

Level of details

Architecture design

Test cases for unit integration

Very detailed

System integration

Delivery to customer

Iteration 1

Design

Coding

Test of Code

System architecture

Analysis document

Iteration 2

Design

Coding

Test of Code

System architecture

Analysis document

Iteration n

Design

Coding

Test of Code

System architecture

Analysis document

Calendar time
Architecture design phase

- Produces a deliverable called system architecture
- System architecture
  - Provides the overall model of solution
  - Identifies increments
  - Used by other phases and increments
Addressing the 3 beasts (1/2)

- Uncertainty, irreversibility, and overall complexity reduced due to smaller scope of increments
- Higher planned coordination complexity
### Addressing the 3 beasts (2/2)

<table>
<thead>
<tr>
<th>Beast</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncertainty</td>
<td>Uncertainty in system development is limited by developing the system increment-by-increment. The uncertainty in customer requirements is not taken into account significantly.</td>
</tr>
<tr>
<td>Irreversibility</td>
<td>A good architecture should specify solid and sound interfaces, thus limiting irreversibility that arises because of errors in development. Irreversibility due to misunderstood requirements is not properly addressed.</td>
</tr>
<tr>
<td>Complexity</td>
<td>The system architecture separates the concerns of the different subsystems and limits the complexity of the overall development. Planned coordination complexity becomes greater, though.</td>
</tr>
</tbody>
</table>
Common output coordination mechanism

- Integration test within each increment
- Corrects problems sooner
- Integration challenges
  - Interaction not limited to external interfaces
  - Extends to internal variables and structures
- Dependencies
  - End-to-start for overall development
  - Start-to-start among increments which may be partially overlapped
Object oriented model

- OO natural match with incremental model
- Reduces problem of integration
- Same language between phases
- Risks
  - Developing structures difficult to understand
  - Building many useless diagrams
Reuse library

Before start of increments

• Reusing elements
  ▪ Increases quality and helps finding defects
  ▪ repeated usage → repeated tests

• Risk of wasting time and effort
  ▪ Functionalities required are not clear at this point

During development of increment

Framework
Reuse library before start of increments

Software Engineering
Reuse library

- **Before start of increments**
- **During development of increments**
  - Increases quality and helps finding defects
  - Essentially a set of utility objects and functions
  - Minimum risk of producing useless components
  - Constantly upgraded and re-factored
- **Framework**
Reuse library during development of increments

- Requirement capture
- Analysis
- Architecture design
- System integration
- Unit Integration
- Delivery to customer

Very generic

Level of details

Very detailed

Software Engineering
Reuse library

- Before start of increments
- During development of increments
- **Framework**
  - "Active library"
  - Has built-in main function which calls the developers' supplied functions
  - No need for developers to worry about overall logic
  - Requires upfront investment
Evaluation

- System easier to manage and more flexible
- Investment diluted throughout development
- Additional feedbacks from earlier increments
- A need to balance
  - More valuable feedbacks from tighter integration
  - Higher coordination complexity
- OO techniques increase integration
- Customers are still left out
Development Models

- Introduction
- The Waterfall Development Model
- The V-Shaped Software Development Model
- The Incremental Software Development Model

- A synoptic view
## A synoptic view

<table>
<thead>
<tr>
<th>Waterfall</th>
<th>V-shaped</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigidity of the overall process</td>
<td>As waterfall</td>
<td>The process becomes more flexible due to the different increments</td>
</tr>
<tr>
<td>Significant upfront investment</td>
<td>Even more upfront investments</td>
<td>After an initial large investment, subsequent investments are diluted in increments</td>
</tr>
<tr>
<td>Difficulty to collect all the desired features from the customer upfront</td>
<td>As waterfall</td>
<td>As waterfall, unless the customer is made aware of the different increments – this may cause other problems, though</td>
</tr>
<tr>
<td>Difficulty to determine the HW/SW specs and predict the HW/SW evolutions of the target environment</td>
<td>As waterfall</td>
<td>The development in increments limits this problem</td>
</tr>
<tr>
<td>Work specialization does not suit most of today SW development situations</td>
<td>As waterfall</td>
<td>Within increments, the development teams may be more integrated</td>
</tr>
<tr>
<td>Lack of integration between phases</td>
<td>Due to the introduction of tests, there are more earlier checks for integration between phases</td>
<td>As V-shaped. If Object Orientation is used, the integration becomes much higher</td>
</tr>
<tr>
<td>When there is a panic, people focus on the code; the other artifacts produced during development – analysis, design, ... may soon become obsolete</td>
<td>As waterfall</td>
<td>As waterfall, unless the customer is made aware of the different increments – this may cause other problems, though</td>
</tr>
<tr>
<td>Customer cannot perceive the evolution of the product</td>
<td>As waterfall</td>
<td></td>
</tr>
</tbody>
</table>