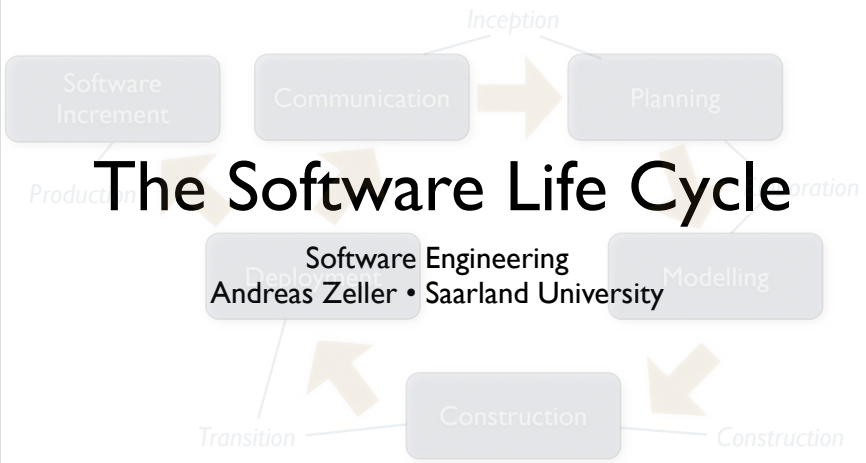
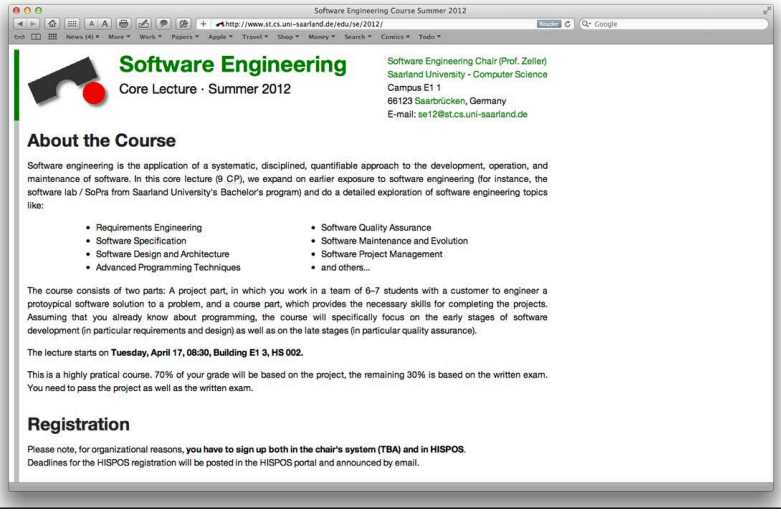


The Software Life Cycle

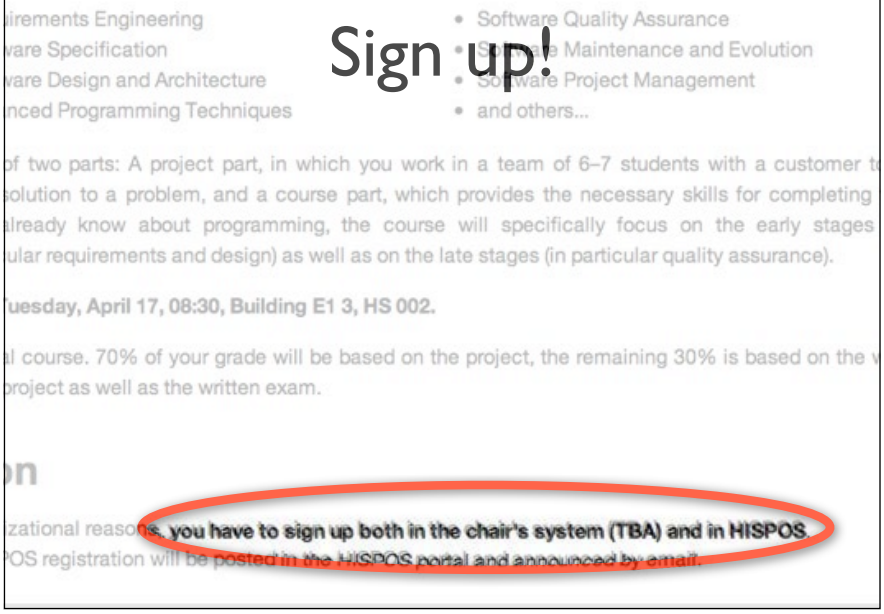
Software Engineering
Andreas Zeller • Saarland University

Sign up!



http://www.st.cs.uni-saarland.de/edu/se/2012/

Sign up!



Select Projects

Thursday, 18:00 – Monday, 12:00

Project preferences	NO!	no	.	yes	YES!
Advanced Automatic Repository Creation for Redmine:	○	○	○	◉	○
APex: Reports and visualization:	○	○	○	○	◉
BugEx Online:	○	○	○	◉	○
BugExlipse (BugEx Eclipse Plugin):	○	◉	○	○	○
Buy one, pay one: Type-checks to enforce resource-aware policies on cryptographic protocols:	○	○	◉	○	○
Cognitive Load: Data composition:	○	○	○	◉	○
Database for Lipid Imaging Mass Spectrometry Data:	○	○	◉	○	○
Database Systems Benchmark Running Framework:	○	◉	○	○	○
Dynamic MapReduce Cluster Management Tool:	○	○	◉	○	○
Graph Editor for Syntactic and Semantic Annotation of Natural Language:	○	○	○	◉	○
HBKsaar: Visual arts scheduling software:	○	◉	○	○	○
I, Librarian: Better management of meta-information:	○	○	◉	○	○
Increasing Software Quality: Integrating High Quality Assertion Support in Eclipse:	○	○	◉	○	○
Integration of Robotic Mapping in the NIFTi project with Google maps:	○	○	◉	○	○
Making statistical queries on databases safe: Type-checking for differential privacy:	○	○	◉	○	○
Markerless Motion Capture: Meta-data and more:	○	○	◉	○	○
Moskito: Integrating the concept of non-essential changes:	○	○	◉	○	○
Open3dMap: Crowd-sourced peer-to-peer 3d mapping:	○	○	○	◉	○

The Software Life Cycle

Software Engineering
Andreas Zeller • Saarland University

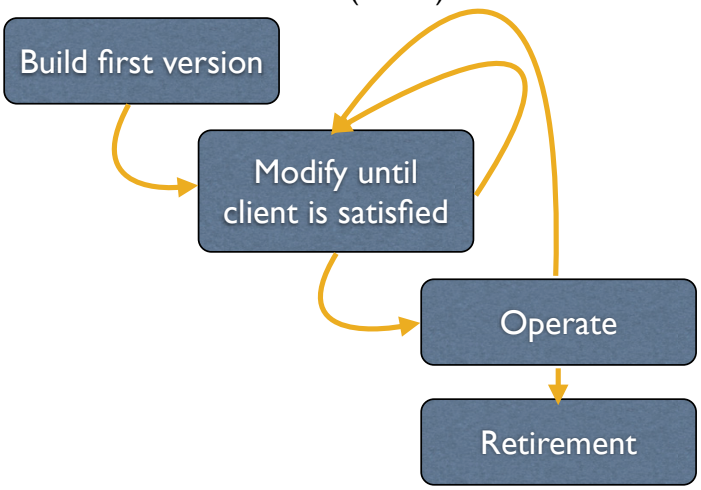
A Software Crisis

Denver International Airport (DIA)

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

Code and Fix

(1950–)



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
project initiation
requirements

Planning
estimating
scheduling
tracking

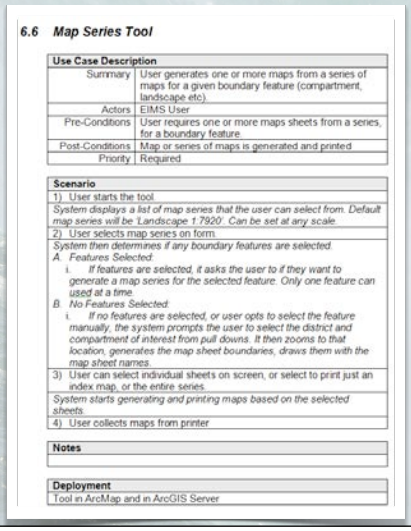
Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

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

Waterfall Model

(1968)

Communication
project initiation
requirements

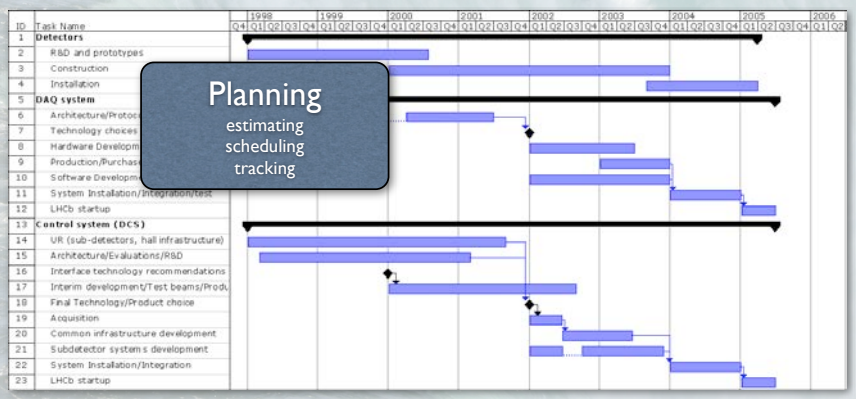
Planning
estimating
scheduling
tracking

Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

Planning



Waterfall Model

(1968)

Communication
project initiation
requirements

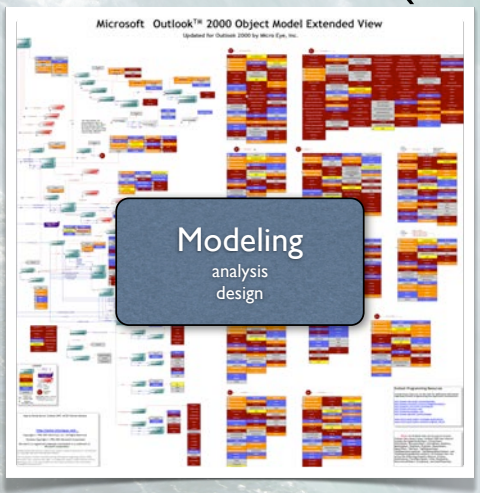
Planning
estimating
scheduling
tracking

Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

Waterfall Model (1968)

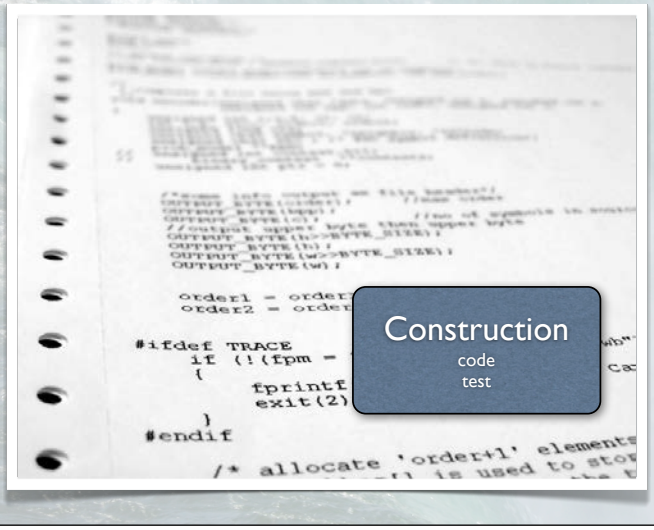


Waterfall Model

(1968)

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

Waterfall Model



Construction
code
test

Waterfall Model

(1968)

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

Deployment

Deployment
delivery
support
feedback

Waterfall Model

(1968)

Communication
project initiation
requirements

Planning
estimating
scheduling
tracking

Modeling
analysis
design

Construction
code
test

Deployment
delivery
support
feedback

Waterfall Model

(1968)

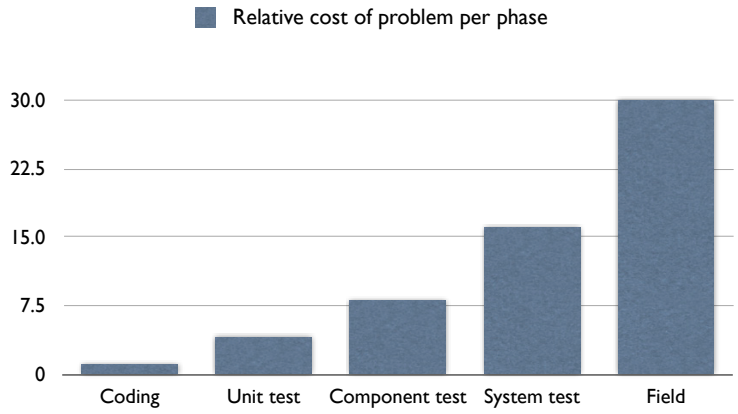
- Real projects rarely follow a sequential flow
- Hard to state all requirements explicitly
- No maintenance or evolution involved
- Customer must have patience
- Any blunder can be disastrous

Boehm's first law

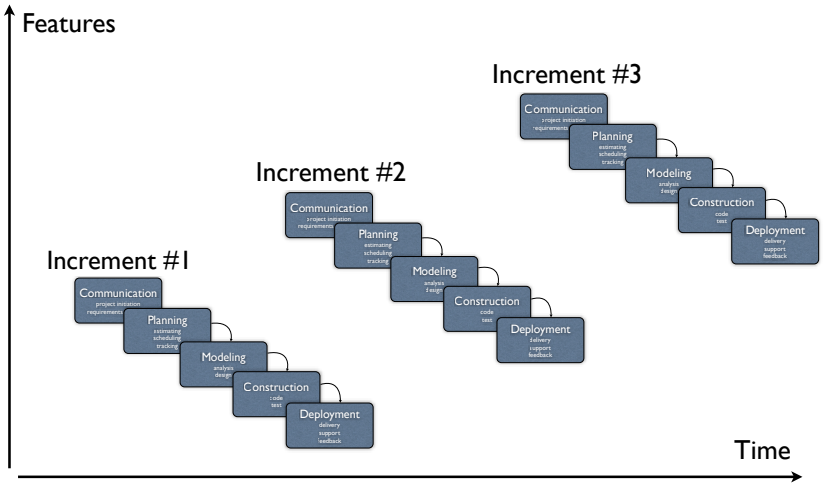
Errors are most frequent during *requirements* and *design* activities and are the more expensive the later they are removed.

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

Problem Cost



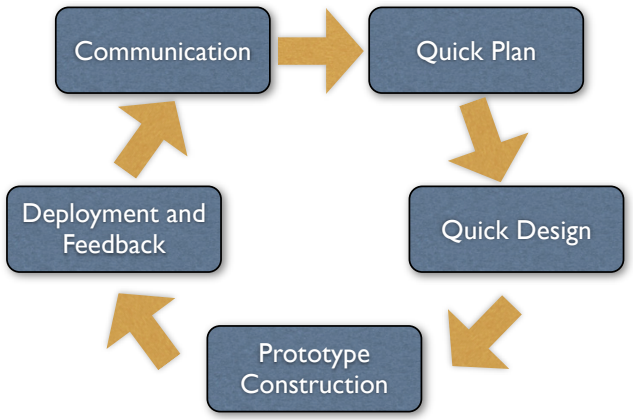
Incremental Model



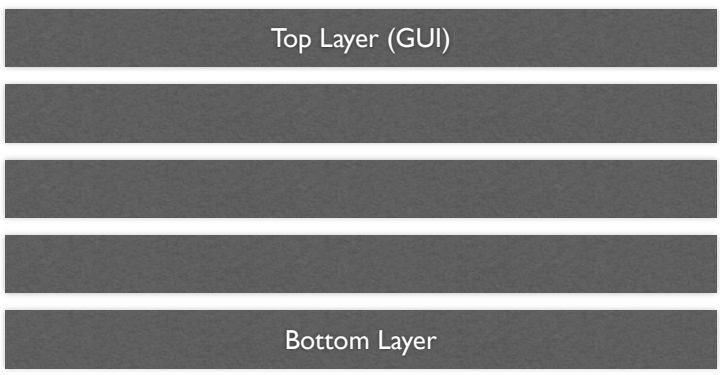
Incremental Model

- 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

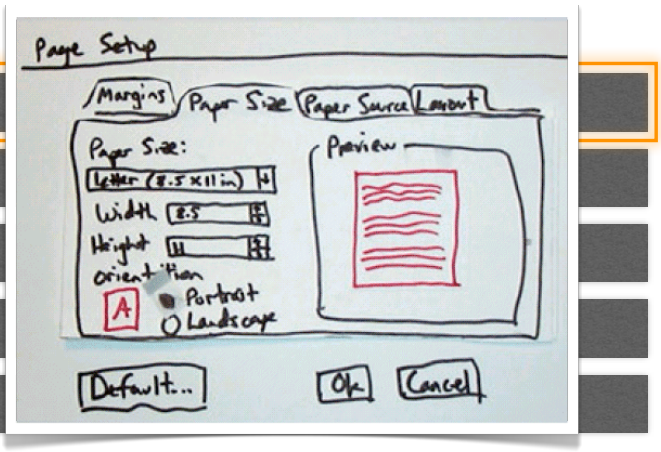
Prototyping



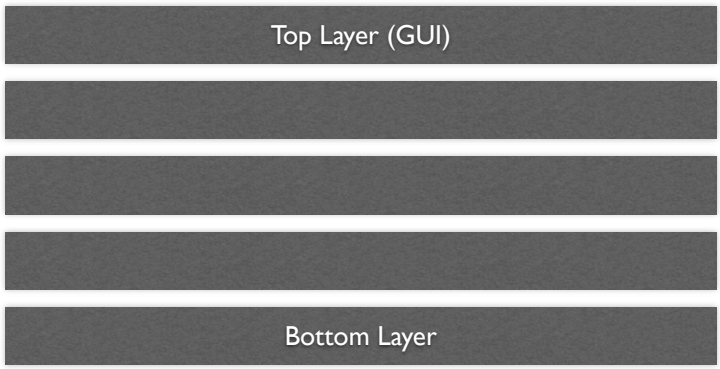
Prototypes



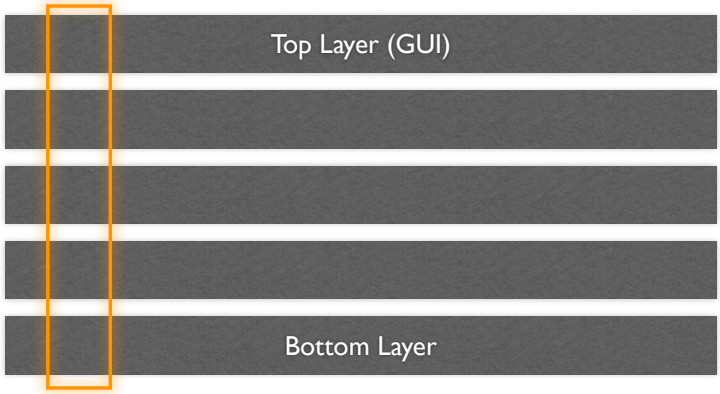
Horizontal Prototype



Prototypes



Vertical Prototype

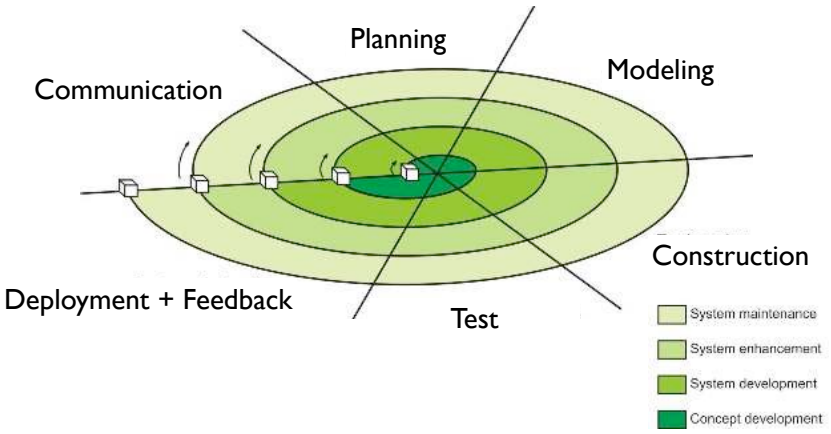


Prototypes

- 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)

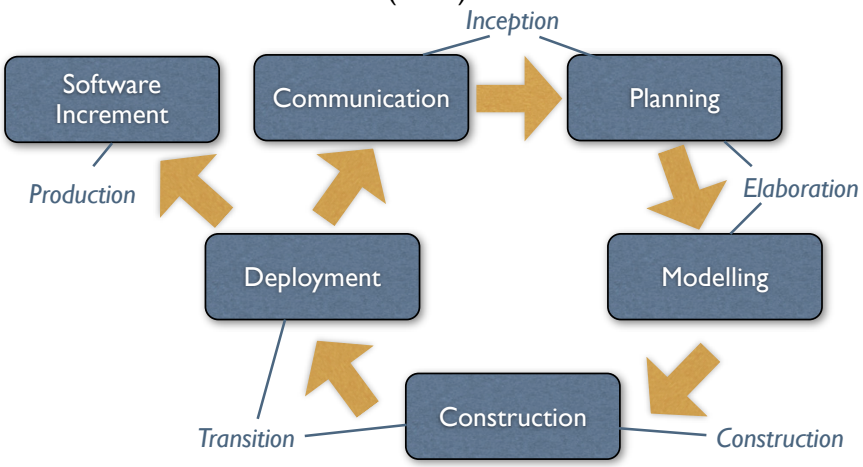


Spiral Model

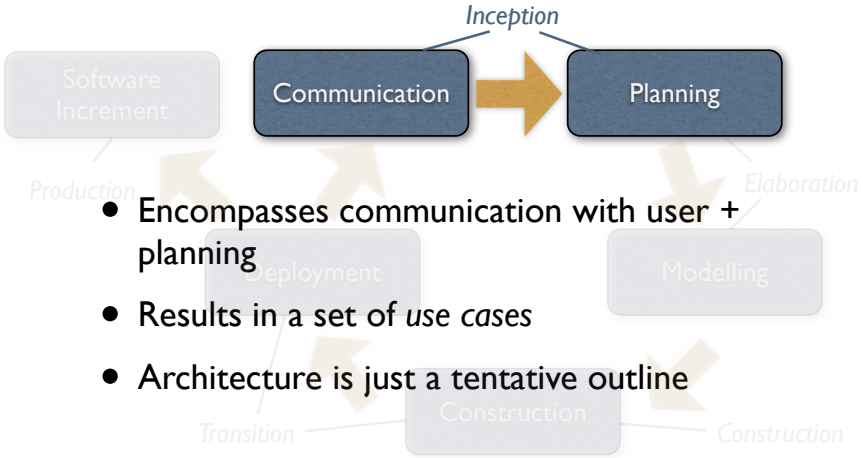
- System is developed in series of evolutionary releases
- Milestones for each iteration of the spiral
- Process does not end with delivery
- Reflects iterative nature of development

Unified Process

(1999)

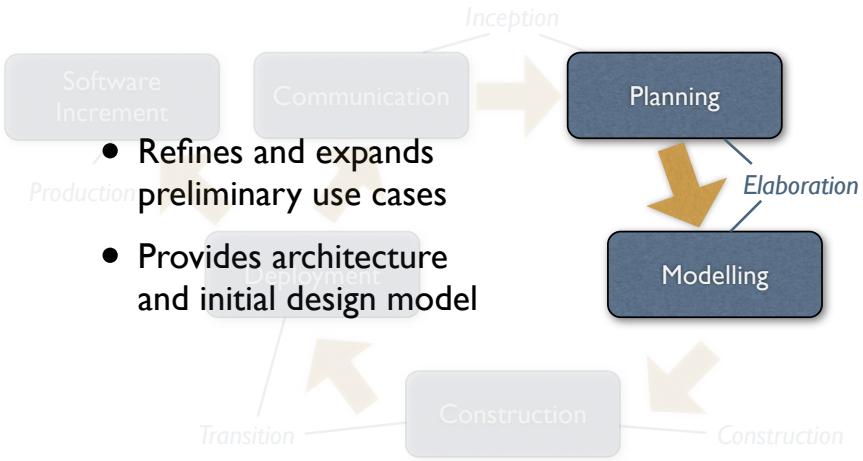


Inception



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

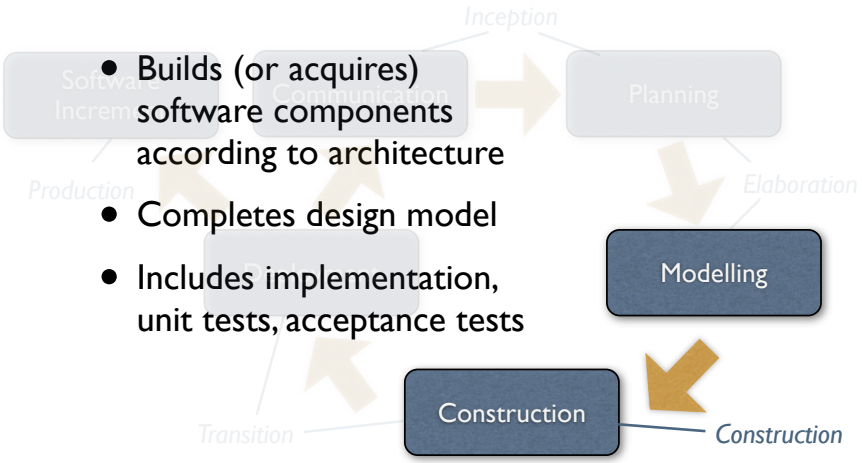
Elaboration



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

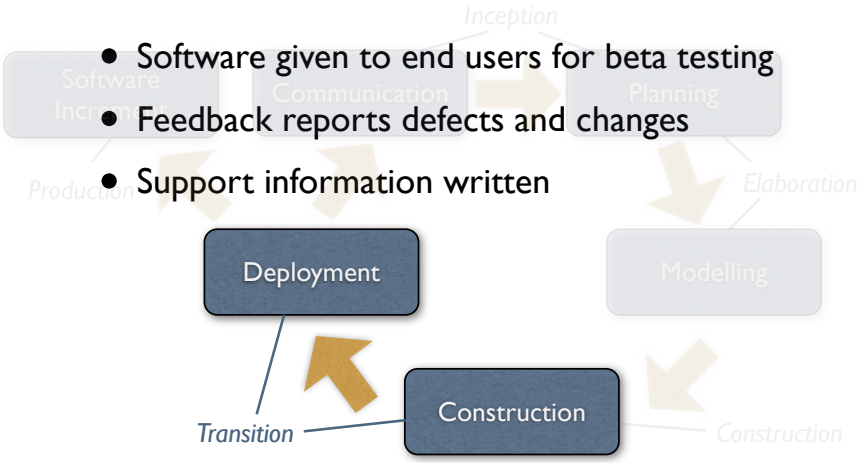
Construction

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



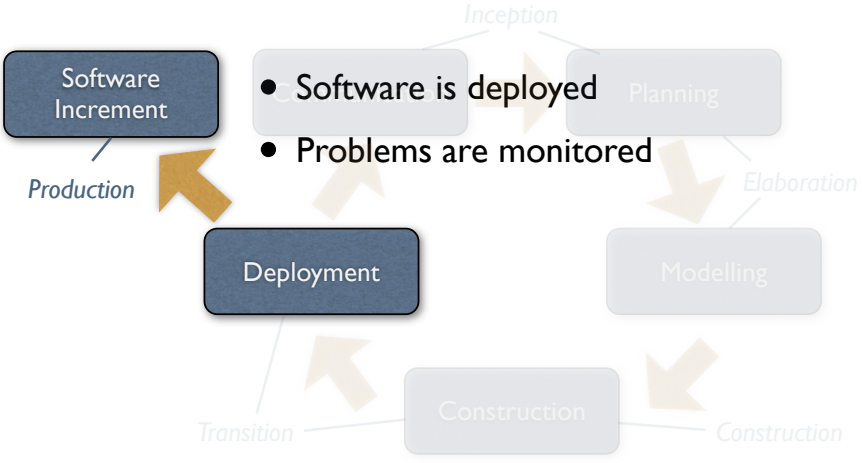
Transition

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

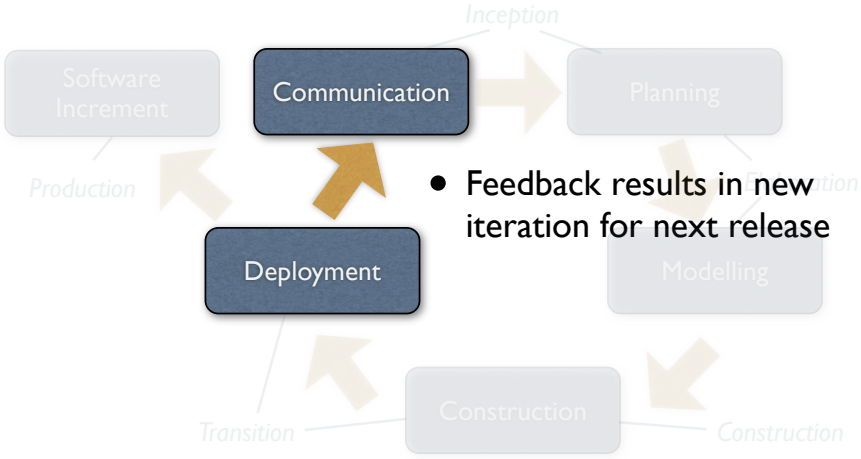


Production

- Software is deployed
- Problems are monitored

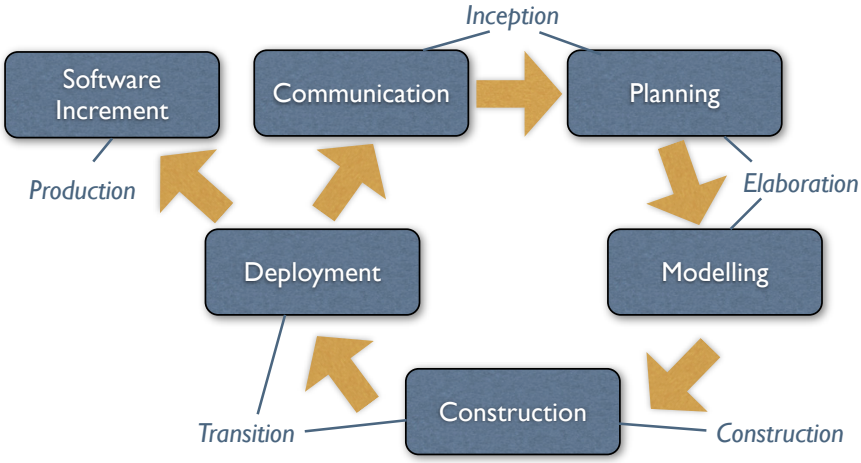


Re-Iteration



- 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)

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

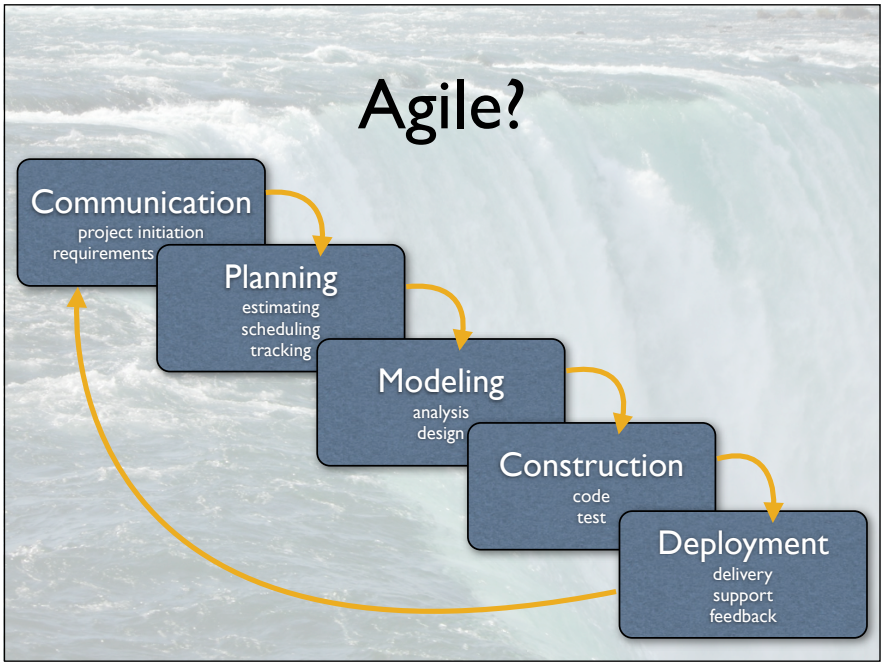
Agile Alliance

Manifesto for Agile Software Development (2001)

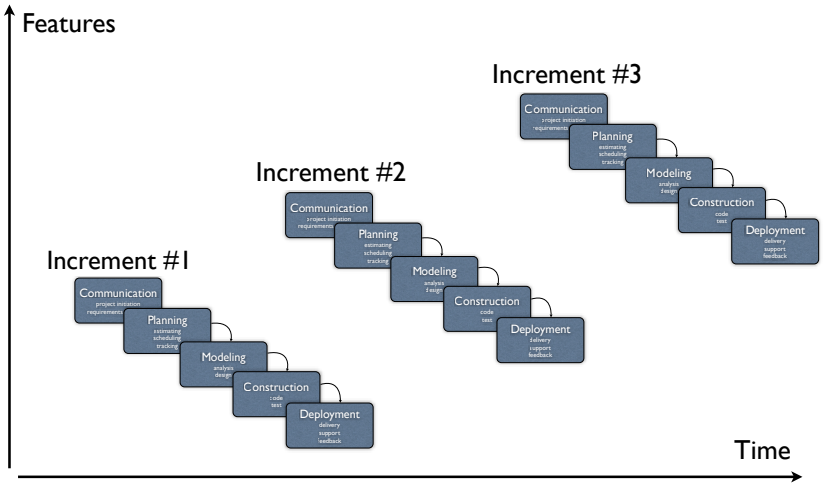
- Individuals and activities over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan..

What is Agile Development?

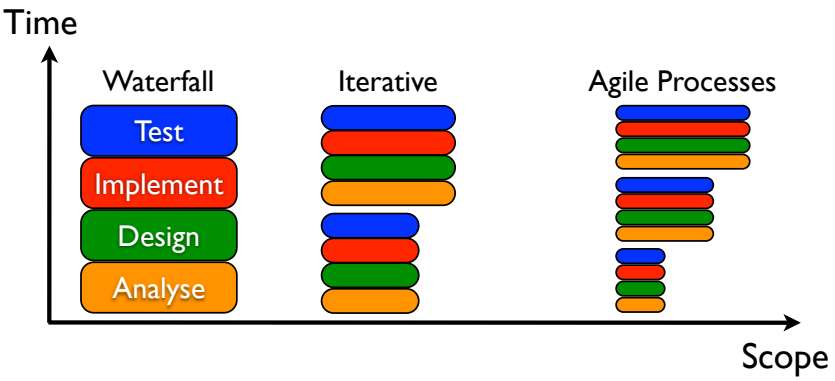
- Fast development? Hacking? Prototyping? Uncontrolled fun? Programmer heaven?
- Agility = ability to react to changing situations quickly, appropriately, and effectively.
  - notice changes early
  - initiate action promptly
  - create a feasible and effective alternative plan quickly
  - reorient work and resources quickly and effectively



Incremental Model



Agile Processes



Credits: Prof. Bodik

Agile vs. Plan-driven

Agile	Plan-driven
• Low criticality	• High criticality
• Senior developers	• Junior developers
• Requirements change very often	• Requirements don't change too often
• Small number of developers	• Large number of developers
• Culture that thrives on chaos	• Culture that demands order

What is an Agile Process?

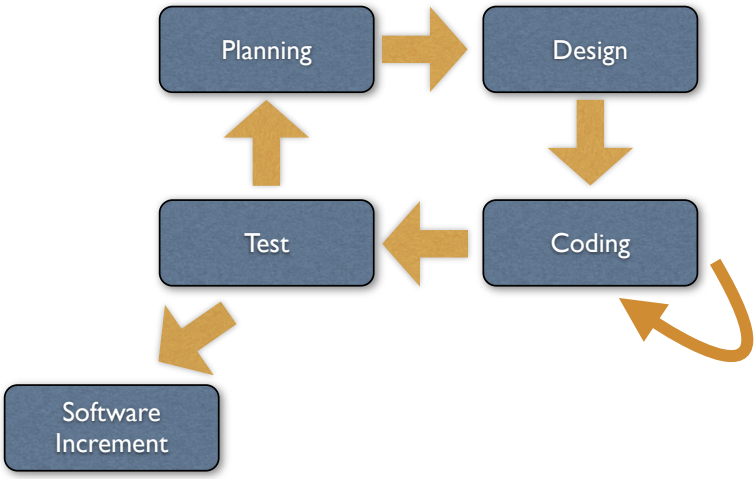
- Difficult to predict which requirements will persist or change in the future.
- For many types of software, design and development are interleaved.
- Analysis, design, construction, and testing are not as predictable.

So, how to tackle unpredictability?

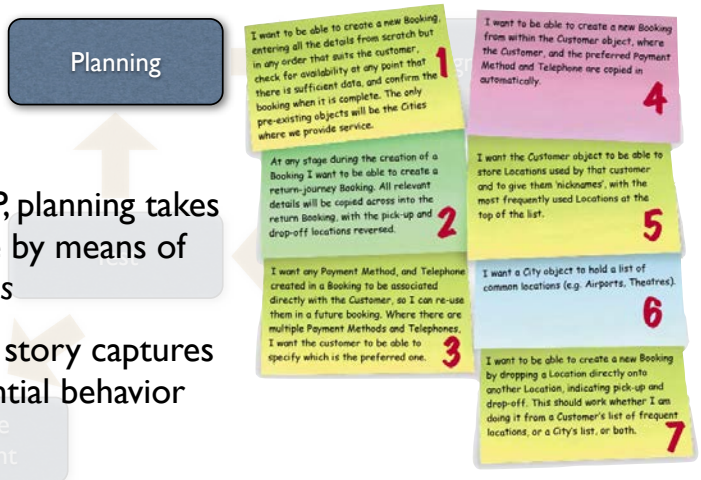
make the process adaptable...

Extreme Programming

(1999–)

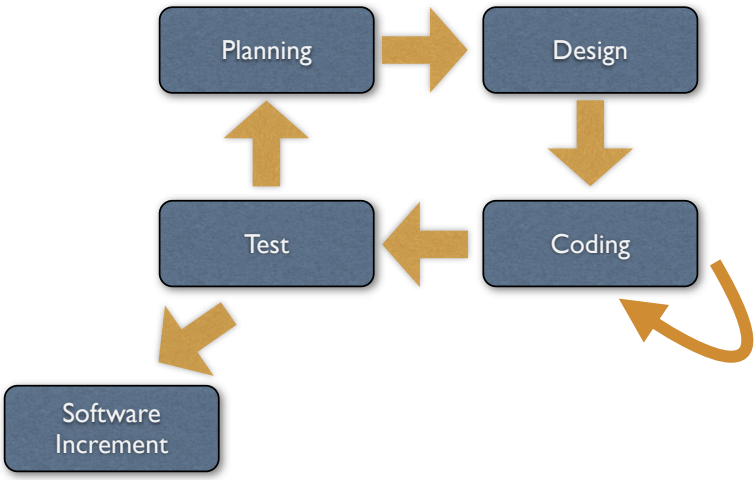


Planning

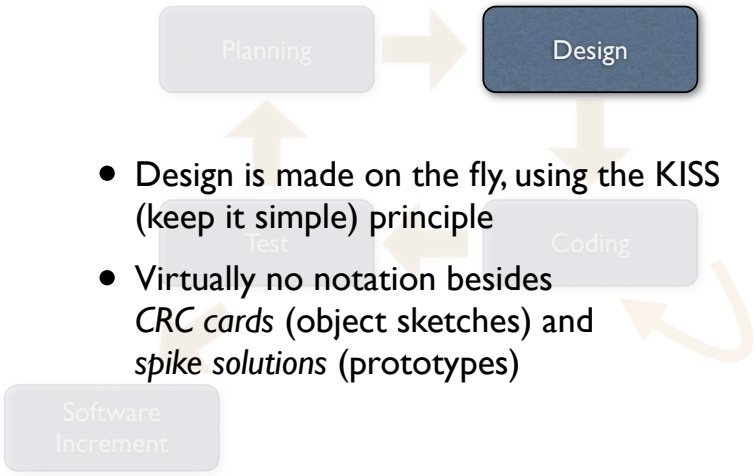


- In XP, planning takes place by means of *stories*
- Each story captures essential behavior

Extreme Programming

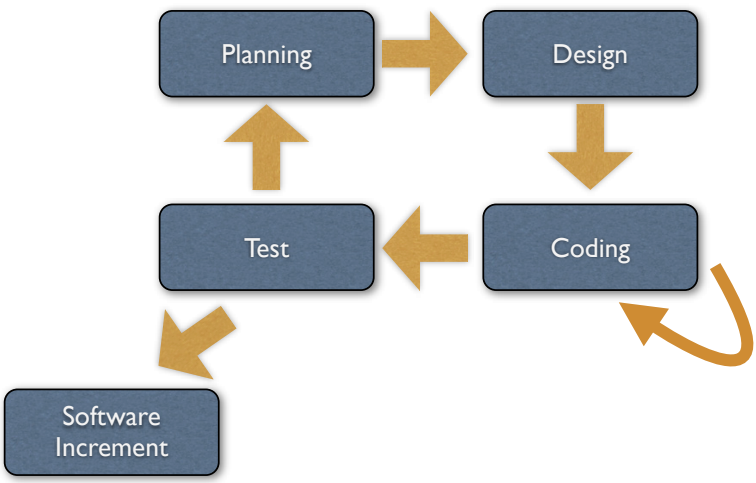


Extreme Programming



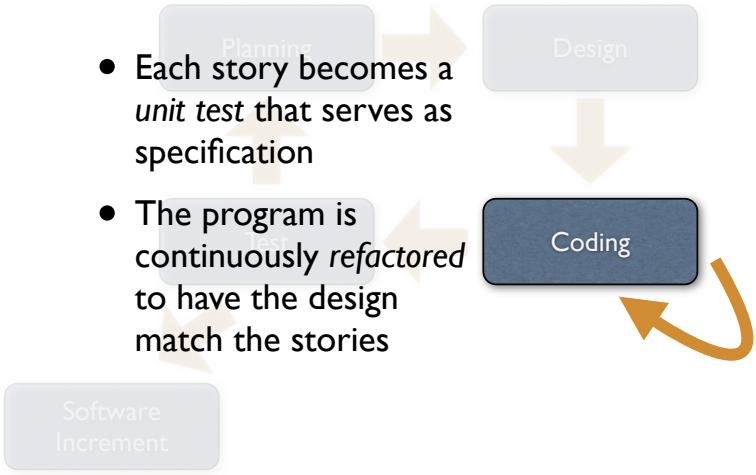
- Design is made on the fly, using the KISS (keep it simple) principle
- Virtually no notation besides *CRC cards* (object sketches) and *spike solutions* (prototypes)

Extreme Programming



Coding

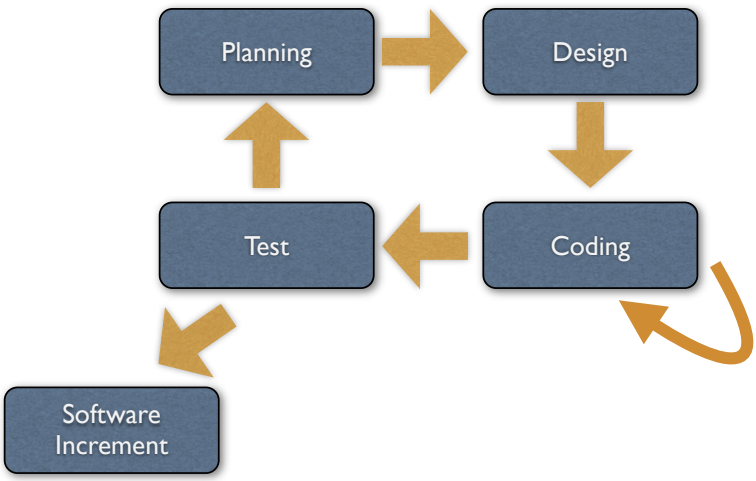
- Each story becomes a *unit test* that serves as specification
- The program is continuously *refactored* to have the design match the stories



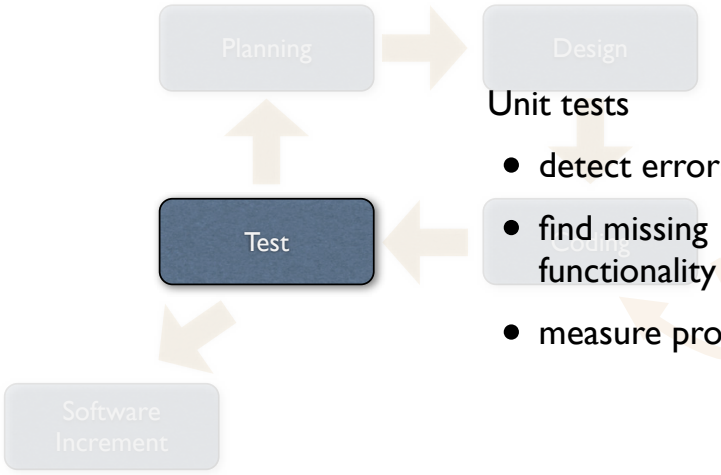
Coding

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

Extreme Programming



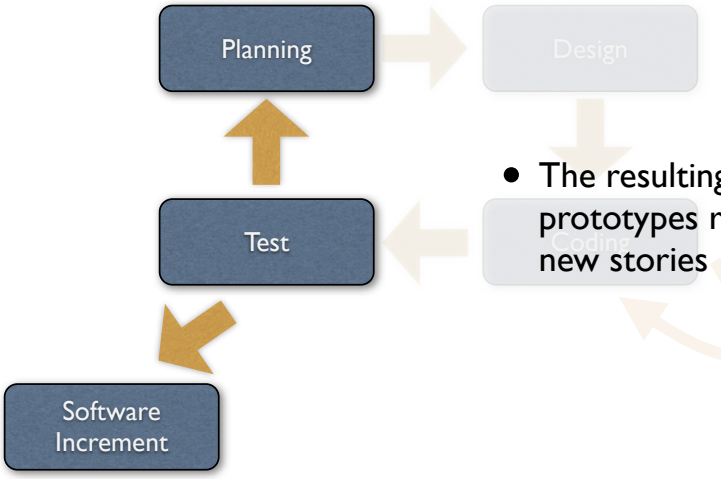
Testing



Unit tests

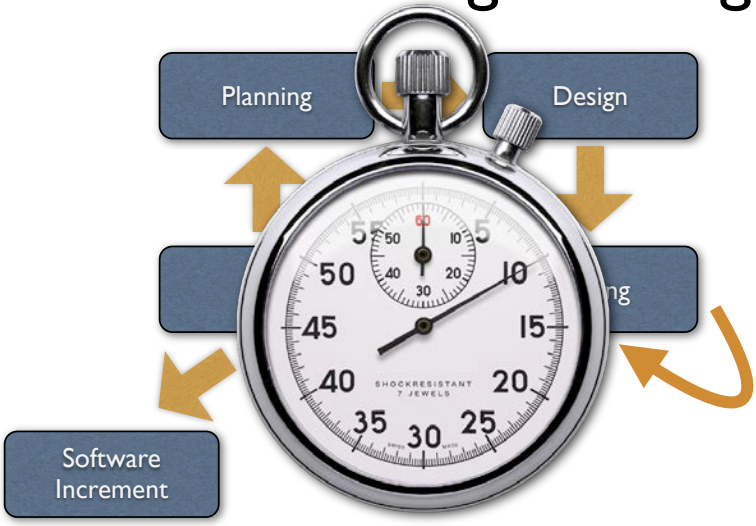
- detect errors
- find missing functionality
- measure progress

Extreme Programming



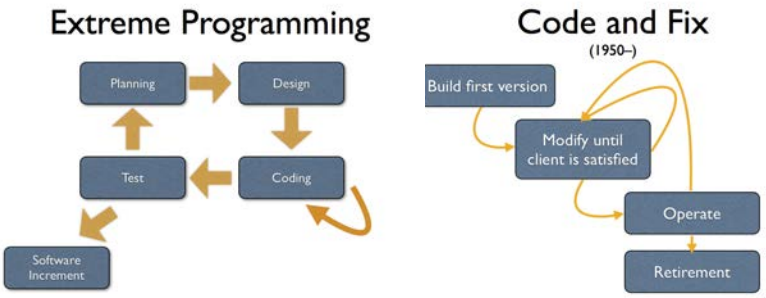
- The resulting prototypes result in new stories

Extreme Programming



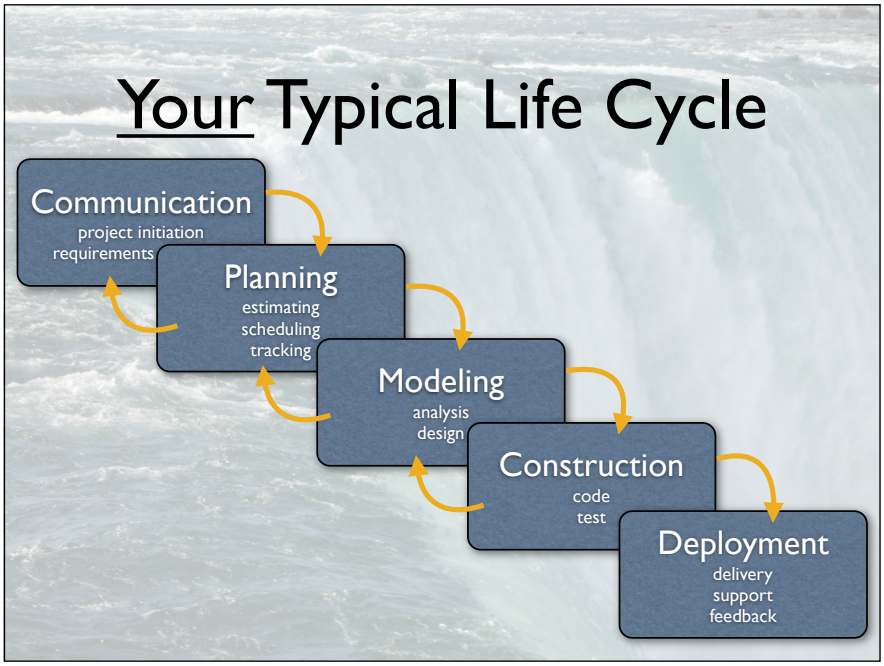
Extreme Programming is fast – with multiple deliverables per day!

Spot the Difference



So, aren't agile techniques just "code and fix" in disguise? Why not? (Hint: Think about explicit requirements, and explicit quality assurance)

Your Typical Life Cycle



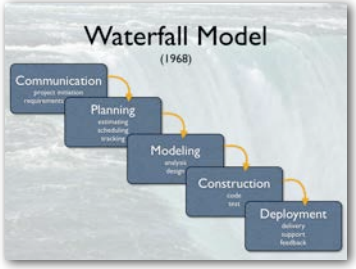
Your Typical Life Cycle

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

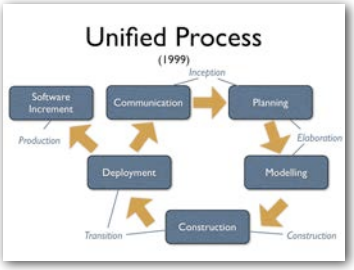
13 iterations total!

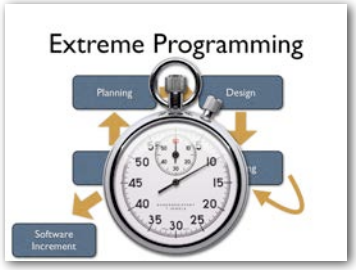
(it's ∞ iterations only if you are very, very successful)


Code and Fix
CODING HORROR
CODING HORROR
CODING HORROR
CODING HORROR


Waterfall Model
(1968)
Communication
Planning
Modeling
Construction
Deployment

Summary


Unified Process
(1999)
Inception
Software Increment
Communication
Planning
Production
Elaboration
Deployment
Modelling
Transition
Construction
Construction


Extreme Programming
Planning
Design
Software Increment