The Software Life Cycle

Software Engineering
Andreas Zeller • Saarland University

1

A Software Crisis

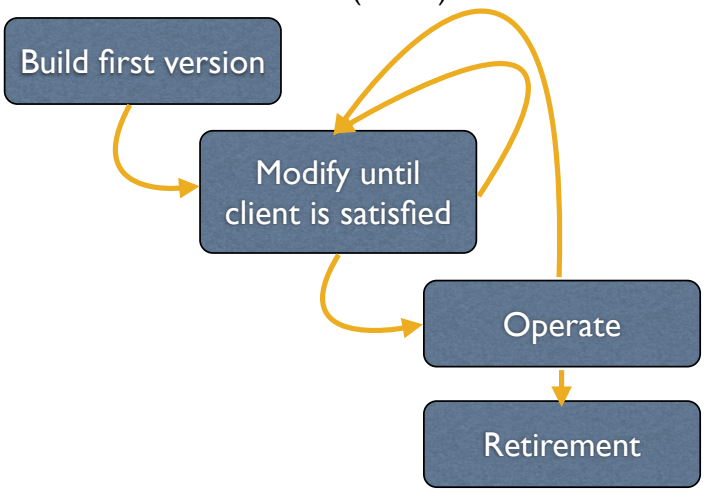
Denver International Airport (DIA)

Construction started in 1989 • 53 sq miles • Planned: 1.7 bio USD costs, opening 1993

2

Code and Fix

(1950–)



3

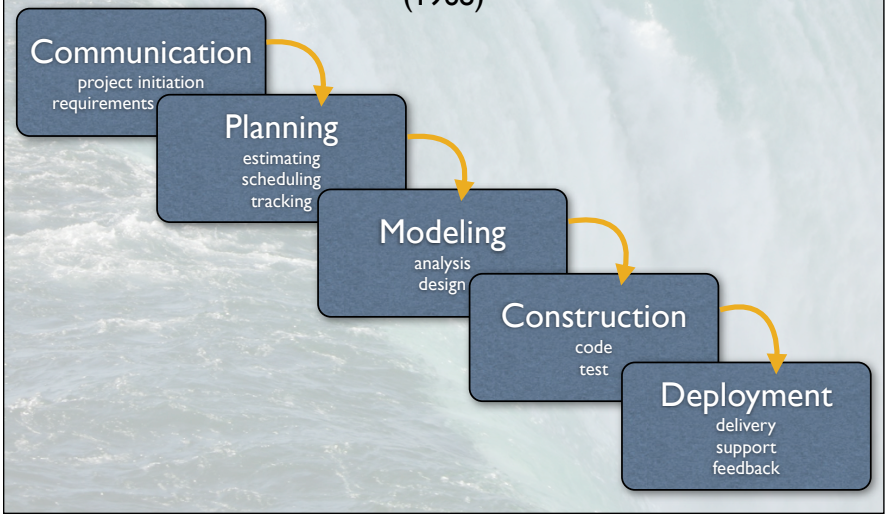
Code and Fix: Issues

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

Code and Fix

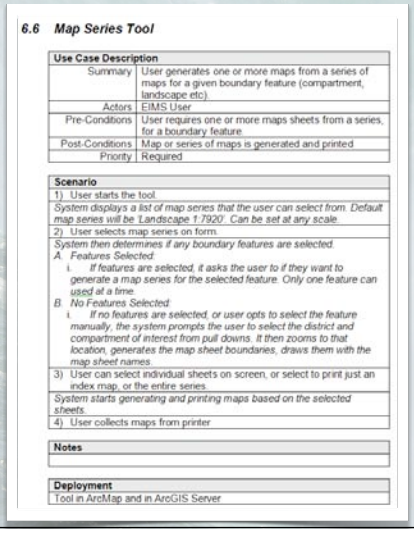
Waterfall Model

(1968)



Communication

Communication
project initiation
requirements gathering



6.6 Map Series Tool

Use Case Description	
Summary	User generates one or more maps from a series of maps for a given boundary feature (compartment, landscape etc).
Actors	EIMS User
Pre-Conditions	User requires one or more maps sheets from a series, for a boundary feature
Post-Conditions	Map or series of maps is generated and printed
Priority	Required

Scenario
1) User starts the tool.
System displays a list of map series that the user can select from. Default map series will be 'Landscape 1:7920'. Can be set at any scale.
2) User selects map series on form.
System then determines if any boundary features are selected. *A. Features Selected* *i. If features are selected, it asks the user to if they want to generate a map series for the selected feature. Only one feature can used at a time.* *B. No Features Selected* *i. If no features are selected, or user opts to select the feature manually, the system prompts the user to select the district and compartment of interest from pull downs. It then zooms to that location, generates the map sheet boundaries, draws them with the map sheet names.*
3) User can select individual sheets on screen, or select to print just an index map, or the entire series.
System starts generating and printing maps based on the selected sheets.
4) User collects maps from printer.

Notes

Deployment
Tool in ArcMap and in ArcGIS Server

7

Waterfall Model

(1968)

Communication
project initiation
requirements

Planning
estimating
scheduling
tracking

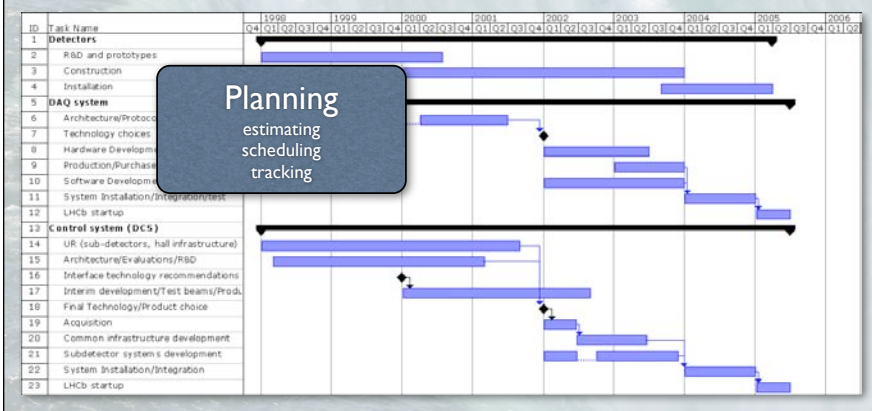
Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

8

Planning



ID	Task Name
1	**Detectors**
2	R&D and prototypes
3	Construction
4	Installation
5	**DAQ system**
6	Architecture/Protocol
7	Technology choices
8	Hardware Developme
9	Production/Purchase
10	Software Developme
11	System Installation/Integration/test
12	LHCb startup
13	**Control system (DCS)**
14	UR (sub-detectors, hall infrastructure)
15	Architecture/Evaluations/R&D
16	Interface technology recommendations
17	Interim development/Test beams/Prod.
18	Final Technology/Product choice
19	Acquisition
20	Common infrastructure development
21	Subdetector systems development
22	System Installation/Integration
23	LHCb startup

Planning
estimating
scheduling
tracking

9

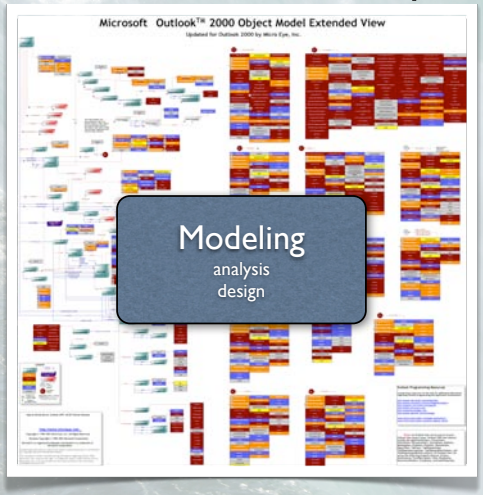
Waterfall Model

(1968)

- **Communication**
  project initiation
  requirements
- **Planning**
  estimating
  scheduling
  tracking
- **Modeling**
  analysis
  design
- **Construction**
  code
  test
- **Deployment**
  delivery
  support
  feedback

10

Waterfall Model (1968)



Modeling
analysis
design

11

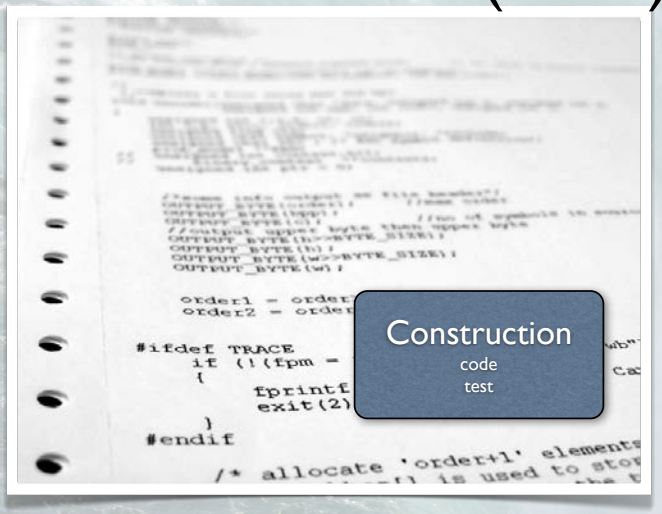
Waterfall Model

(1968)

- **Communication**
  project initiation
  requirements
- **Planning**
  estimating
  scheduling
  tracking
- **Modeling**
  analysis
  design
- **Construction**
  code
  test
- **Deployment**
  delivery
  support
  feedback

12

Waterfall Model (1968)



Construction
code
test

13

Waterfall Model

(1968)

Communication
project initiation
requirements

Planning
estimating
scheduling
tracking

Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

14

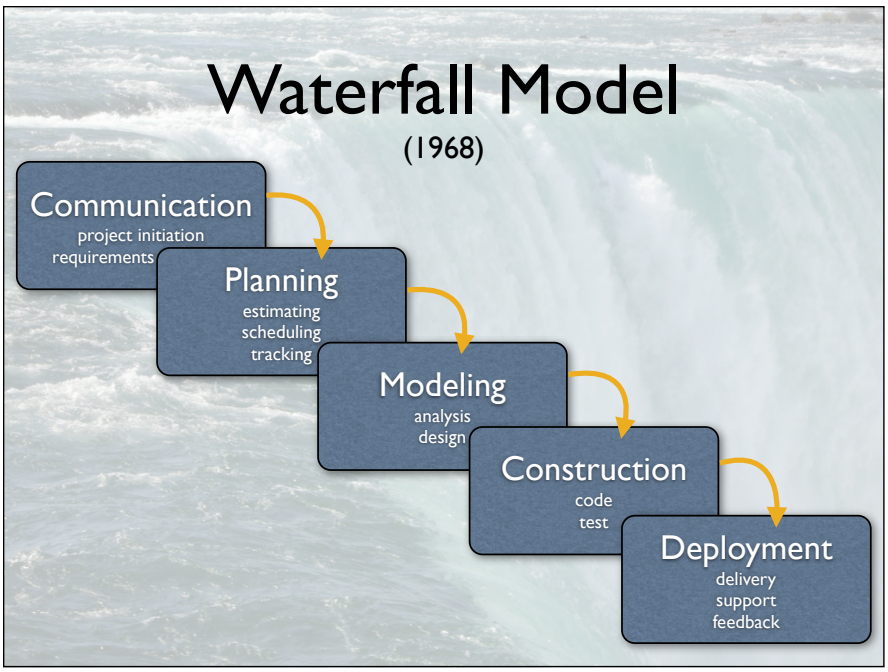
Deployment

Deployment
delivery
support
feedback

15

Waterfall Model

(1968)



16

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

17

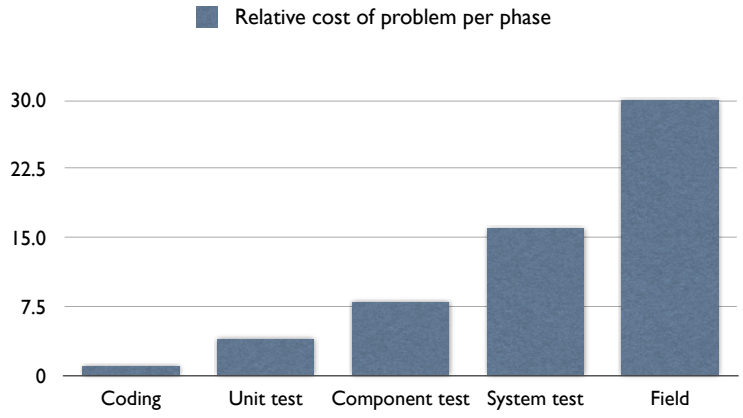
Boehm's first law

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

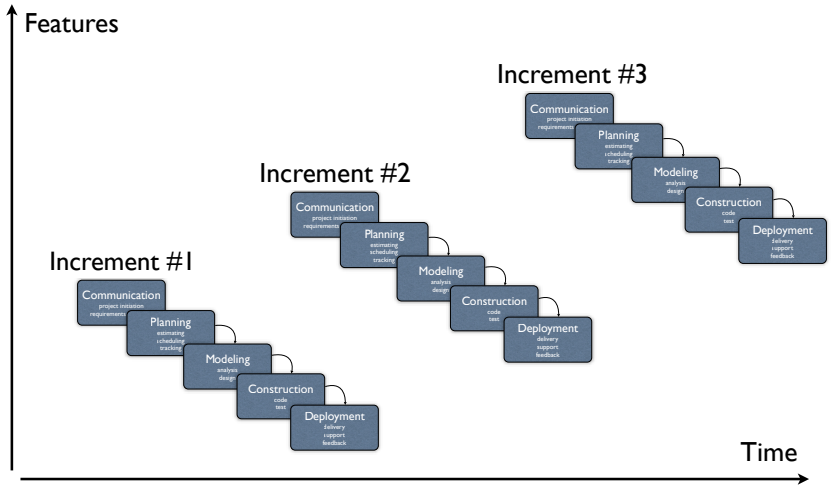
18

This and other laws are found in Endres/Rombach: Handbook of Software and Systems Engineering. Evidence: Several studies before 1974

Problem Cost



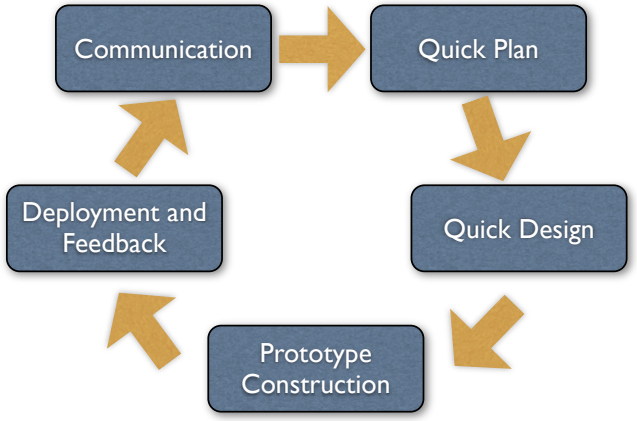
Incremental Model



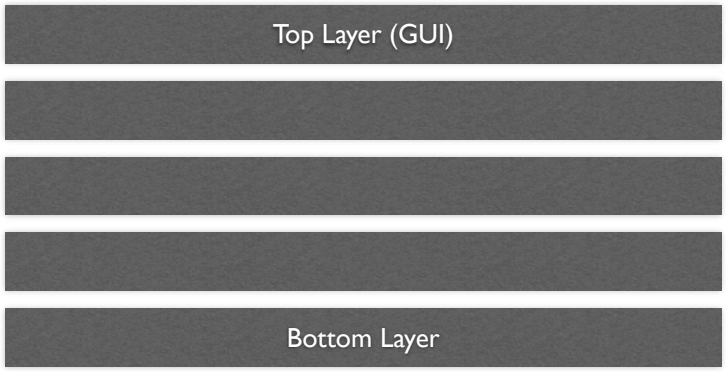
Incremental Model

- 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

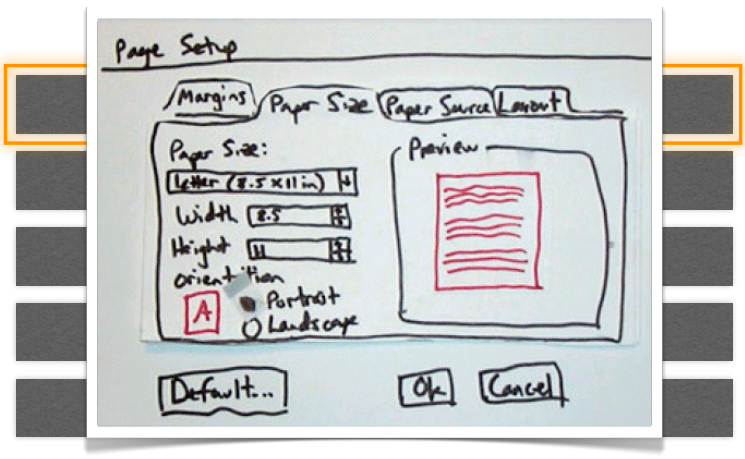
Prototyping



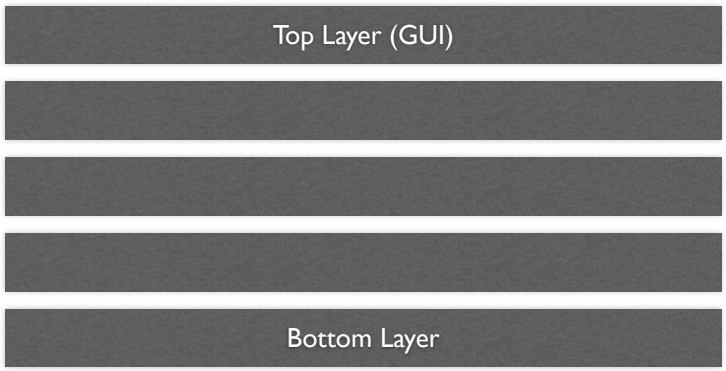
Prototypes



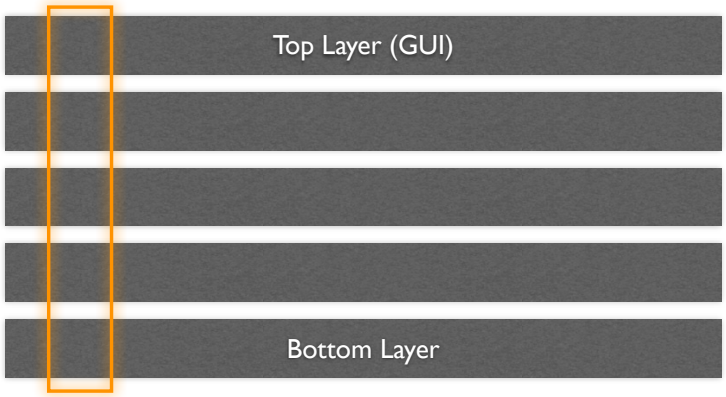
Horizontal Prototype



Prototypes



Vertical Prototype

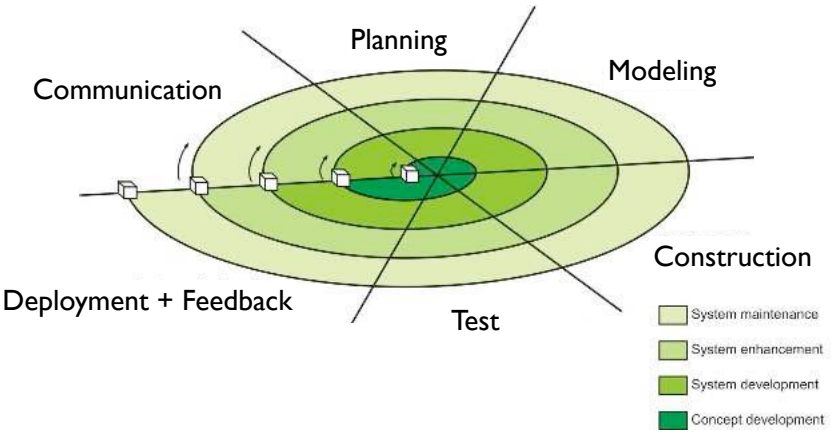


Prototypes

- 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)

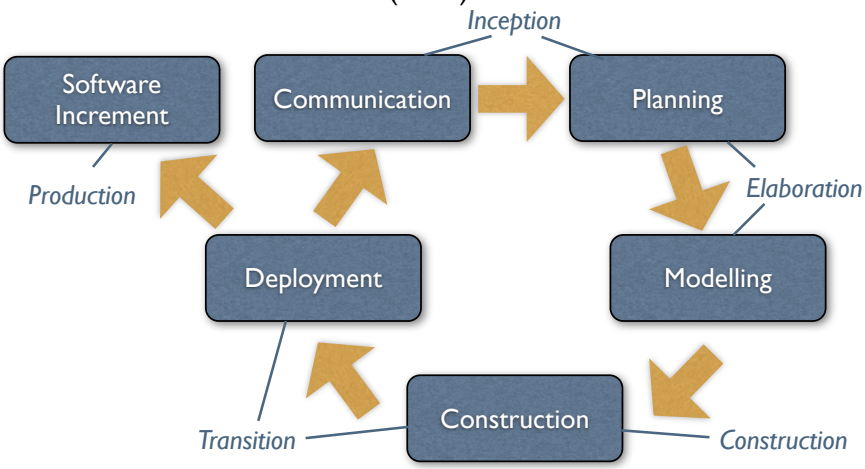


Spiral Model

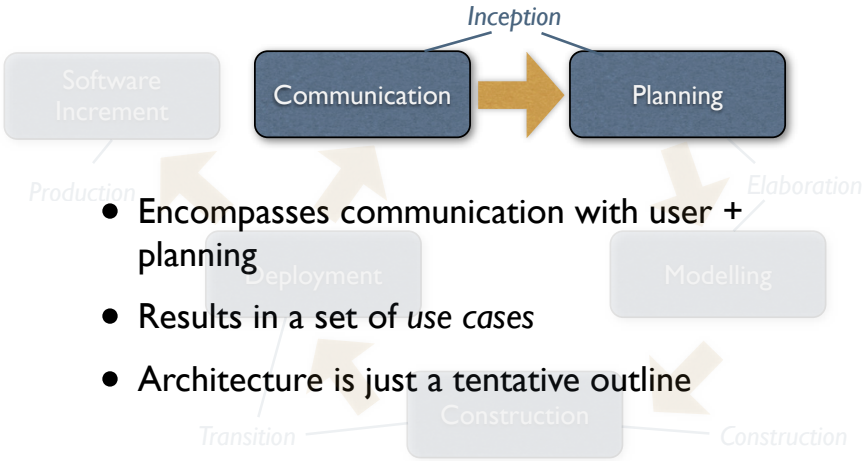
- System is developed in series of evolutionary releases
- Milestones for each iteration of the spiral
- Process does not end with delivery
- Reflects iterative nature of development

Unified Process

(1999)

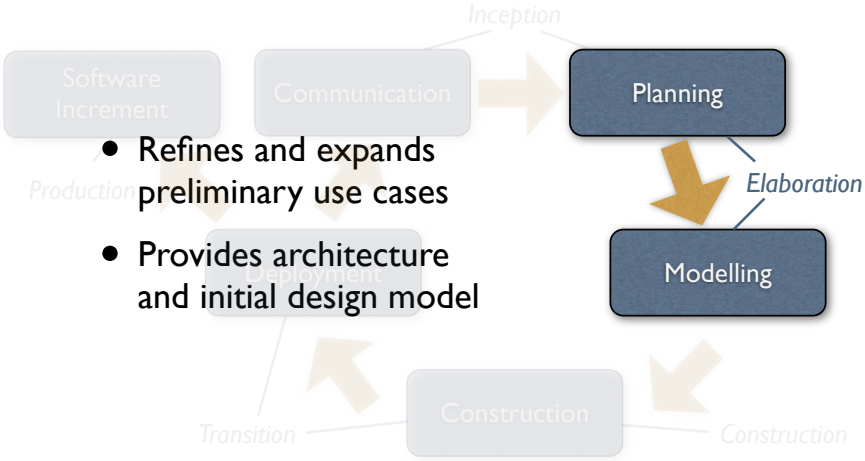


Inception



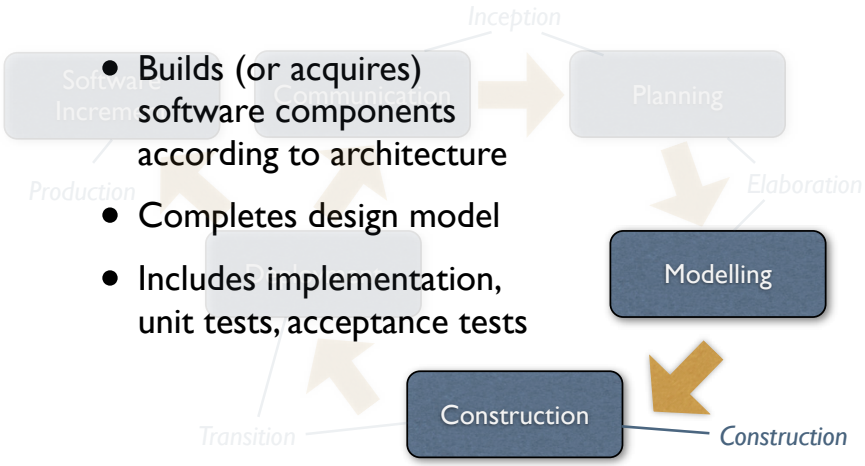
- Encompasses communication with user + planning
- Results in a set of *use cases*
- Architecture is just a tentative outline

Elaboration



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

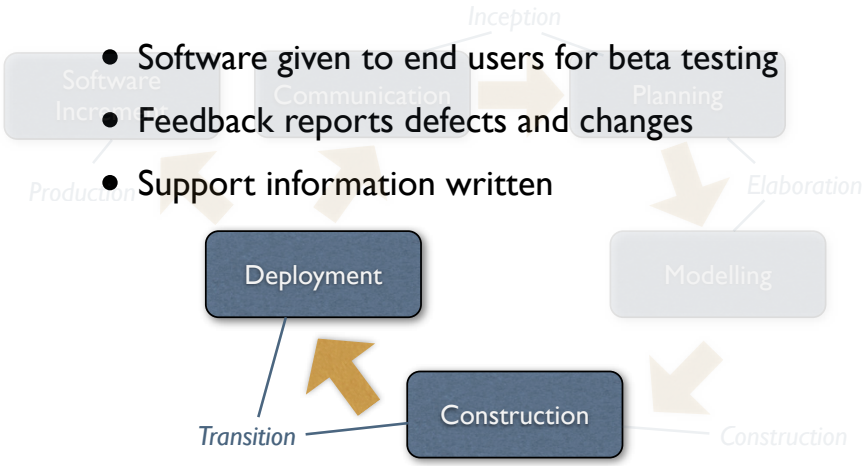
Construction



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

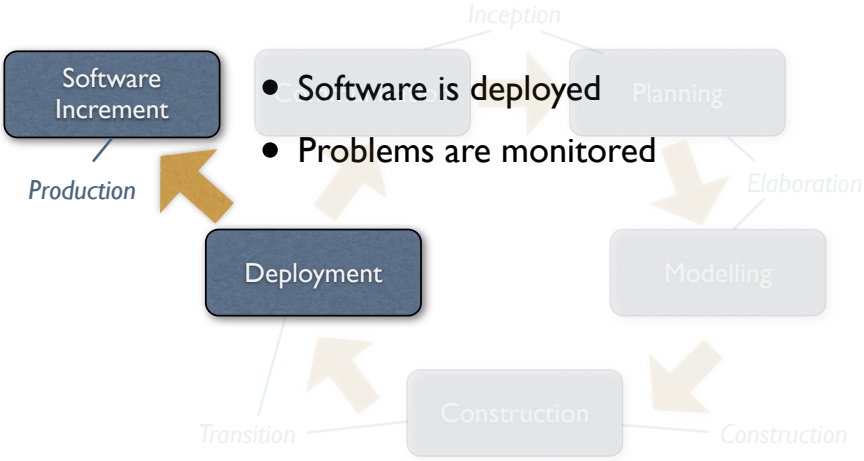
Transition

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



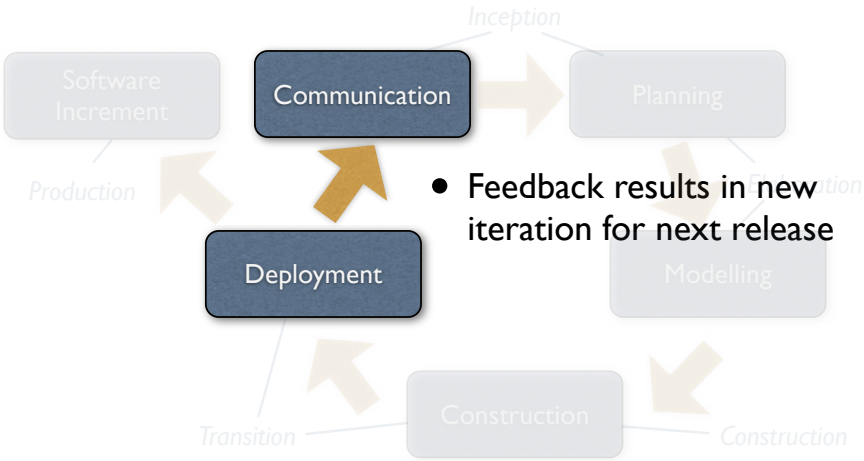
Production

- Software is deployed
- Problems are monitored

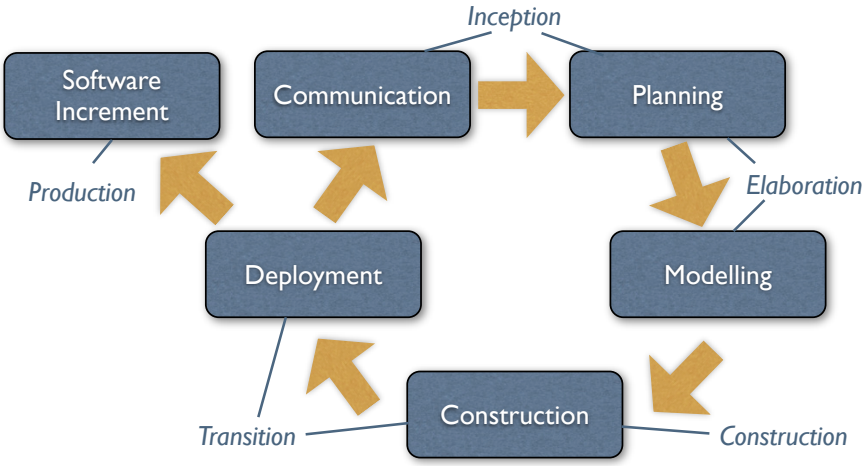


Re-Iteration

- Feedback results in new iteration for next release



Unified Process

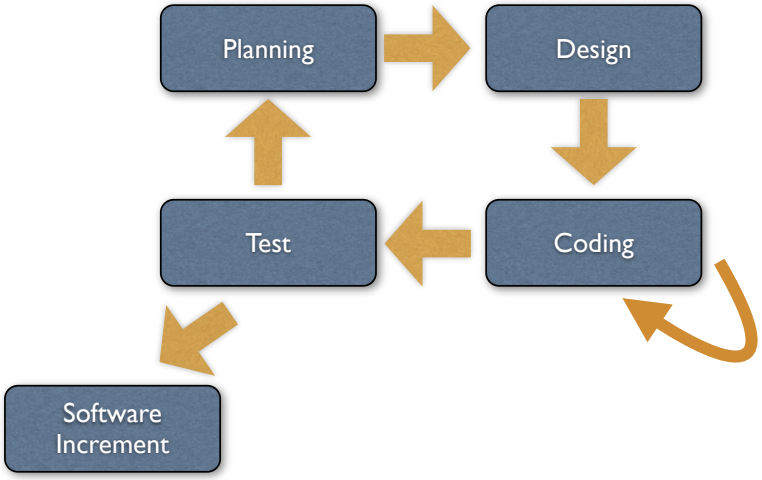


Unified Process

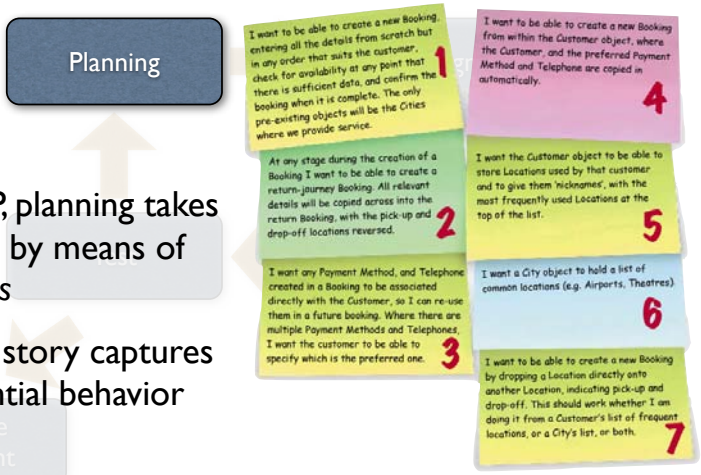
- Draws on best features of conventional process models
- Emphasizes software architecture and design
- Integrates with UML modeling techniques (more on this later)

Extreme Programming

(1999–)



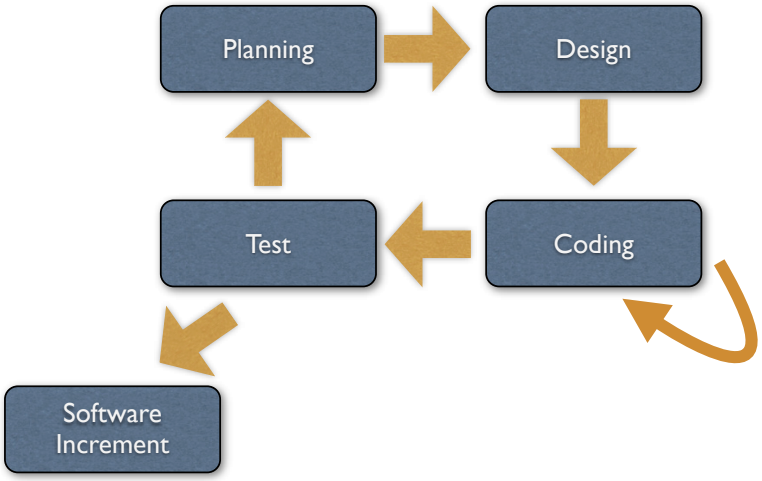
Planning



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

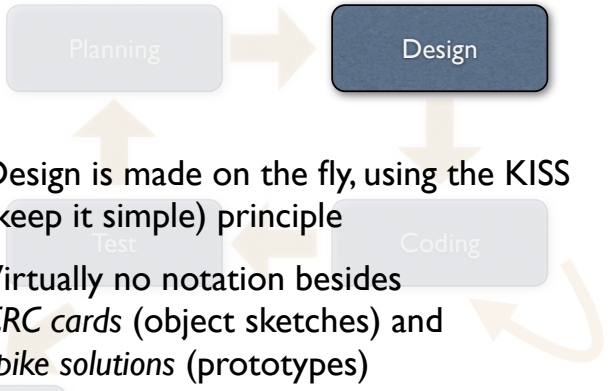
40

Extreme Programming



41

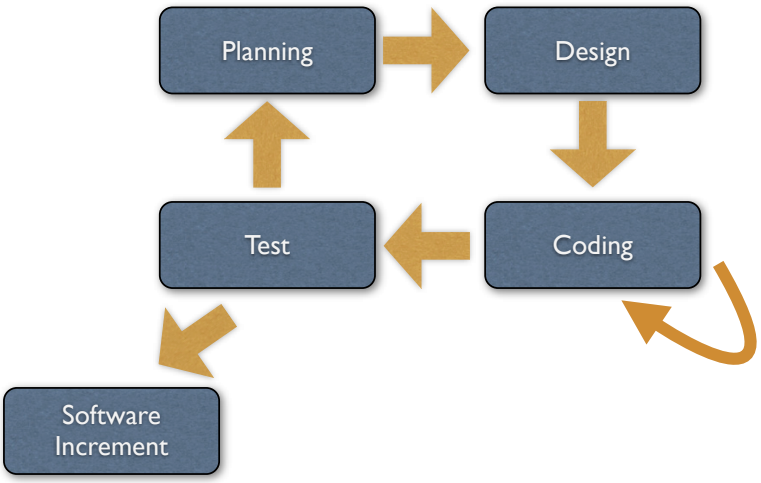
Extreme Programming



- 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)

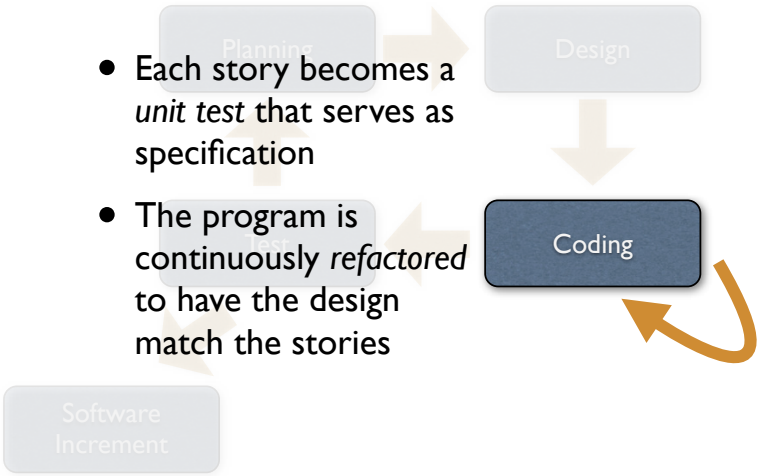
42

Extreme Programming



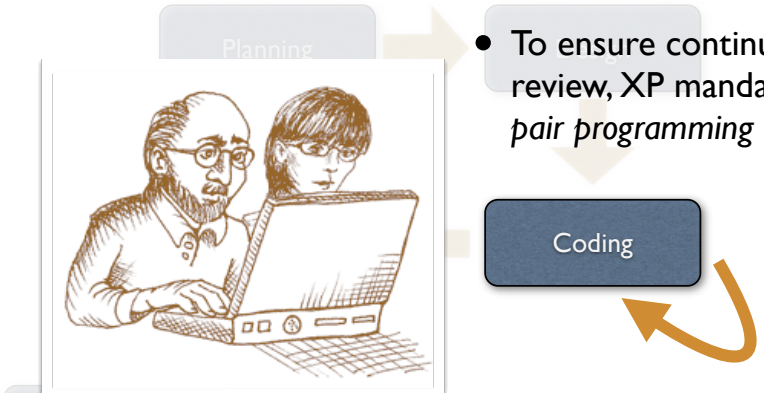
Coding

- Each story becomes a *unit test* that serves as specification
- The program is continuously *refactored* to have the design match the stories

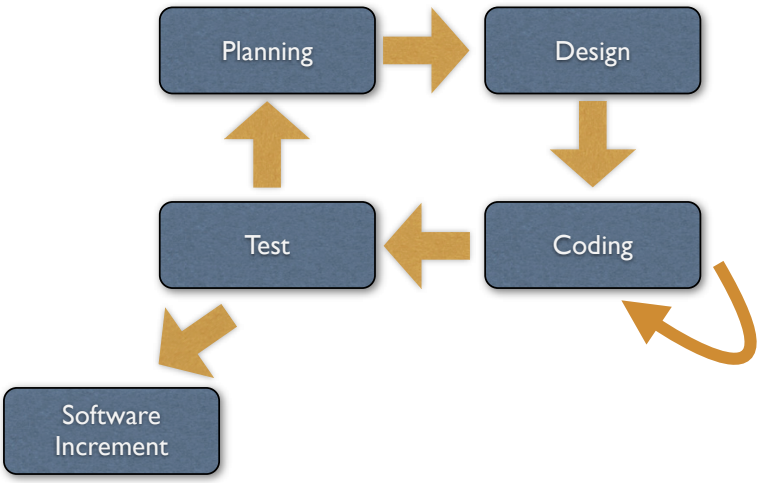


Coding

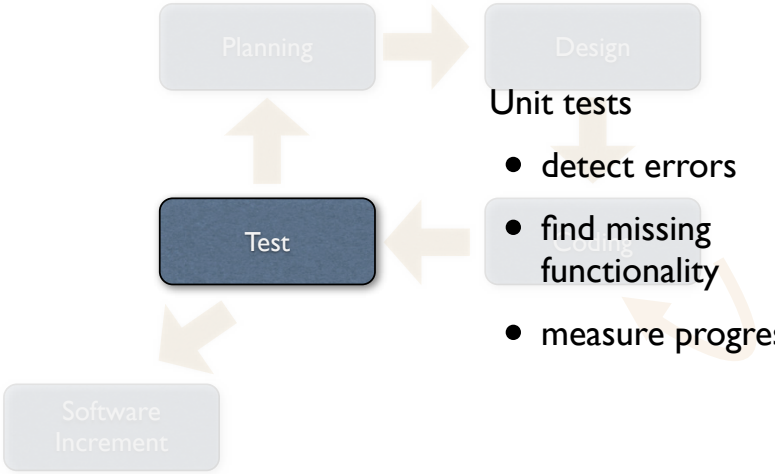
- To ensure continuous review, XP mandates *pair programming*



Extreme Programming



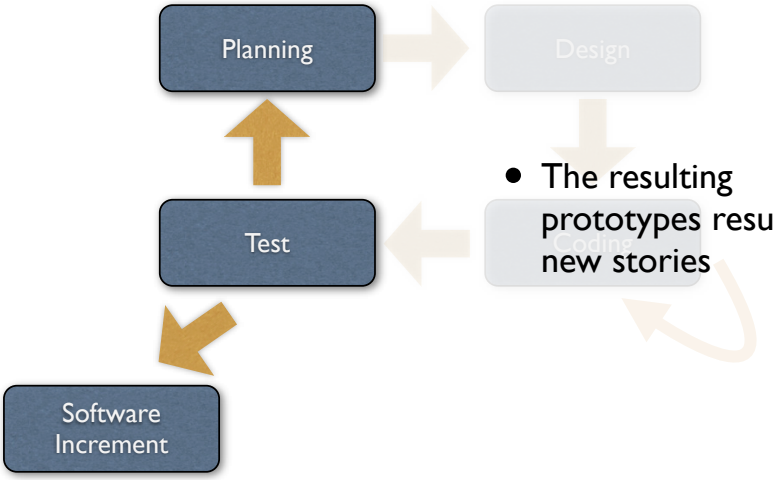
Testing



Unit tests

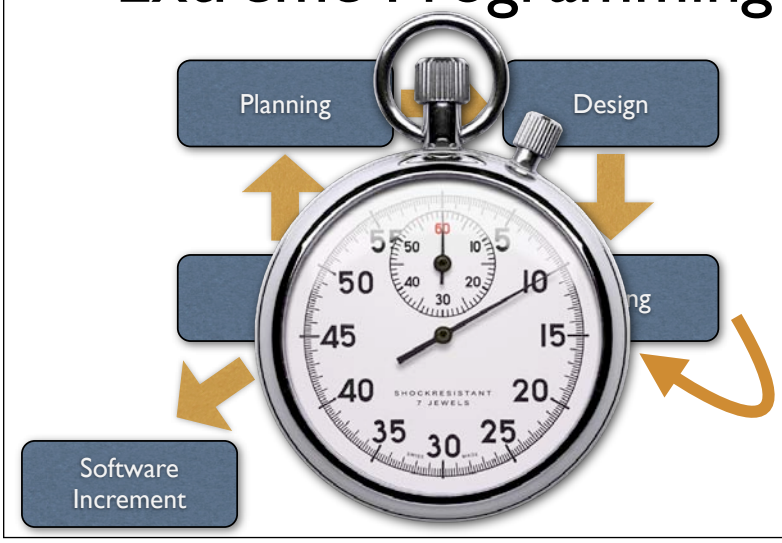
- detect errors
- find missing functionality
- measure progress

Extreme Programming



- The resulting prototypes result in new stories

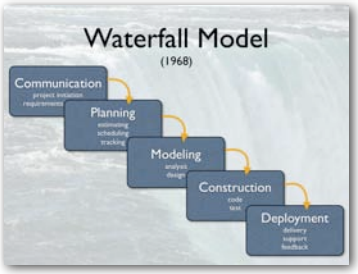
Extreme Programming



Extreme Programming is fast – with multiple deliverables per day!

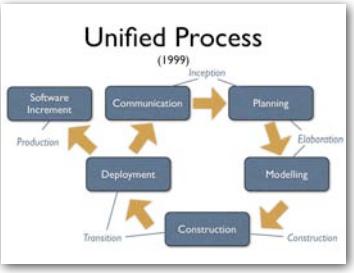
Code and Fix

Waterfall Model
(1968)



Summary

Unified Process
(1999)



Extreme Programming