

Software Engineering (Part 4)

Copyright © 2024 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover these software engineering topics:

Stages of SW dev

How to order the stages

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- Process models

Objectives

Software Engineering lectures:

Part 1	Requirements analysis Design (general)
Part 2	Design (object-oriented) Implementation Debugging
Part 3	Testing Evaluation
Part 4	Maintenance Process models

So the system is finished. Or is it?

Agenda

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- **Maintenance**
- Process models

Maintenance

- ***Maintenance***
 - Alias ***continuance***
 - How can I ensure that the system continues to fulfill the users' needs through time?

Maintenance

Rod Stephens.
Beginning Software Engineering.
Wiley, 2015

- **Perfective** maintenance
 - Add new features, improve (performance of) existing features
 - Analyze **execution profiles**
- **Adaptive** maintenance
 - Modify the system to meet changes in its environment
- **Corrective** maintenance
 - Fix bugs
- **Preventive** maintenance
 - **Refactor code** to make it more maintainable

Maintenance: Profiling

- Profiling concord.py
 - See profiling1/
 - concord.py
 - From *Python Language (Part 5)* lecture
 - writeprofile.py
 - buildandrun
 - buildandrun.bat

Maintenance: Profiling

- Profiling concord.py
 - See profiling2/
 - concord.py
 - writeprofile.py
 - buildandrun
 - buildandrun.bat

Maintenance: Profiling

- Suppose you didn't spot that opportunity for improvement
- What would you do?

Maintenance: Profiling

- Tool support for profiling
 - **Python:** *cProfile* module
 - Example...

Maintenance: Profiling

```
$ cd profiling1
$ ./buildandrun
```

```
# Create concord.profile
```

```
python -m cProfile -o concord.profile concord.py < Bible.txt
```

```
welcome: 1
```

```
to: 13569
```

```
you: 2621
```

```
have: 3905
```

```
arrived: 3
```

```
at: 1571
```

```
a: 8178
```

```
plain: 76
```

```
text: 1
```

```
...
```

```
...
```

```
alleluia: 4
```

```
omnipotent: 1
```

```
chalcedony: 1
```

```
sardonyx: 1
```

```
chrysolite: 1
```

```
chrysoprasus: 1
```

```
transparent: 1
```

```
proceeding: 1
```

```
# Generate the report
```

```
python writeprofile.py concord.profile > report.txt
```

```
# To view the report examine the contents of report.txt
```

```
$
```

Maintenance: Profiling

```
$ cat report.txt
Mon Apr 24 20:03:51 2023      concord.profile

        698882 function calls (698878 primitive calls) in 0.798 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
114157  0.277      0.000    0.659    0.000    concord.py:13(process_line)
114157  0.257      0.000    0.257    0.000    {method 'findall' of 're.Pattern' objects}
      1  0.079      0.079    0.797    0.797    concord.py:25(main)
    12614  0.058      0.000    0.058    0.000    {built-in method builtins.print}
114157  0.050      0.000    0.078    0.000    /usr/lib/python3.10/re.py:288(_compile)
114157  0.029      0.000    0.106    0.000    /usr/lib/python3.10/re.py:249(compile)
114171  0.027      0.000    0.027    0.000    {built-in method builtins.isinstance}
114157  0.019      0.000    0.019    0.000    {method 'lower' of 'str' objects}
      592  0.001      0.000    0.002    0.000    /usr/lib/python3.10/codecs.py:319(decode)
      592  0.001      0.000    0.001    0.000    {built-in method _codecs.utf_8_dec
...

```

Maintenance: Profiling

```
$ cd profiling2
$ ./buildandrun
```

```
# Create concord.profile
```

```
python -m cProfile -o concord.profile concord.py < Bible.txt
```

```
welcome: 1
```

```
to: 13569
```

```
you: 2621
```

```
have: 3905
```

```
arrived: 3
```

```
at: 1571
```

```
a: 8178
```

```
plain: 76
```

```
text: 1
```

```
...
```

```
...
```

```
alleluia: 4
```

```
omnipotent: 1
```

```
chalcedony: 1
```

```
sardonyx: 1
```

```
chrysolite: 1
```

```
chrysoprasus: 1
```

```
transparent: 1
```

```
proceeding: 1
```

```
# Generate the report
```

```
python writeprofile.py concord.profile > report.txt
```

```
# To view the report examine the contents of report.txt
```

```
$
```

Maintenance: Profiling

```
$ cat report.txt
```

```
Mon Apr 24 20:07:54 2023      concord.profile
```

```
356414 function calls (356410 primitive calls) in 0.577 seconds
```

```
Ordered by: internal time
```

ncalls	tottime	percall	cumtime	percall	filename:lineno (function)
114157	0.236	0.000	0.451	0.000	concord.py:13 (process_line)
114157	0.196	0.000	0.196	0.000	{method 'findall' of 're.Pattern' objects}
1	0.068	0.068	0.577	0.577	concord.py:24 (main)
12614	0.057	0.000	0.057	0.000	{built-in method builtins.print}
114157	0.018	0.000	0.018	0.000	{method 'lower' of 'str' objects}
592	0.001	0.000	0.001	0.000	{built-in method _codecs.utf_8_decode}
592	0.001	0.000	0.002	0.000	/usr/lib/python3.10/codecs.py:319 (decode)
1	0.000	0.000	0.577	0.577	concord.py:1 (<module>)

```
...
```

Aside: Performance vs. Coupling

- Which version of concord.py is better?
- Version 2 has:
 - Better performance
 - By a large margin
- Version 1 has:
 - Weaker function-level coupling
 - By a small margin

Maintenance: Profiling

Language	Profiling Tool
Python	<i>cProfile</i>
Java	<i>hprof</i> & <i>JPerfAnal</i> *
C (x86-64 or ARM)	<i>gprof</i> *
C (x86-64)	<i>OProfile</i> *
JavaScript (browser)	<i>Chrome Developer Tools</i> <i>Firefox Performance Tool</i>
JavaScript (Node.js)	<i>Node.js profiler</i>

* See me if you want an example

Maintenance: Refactoring



Martin Fowler



2000

Maintenance: Refactoring

- ***Bad smells*** in code

Martin Fowler.

Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Duplicated code

Long method

Long parameter list

Divergent change

Shotgun surgery

Feature envy

Data clumps

Primitive obsession

Switch statements

Parallel inheritance hierarchies

Lazy class

Speculative generality

Temporary field

Message chains

Middle man

Inappropriate intimacy

Alternative classes with diff
interfaces

Incomplete library class

Data class

Refused bequest

Comments

Maintenance: Refactoring

1. Composing methods (9)

- Extract method
- Inline method
- Inline temp
- Replace temp with query
- Introduce explaining variable
- Split temporary variable
- Remove assignments to parameters
- Replace method with method object
- Substitute algorithm

2. Moving features between objects (8)

- Move method
- Move field
- Extract class
- Inline class
- Hide delegate
- Remove middle man
- Introduce foreign method
- Introduce local extension

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

3. Organizing data (16)

- Self encapsulate field
- Replace data value with object
- Change value to reference
- Change reference to value
- Replace array with object
- Duplicate observed data
- Change unidirectional association to bidirectional
- Change bidirectional association to unidirectional
- Replace magic number with symbolic constant
- Encapsulate field
- Encapsulate collection
- Replace record with data class
- Replace record with class data
- Replace type code with subclasses**
- Replace type code with state/strategy
- Replace subclass with fields

4. Simplifying conditional expressions (8)

- Decompose conditional
- Consolidate conditional expression
- Consolidate duplicate conditional fragments
- Remove control flag
- Replace nested conditional with guard clauses
- Replace conditional with polymorphism
- Introduce null object
- Introduce assertion

Martin Fowler.

Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

5. Making method calls simpler (15)

- Rename method
- Add parameter
- Remove parameter
- Separate query from modifier
- Parameterize method
- Replace parameter with explicit methods
- Preserve whole object
- Replace parameter with method
- Introduce parameter object
- Remove setting method
- Hide method
- Replace constructor with factory method
- Encapsulate downcast
- Replace error code with exception
- Replace Exception with test

6. Dealing with generalization (12)

- Pull up field
- Pull up method
- Pull up constructor body
- Push down method
- Push down field
- Extract subclass
- Extract superclass
- Extract Interface
- Collapse hierarchy
- Form template method
- Replace inheritance with delegation
- Replace delegation with inheritance

Martin Fowler.

Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

7. Big refactorings (4)

Tease apart inheritance

Convert procedural design to objects

Separate domain from presentation

Extract hierarchy

Total: 72

Martin Fowler.

Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

- ***Replace Type Code with Subclasses***
 - You have an immutable type code that affects the behavior of a class
 - Replace the type code with subclasses

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

- *Replace Type Code with Subclasses*

```
public class Shape
{
    private static final int RECTANGLE = 0;
    private static final int SQUARE = 1;
    private int shapeType;
    ...
    public void move()
    {
        switch (shapeType)
        {   case RECTANGLE:
            ...
            break;
            case SQUARE:
            ...
            break;
        }
    }
}
```

Before

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

- *Replace Type Code with Subclasses*

After

```
public abstract class Shape
{
    public abstract void move();
}
public class Rectangle extends Shape
{
    public void move { ... }
}
public class Square extends Rectangle
{
    public void move { ... }
}
```

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

Smell	Common Refactorings
Alternative classes with diff interfaces	Rename method, move method
Comments	Extract method, introduce assertion
Data class	Move method, encapsulate field, encapsulate collection
Data clumps	Extract class, introduce parameter object, preserve whole object
Divergent change	Extract class
Duplicated code	Extract method, extract class, pull-up method, form template method
Feature envy	Move method, move field, extract method
Inappropriate intimacy	Move method, move field, change bidirectional association to unidirectional, replace inheritance with delegation, hide delegate

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

Smell	Common Refactorings
Primitive obsession	Replace data value with object, extract class, introduce parameter object, replace array with object, replace type code with class, replace type code with subclasses, replace type code with state/strategy
Refused bequest	Replace inheritance with delegation
Shotgun surgery	Move method, move field, inline class
Speculative generality	Collapse hierarchy, inline class, remove parameter, rename method
Switch statements	Replace conditional with polymorphism, replace type code with subclasses , replace type code with state/strategy, replace parameter with explicit methods, introduce null object
Temporary field	Extract class, introduce null object

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

Maintenance: Refactoring

Smell	Common Refactorings
Incomplete library class	Introduce foreign method, introduce local extension
Large class	Extract class, extract subclass, extract interface, replace data value with object
Lazy class	Inline class, collapse hierarchy
Long method	Extract method, replace temp with query, replace method with method object, decompose conditional
Long parameter list	Replace parameter with method, introduce parameter object, preserve whole object
Message chains	Hide delegate
Middle man	Remove middle man, inline method, replace delegation with inheritance
Parallel inheritance hierarchies	Move method, move field

Martin Fowler.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley. New York. 2000.

How should you order those stages?

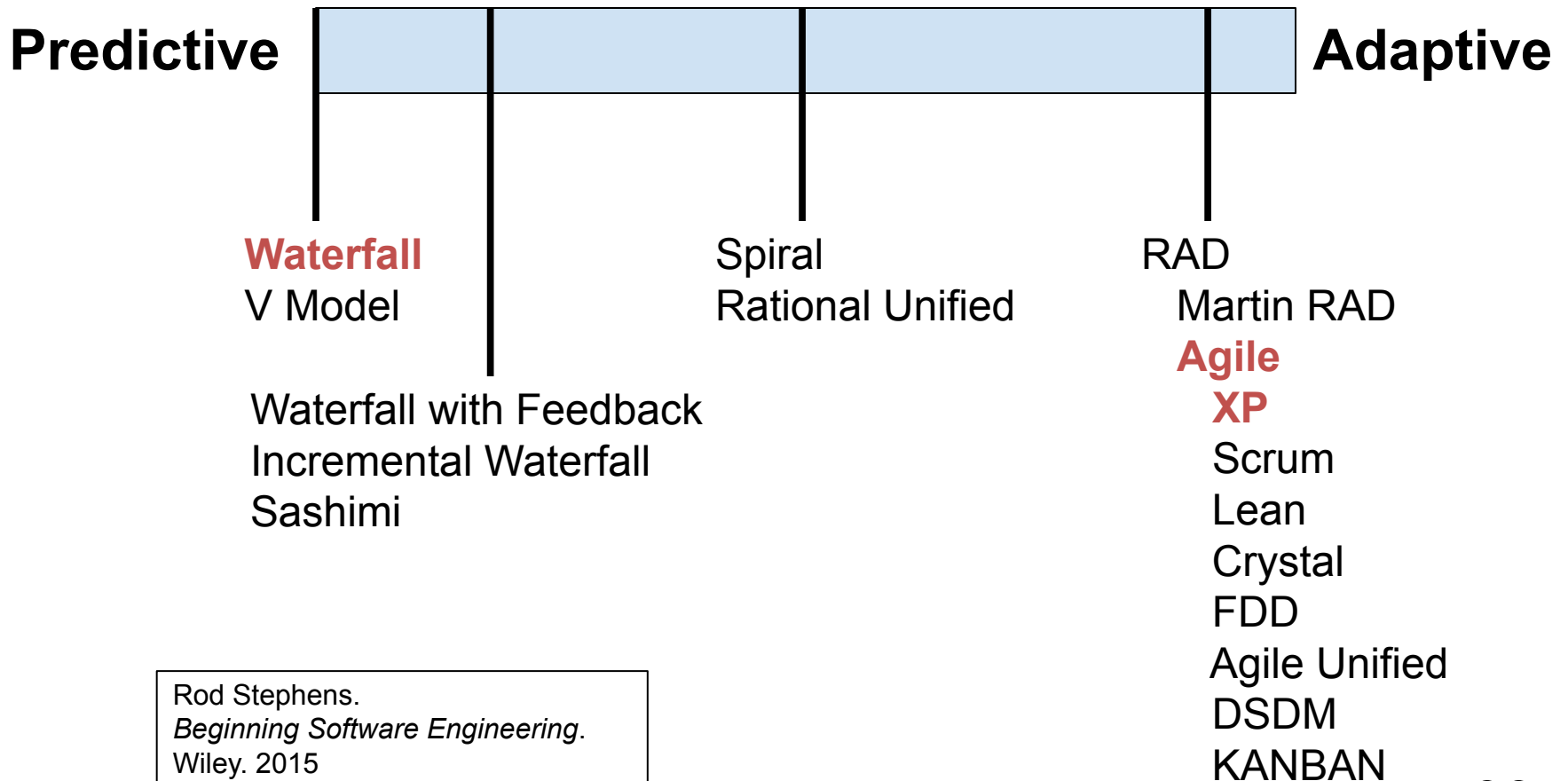
Agenda

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance
- **Process models**

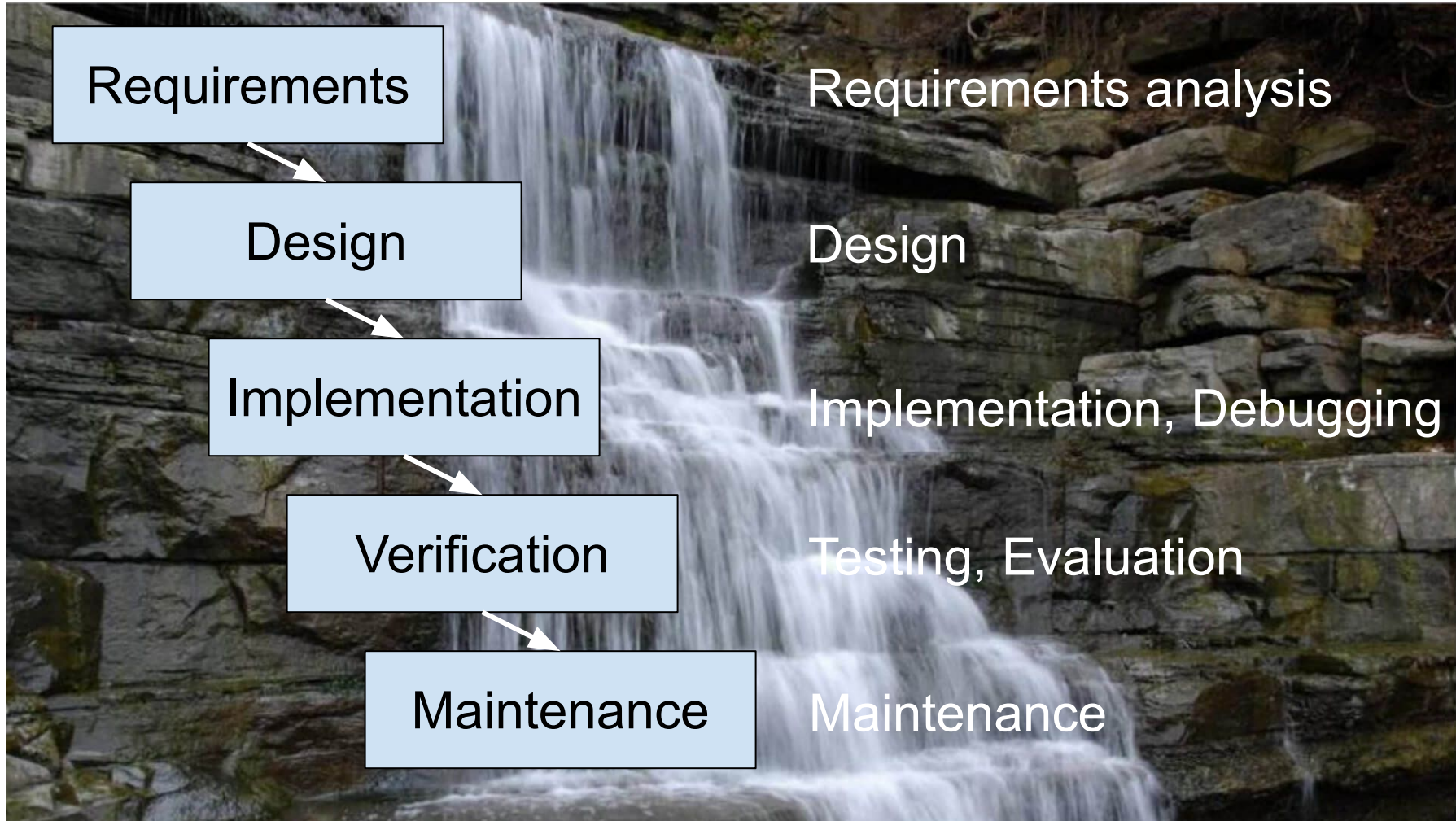
Process Models

- *Process models*
 - How should you order those stages?
 - (And much more)

Process Models



Process Models: Waterfall



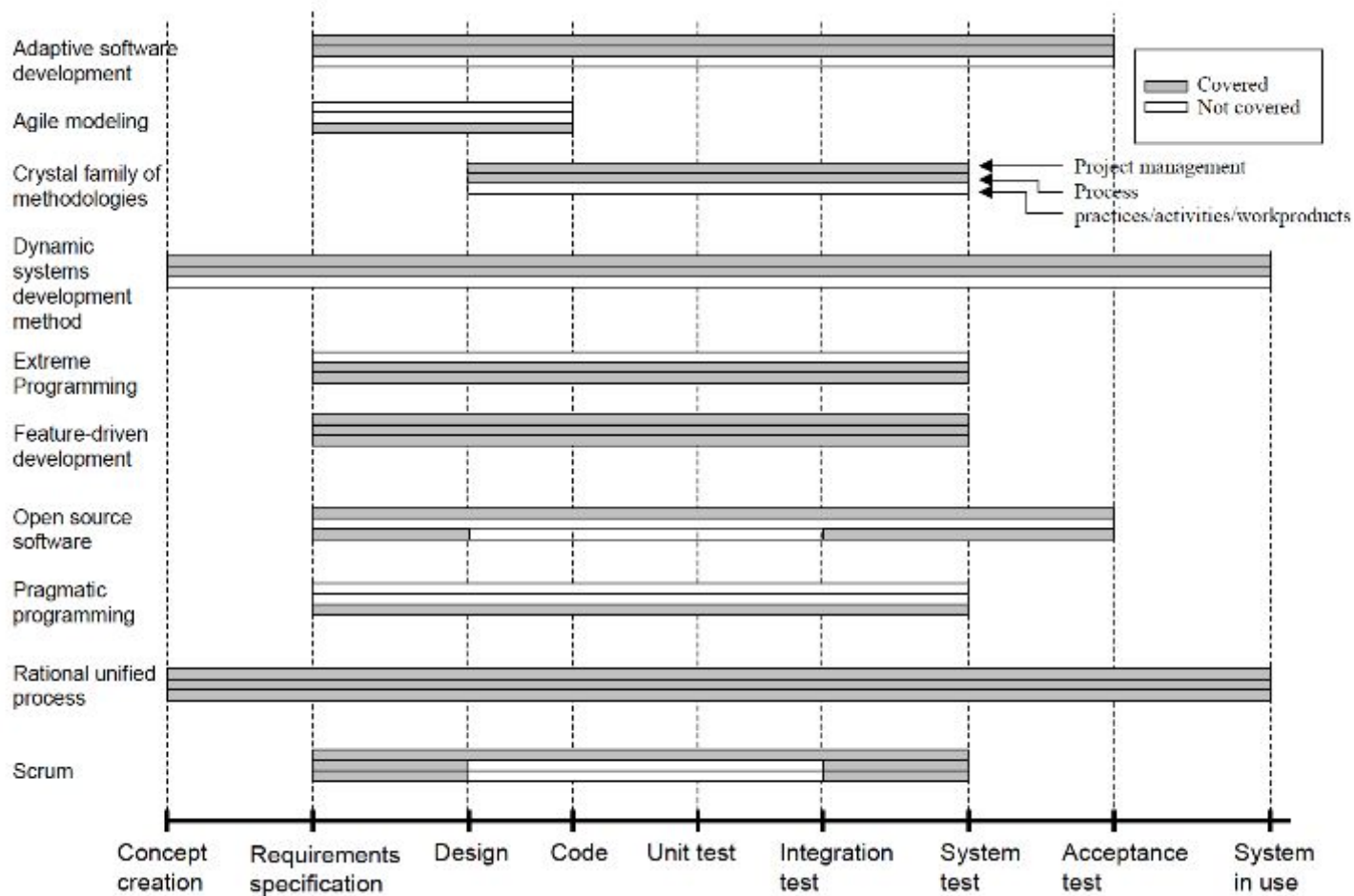
Process Models: Waterfall

- Completely predictive (non-adaptive)
 - From manufacturing industry
- Used by many early software dev projects
 - No other process models were known!
- Required by many funding agencies
 - Agency defines requirements
 - SW company does the rest, while agency monitors progress

Process Models: Waterfall

- Commentary
 - Perfect if all predictions are correct
 - It's **hardly ever** the case that all predictions are correct!

Process Models: Agile

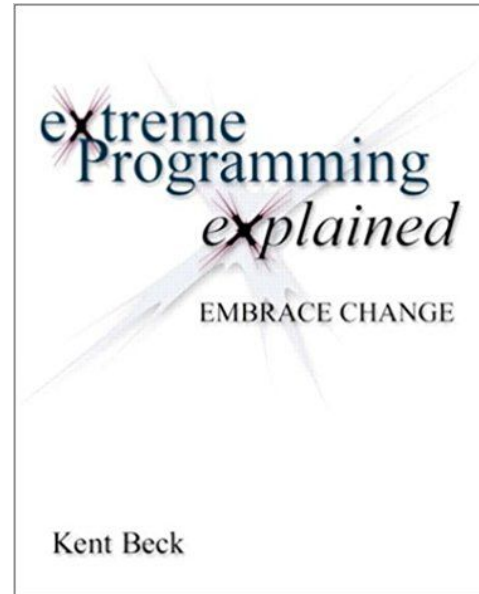


Abrahamson P, Salo O, Ronkainen J, Warsta J (2002). *Agile Software Development Methods: Review and Analysis*. (Technical report). VTT. 478.

Process Models: Agile

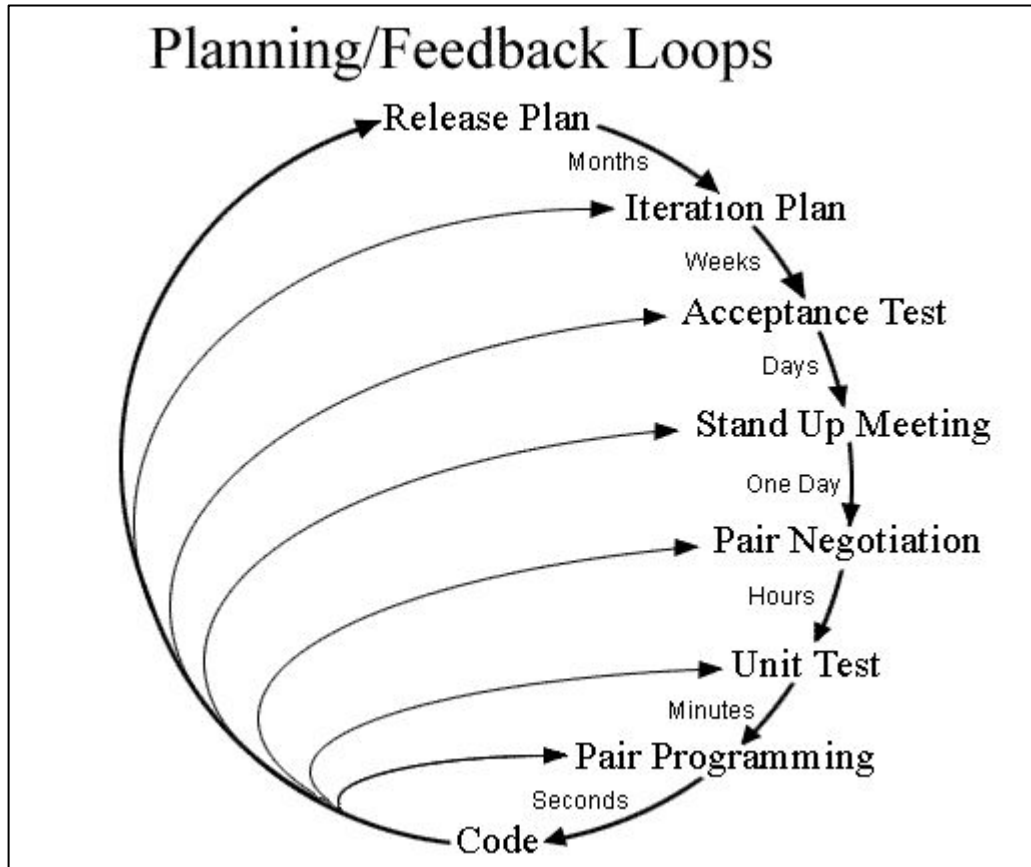


Kent Beck



2000

Process Models: Agile



Requirements analysis
Evaluation
Design
Testing
Impl., Debugging

Maintenance

Diagram from Wikipedia *Extreme Programming* page

Process Models: Agile

- As adaptive (non-predictive) as possible
 - “Extremely” adaptive
 - “Embrace change”
- Essentially, code is the only artifact produced

Process Models: Agile

- The planning game
- **Small releases**
- Metaphor
- Simple design
- **Testing**
- **Refactoring**
- **Pair programming**
- Collective ownership
- Continuous integration
- 40-hour work week
- **On-site customer**
- Coding standards

Kent Beck.
Extreme Programming Explained: Embrace Change.
Addison-Wesley. New York. 2000.

Process Models: Agile

- Commentary
 - Appealing!
 - Too extreme?
 - An excuse for programmers to avoid some tasks that they find less fun?

Process Models

Predictive vs. Adaptive models:

Use Predictive When:	Use Adaptive When:
Developers are plan-oriented, adequately skilled, and have access to external knowledge	Developers are agile, highly skilled, collocated, and collaborative
Customers are not collocated	Customers are collocated
Requirements are knowable early and largely stable	Requirements are largely emergent and change rapidly
Team and product are large	Team and product are small
Primary objective is high assurance	Primary objective is rapid value

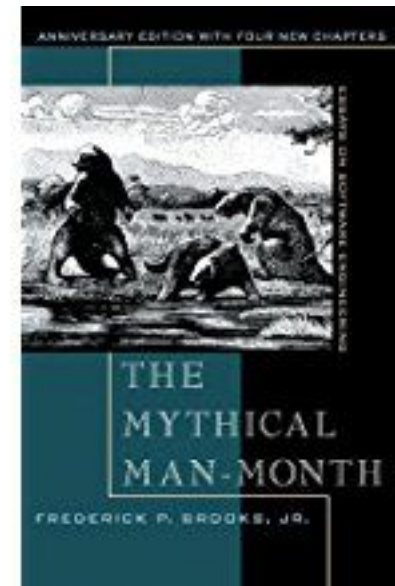
Boehm, B.

“Get Ready for the Agile Methods, With Care”
Computer 35 (1): 64-69.

Process Models: Commentary



Frederick
Brooks



1975
1995

Process Models: Commentary

“All software involves **essential** tasks, the fashioning of the complex conceptual structures that compose the abstract software entity, and **accidental** tasks, the representation of those abstract entities in programming languages and the mapping of these onto machine languages within space and speed constraints. Most of the big gains in software productivity have come from removing artificial barriers that have made the **accidental** tasks inordinately hard.”

Frederick Brooks.
The Mythical Man Month: Essays on Software Engineering
Addison-Wesley. New York. 1995.

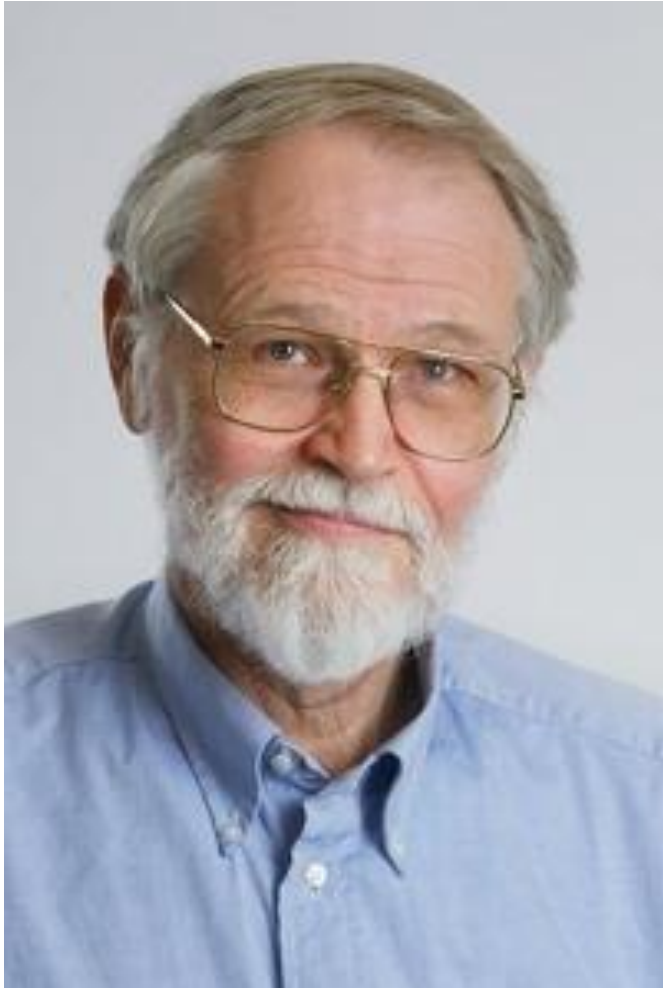
Process Models: Commentary

“How much of what software engineers now do is still devoted to the **accidental**, as opposed to the **essential**? Unless it is more than 9/10 of all effort, shrinking all the **accidental** activities to zero time will not give an order of magnitude improvement.”

“There is **no single development**, in either technology or management technique, which by itself **promises even one order of magnitude improvement** in productivity, in reliability, in simplicity.”

Frederick Brooks.
The Mythical Man Month: Essays on Software Engineering
Addison-Wesley. New York. 1995.

Process Models: Commentary



Brian
Kernighan

Process Models: Commentary

Software Methodology and Snake Oil

- Each methodology has the germ of a useful idea
- Each claims to solve major programming problems
- Some are promoted with religious fervor
- In fact most don't seem to work well
- Or don't seem to apply to all programs
- Or can't be taught to others
- A few are genuinely useful and should be part of everyone's repertoire

Process Models: Commentary

- In summary...
- (Kernighan) Some process models offer good ideas, but...
- (Brooks) Software development is inherently hard, and...
- (Kernighan) Many process models are over-hyped, so...
- (Kernighan) View process models with healthy skepticism

Process Models: Commentary

- Every project is unique
 - Choose a process model that fits the project
 - Be willing to customize that process model

Process Models: Commentary

- Core points:
 - **Requirements:** First determine **who** the users are and **what** your system should do for them
 - **Involve the users!!!**
 - **Design:** Then determine **how** you want your system to work
 - **Implement, test:** Then code and test your system
 - **Evaluate:** Then evaluate your system
 - **Involve the users!!!**
 - Iterate as often as you reasonably can

Summary

- We have covered these software engineering topics:
 - (1) Requirements analysis
 - (2) Design
 - (3) Implementation
 - (4) Debugging
 - (5) Testing
 - (6) Evaluation
 - (7) Maintenance
 - (8) Process models