The Software Life Cycle

Software Engineering
Andreas Zeller • Saarland University

Planning
Modelling
Construction
Transition
Deployment
Communication

Software Increment

Inception
Elaboration
Construction
Transition
Production

Sign up!

http://www.st.cs.uni-saarland.de/edu/se/2012/
Select Projects
Thursday, 18:00 – Monday, 12:00

The Software Life Cycle
Software Engineering
Andreas Zeller • Saarland University

A Software Crisis
Denver International Airport (DIA)
Construction started in 1989 • 53 sq miles
• Planned: 1.7 bio USD costs, opening 1993
Code and Fix
(1950–)

- Build first version
- Modify until client is satisfied
- Operate
- Retirement

Code and Fix: Issues

- No process steps – no specs, docs, tests…
- No separation of concerns – no teamwork
- No way to deal with complexity
Waterfall Model
(1968)

Communication
- project initiation
  - requirements gathering

Planning
- estimating
- scheduling
- tracking

Modeling
- analysis
- design

Construction
- code
- test

Deployment
- delivery
- support
- feedback

Communication

6.6 Map Service Tool

Live Map Description

Map displays a set of map layers that the user can select from. Each map layer contains information about a specific geographic area. The user can select one or more layers to display on the map.

Sample Map Layers:
- Road Network
- Land Cover
- Topography
- Water Bodies

Interaction:
- User selects a map layer
- Map updates to display selected layer
- User can toggle layers on or off

Summary:
- Map service provides a flexible and intuitive way for users to explore geographic information
- Suitable for a wide range of applications, from urban planning to natural resource management

Waterfall Model
(1968)
Planning

Waterfall Model (1968)

Communication
- project initiation
- requirements

Planning
- planning
- estimating
- scheduling
- tracking

Modeling
- analysis
- design

Construction
- code
- test

Deployment
- delivery
- support
- feedback
Waterfall Model
(1968)

Communication
project initiation
requirements gathering

Planning
estimating
scheduling
tracking

Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

Waterfall Model

Construction
code
test

Order = order
order = order

do while (order = order)

print(order)
endif

/* allocate 'orders' elements
 it is used to store

Waterfall Model
(1968)
• Real projects rarely follow a sequential flow
• Hard to state all requirements explicitly
• No maintenance or evolution involved
• Customer must have patience
• Any blunder can be disastrous
Boehm’s first law

Errors are most frequent during requirements and design activities and are the more expensive the later they are removed.

Problem Cost

<table>
<thead>
<tr>
<th>Phase</th>
<th>Relative cost of problem per phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>0</td>
</tr>
<tr>
<td>Unit test</td>
<td>7.5</td>
</tr>
<tr>
<td>Component test</td>
<td>15.0</td>
</tr>
<tr>
<td>System test</td>
<td>22.5</td>
</tr>
<tr>
<td>Field</td>
<td>30.0</td>
</tr>
</tbody>
</table>

Incremental Model

This and other laws are found in
Evidence: Several studies before 1974
Incremental Model

- Each linear sequence produces a particular “increment” to the software
- First increment typically core product; more features added by later increments
- Allows flexible allocation of resources

Prototyping

Communication → Quick Plan

Quick Design → Deployment and Feedback

Prototype Construction → Quick Design

Prototypes

<table>
<thead>
<tr>
<th>Top Layer (GUI)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bottom Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Horizontal Prototype

Prototypes

Vertical Prototype
Prototypes

- A *horizontal prototype* tests a particular *layer* (typically the GUI) of the system
- A *vertical prototype* tests a particular *functionality* across all layers
- Resist pressure to turn a prototype into a final result!

**Spiral Model**

*(1988)*

- System is developed in series of evolutionary releases
- Milestones for each iteration of the spiral
- Process does not end with delivery
- Reflects iterative nature of development
Unified Process
(1999)

- Inception
  - Encompasses communication with user + planning
  - Results in a set of use cases
  - Architecture is just a tentative outline

Elaboration
- Refines and expands preliminary use cases
- Provides architecture and initial design model
Construction

• Builds (or acquires) software components according to architecture
• Completes design model
• Includes implementation, unit tests, acceptance tests

Transition

• Software given to end users for beta testing
• Feedback reports defects and changes
• Support information written

Production

• Software is deployed
• Problems are monitored
Re-Iteration

- Feedback results in new iteration for next release

Unified Process

- Draws on best features of conventional process models
- Emphasizes software architecture and design
- Integrates with UML modeling techniques (more on this later)
If a traditional process is like a battleship, protected against everything that might happen…

an agile process is like a speedboat, being able to change direction very quickly

---

Manifesto for Agile Software Development (2001)

- Individuals and activities over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.
What is Agile Development?

- Agility = ability to react to changing situations quickly, appropriately, and effectively.
  - notice changes early
  - initiate action promptly
  - create a feasible and effective alternative plan quickly
  - reorient work and resources quickly and effectively

Agile?

Communication
- project initiation
- requirements gathering

Planning
- estimating
- scheduling
- tracking

Modeling
- analysis
- design

Construction
- code
- test

Deployment
- delivery
- support
- feedback

Incremental Model

Features

Increment #1
- Communication
- Planning
- Modeling
- Construction
- Deployment

Increment #2
- Communication
- Planning
- Modeling
- Construction
- Deployment

Increment #3
- Communication
- Planning
- Modeling
- Construction
- Deployment

Time
Agile vs. Plan-driven

<table>
<thead>
<tr>
<th>Agile</th>
<th>Plan-driven</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low criticality</td>
<td>High criticality</td>
</tr>
<tr>
<td>Senior developers</td>
<td>Junior developers</td>
</tr>
<tr>
<td>Requirements change very often</td>
<td>Requirements don't change too often</td>
</tr>
<tr>
<td>Small number of developers</td>
<td>Large number of developers</td>
</tr>
<tr>
<td>Culture that thrives on chaos</td>
<td>Culture that demands order</td>
</tr>
</tbody>
</table>

What is an Agile Process?

- Difficult to predict which requirements will persist or change in the future.
- For many types of software, design and development are interleaved.
- Analysis, design, construction, and testing are not as predictable.
So, how to tackle unpredictability?

make the process adaptable...

Extreme Programming
(1999–)

Planning

Design

Test

Coding

Software Increment

• In XP, planning takes place by means of stories
• Each story captures essential behavior

Planning

1. I want to be able to create some buildings from within the Customer object, where the Customer object is able to set up and tear down the building.
2. The Customer object will be able to change the design of a building without disturbing the presentation of the building.
3. I want my Payment Method and Telephone number to be called directly with the Customer object, so I can use them in a function. Where there is multiple Payment Methods and Telephones, I want the customer to be able to specify which is the preferred one.
4. I want to be able to create some buildings that will be used by the Customer, as if you were using them in a .NET building. Where there are multiple Payment Methods and Telephones, I want the Customer object to be able to specify which is the preferred one.
5. I want a City object to hold a list of customers located by a Postcode, Street Address, or City. The address with a number 1 is being used to create a City object to hold a list of friends located by a City, or a City.

Software Increment
Extreme Programming

• Design is made on the fly, using the KISS (keep it simple) principle
• Virtually no notation besides CRC cards (object sketches) and spike solutions (prototypes)
Each story becomes a unit test that serves as specification.
The program is continuously refactored to have the design match the stories.

To ensure continuous review, XP mandates pair programming.
Testing

Unit tests
- detect errors
- find missing functionality
- measure progress

Extreme Programming

- The resulting prototypes result in new stories

Extreme Programming is fast – with multiple deliverables per day!
So, aren’t agile techniques just “code and fix” in disguise? Why not? (Hint: Think about explicit requirements, and explicit quality assurance)

(it’s ∞ iterations only if you are very, very successful)

Your Typical Life Cycle

- 2 iterations for requirements
- 3 iterations for use cases
- 4–5 iterations for GUI design
- 2 iterations for models
- 2–∞ iterations for prototype

13 iterations total!
Summary