Object-Oriented Software Design **and Software Processes**

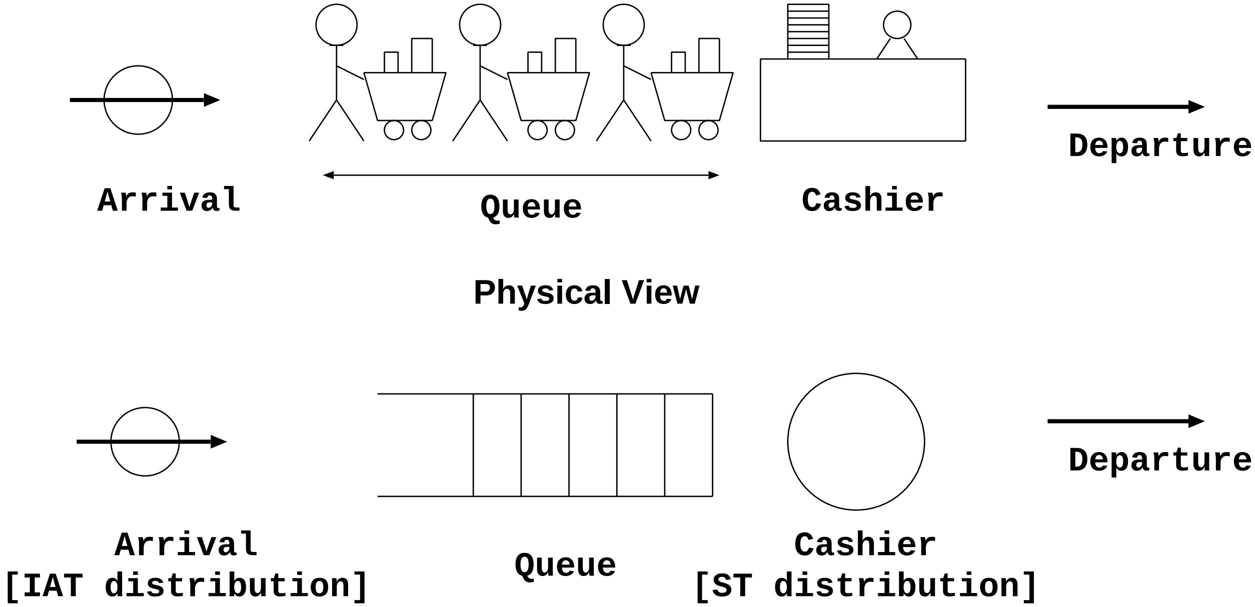
Hans Vangheluwe

Modelling, Simulation and Design Lab (MSDL)
School of Computer Science, McGill University, Montréal, Canada

Overview

1. (Software) Process: definition
2. Various Software Processes
3. The Process Influences Productivity:
   Dynamic Process Modelling using Forrester System Dynamics

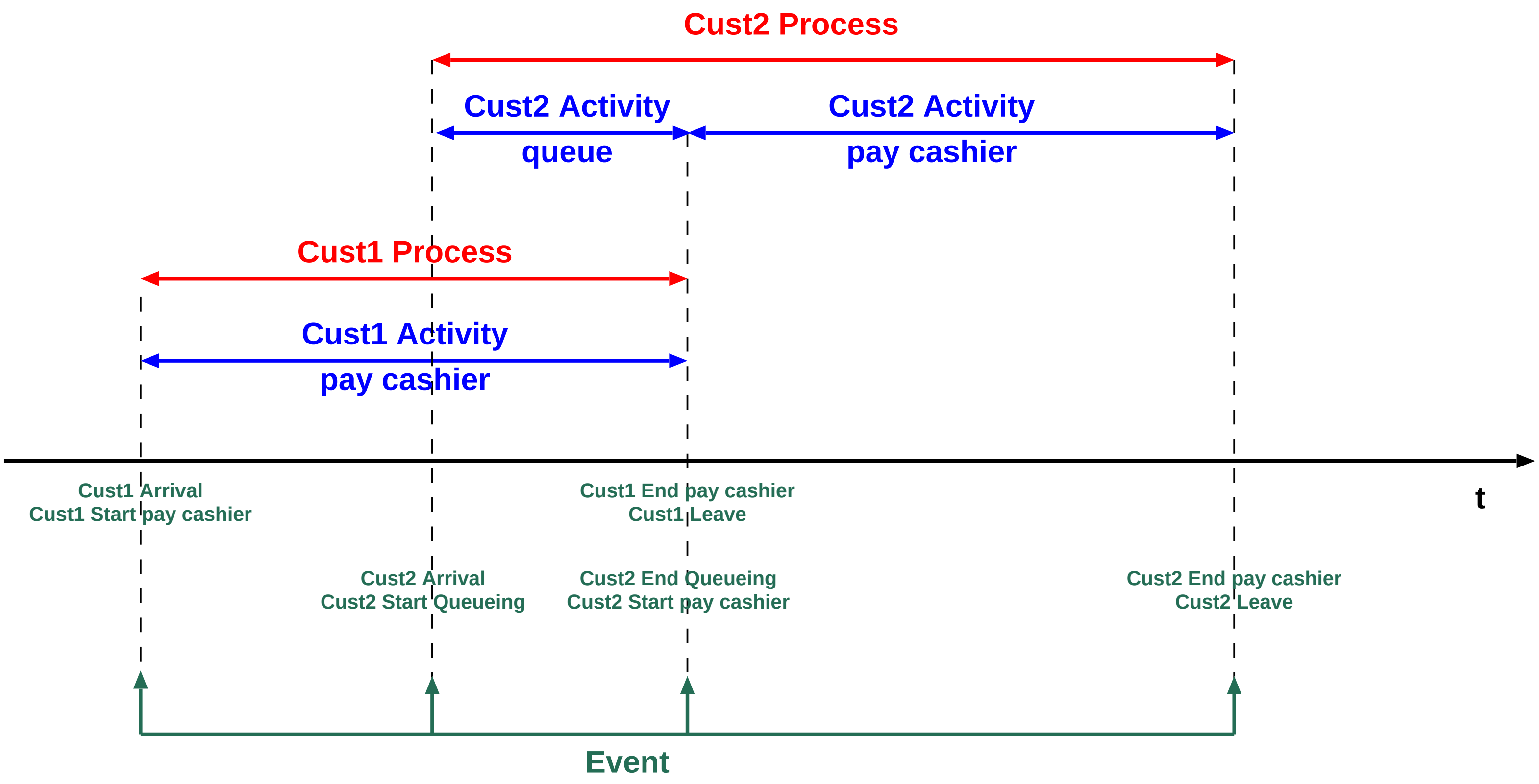
Process: A Queueing System



Physical View

Abstract View

Event/Activity/Process



Software Processes

“The Software Engineering **process** is the total set of Software Engineering **activities** needed to transform requirements into software”.

Watts S. Humphrey. Software Engineering Institute, CMU.

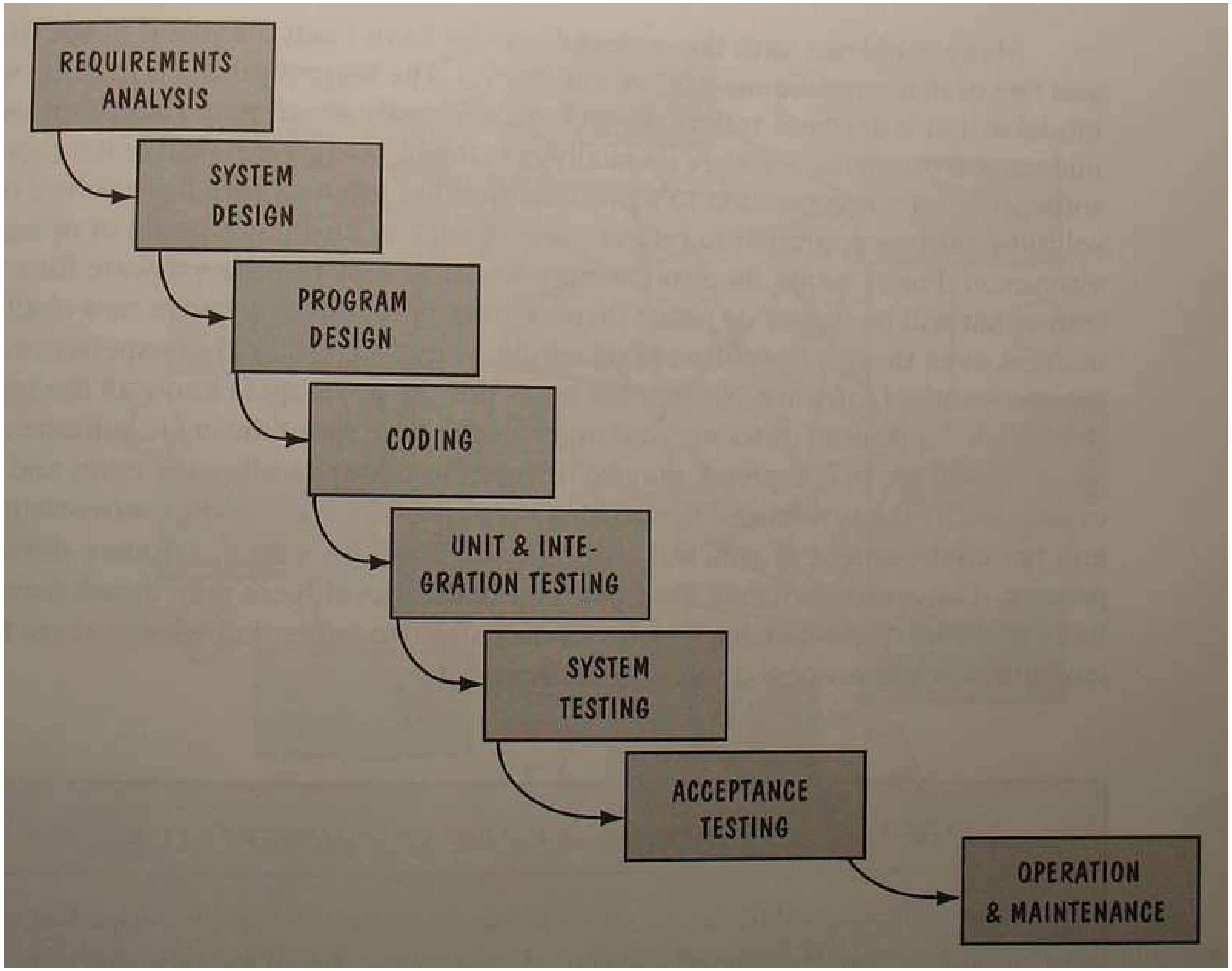
(`portal.acm.org/citation.cfm?id=75122`)

Software Processes

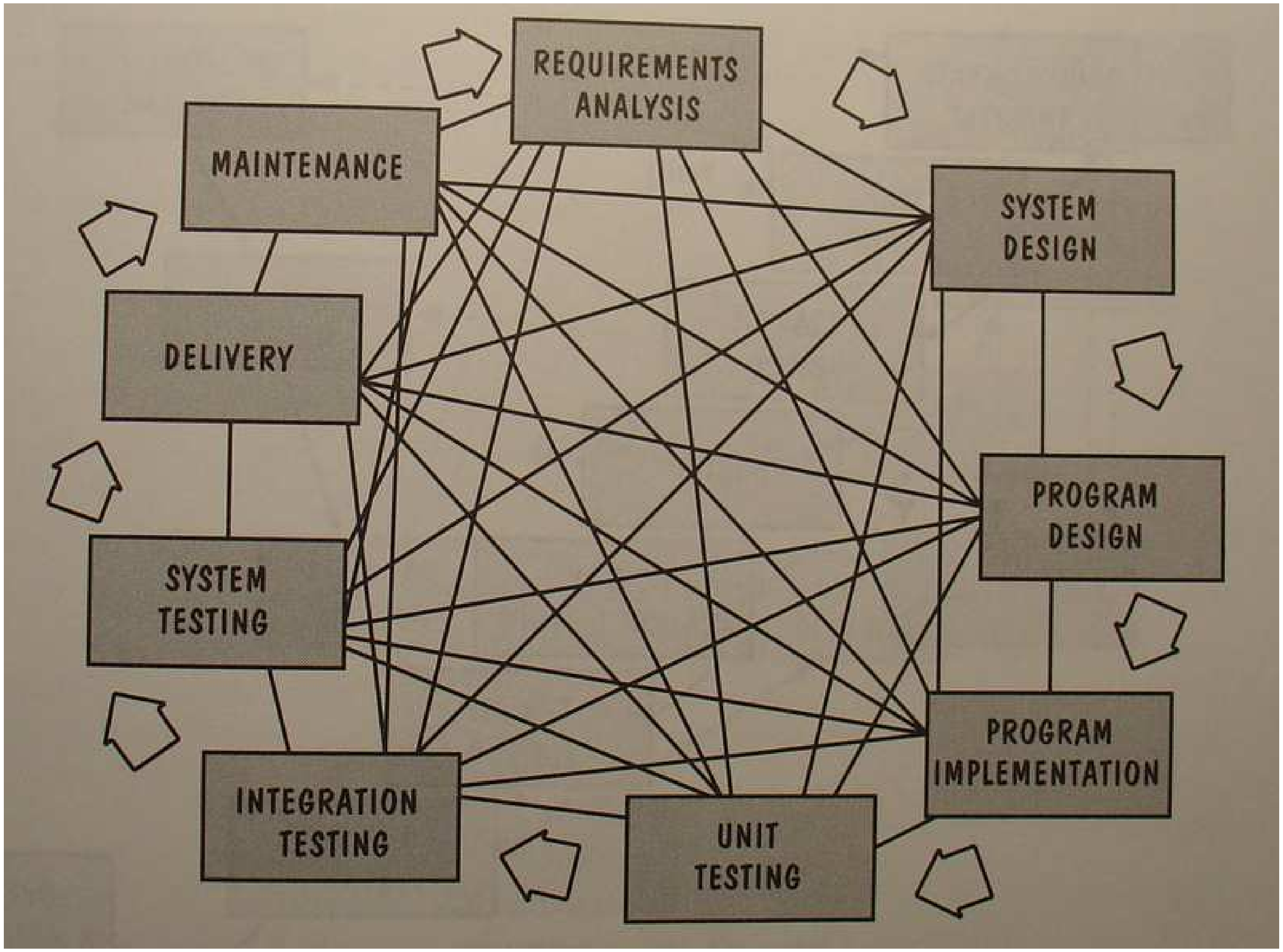
- Waterfall (Royce)
- V Model (German Ministry of Defense)
- Prototyping
- Operational Specification (Zave)
- Transformational (automated software synthesis) (Balzer)
- Phased Development: Increment and Iteration
- Spiral Model (Boehm)
- Rational Unified Process (RUP)
- Extreme Programming (XP)
- System Dynamics (Dynamic Process Model)
  (see Process $\sim$ Productivity)

Shari Lawrence Pfleeger. Software Engineering: Theory and Practice (Second Edition). Prentice Hall. 2001.
Chapter 2: Modelling the Process and Life Cycle.

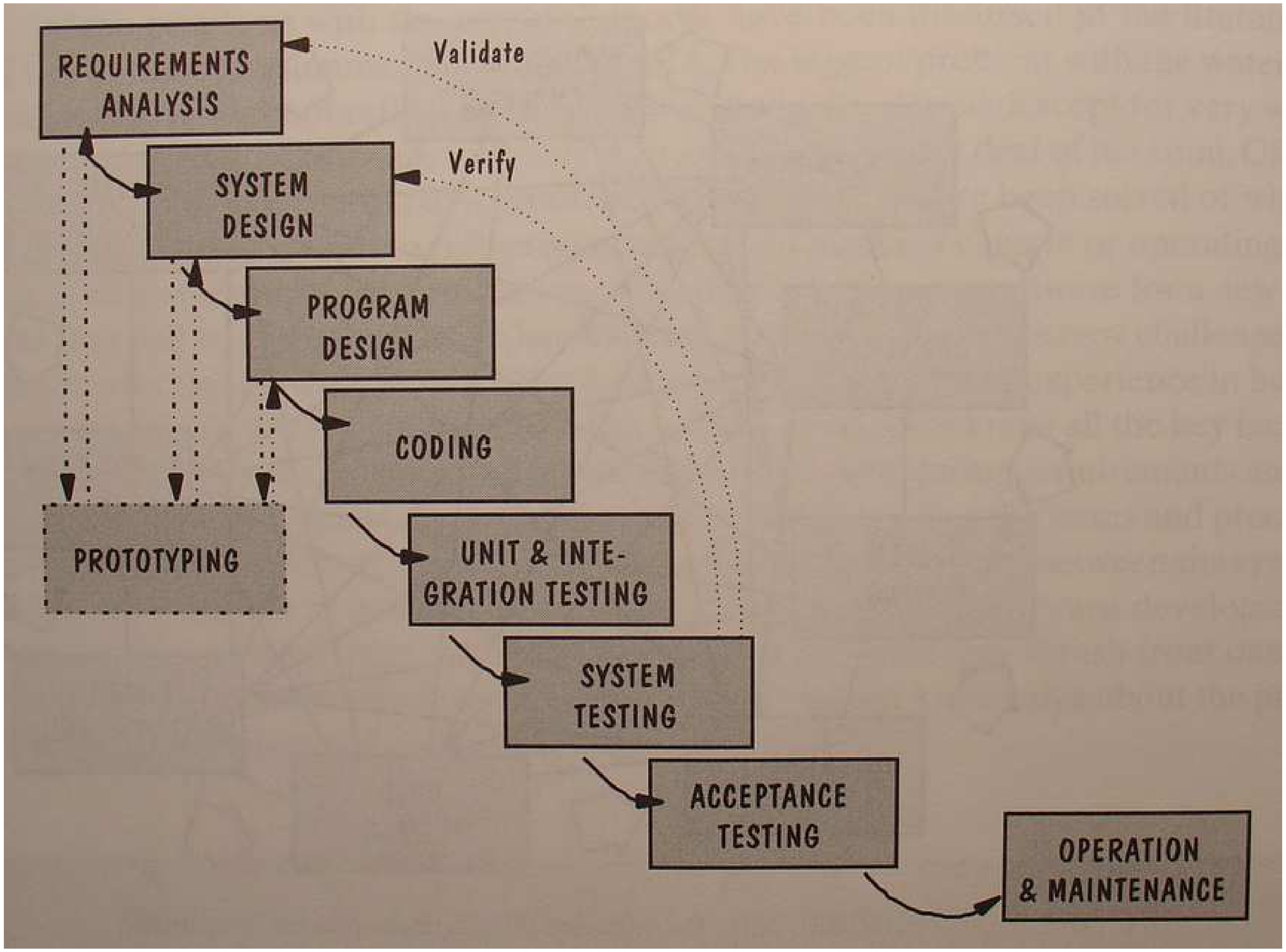
Waterfall Process (Royce)



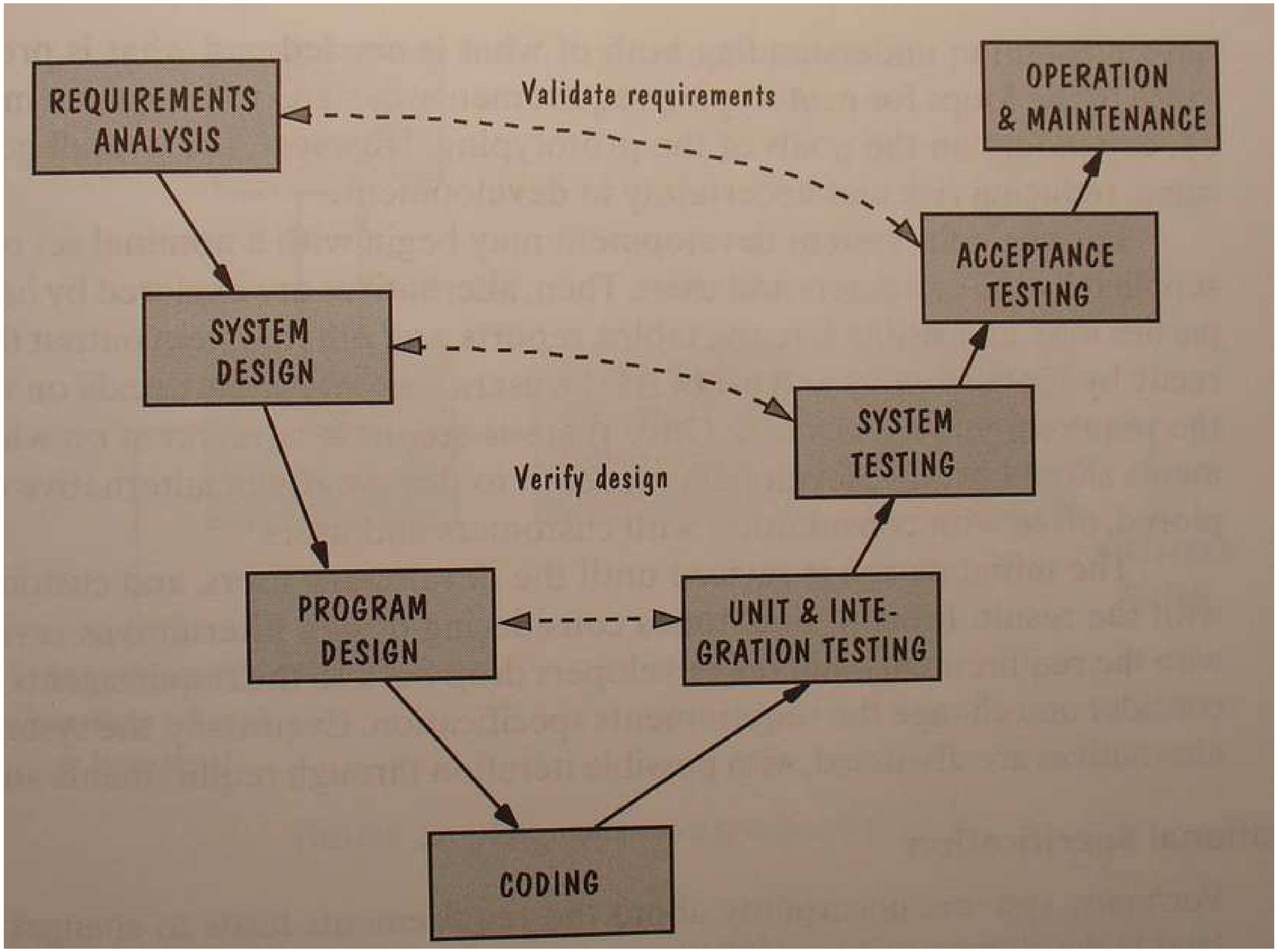
Waterfall Process in Reality



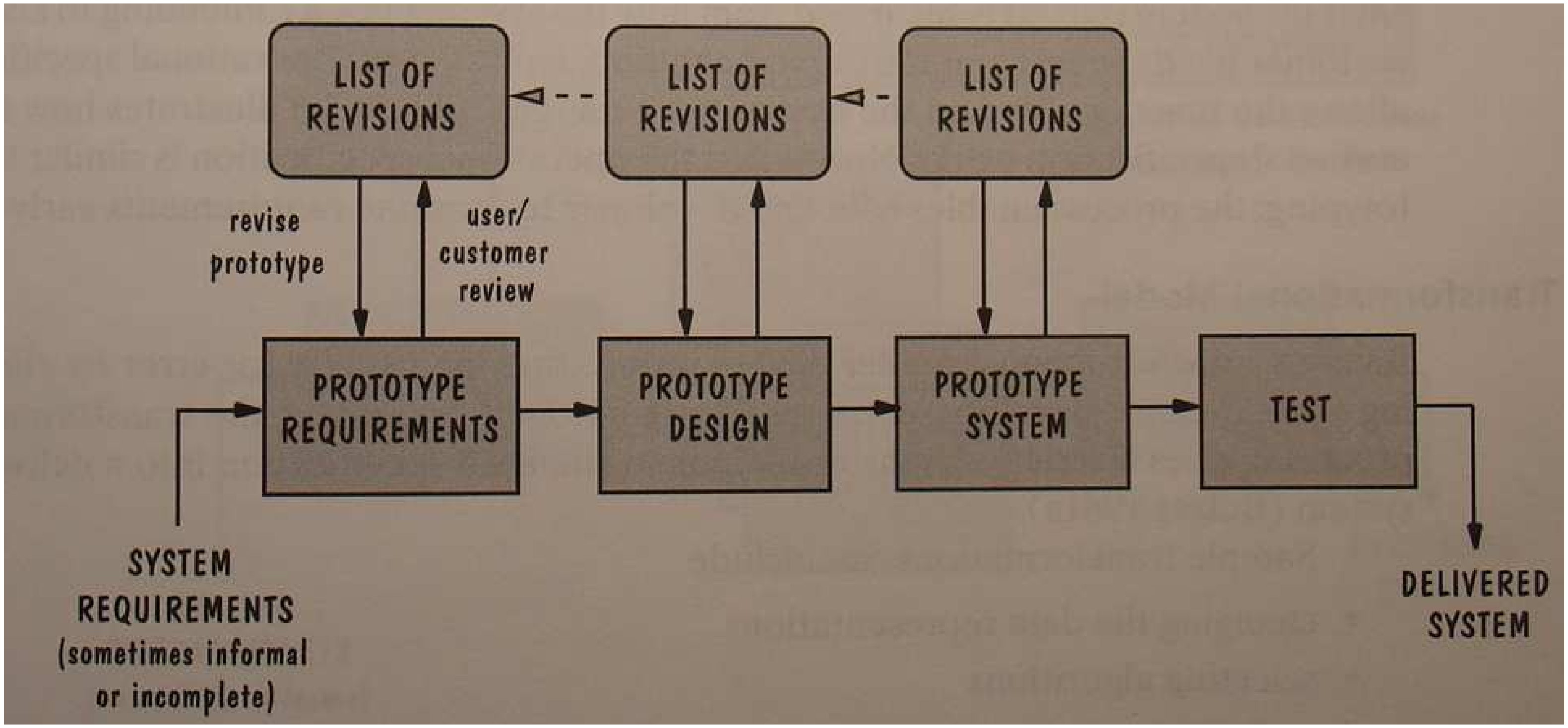
Waterfall Process with Prototyping



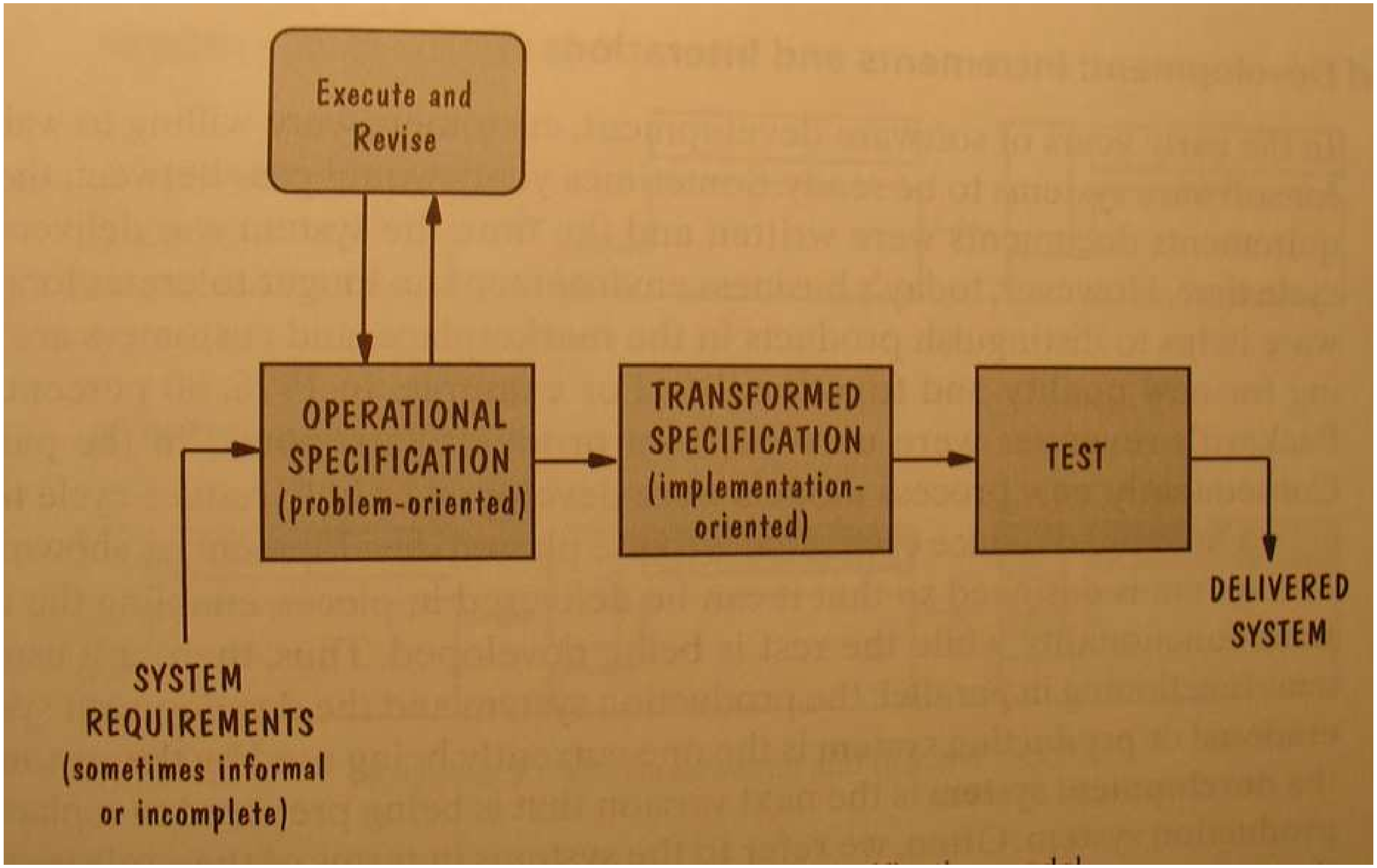
V Model (German Ministry of Defense)



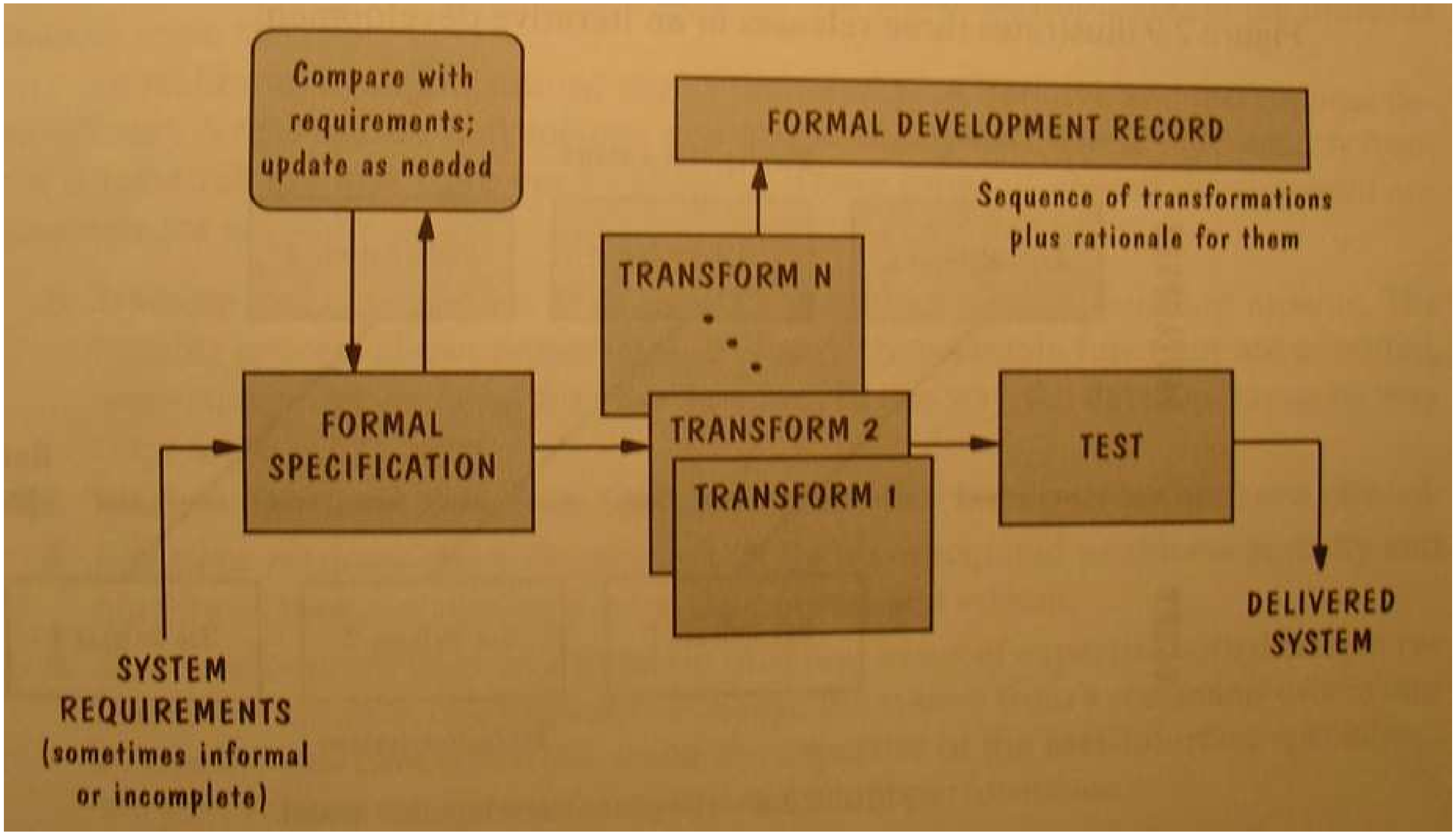
Prototyping Process



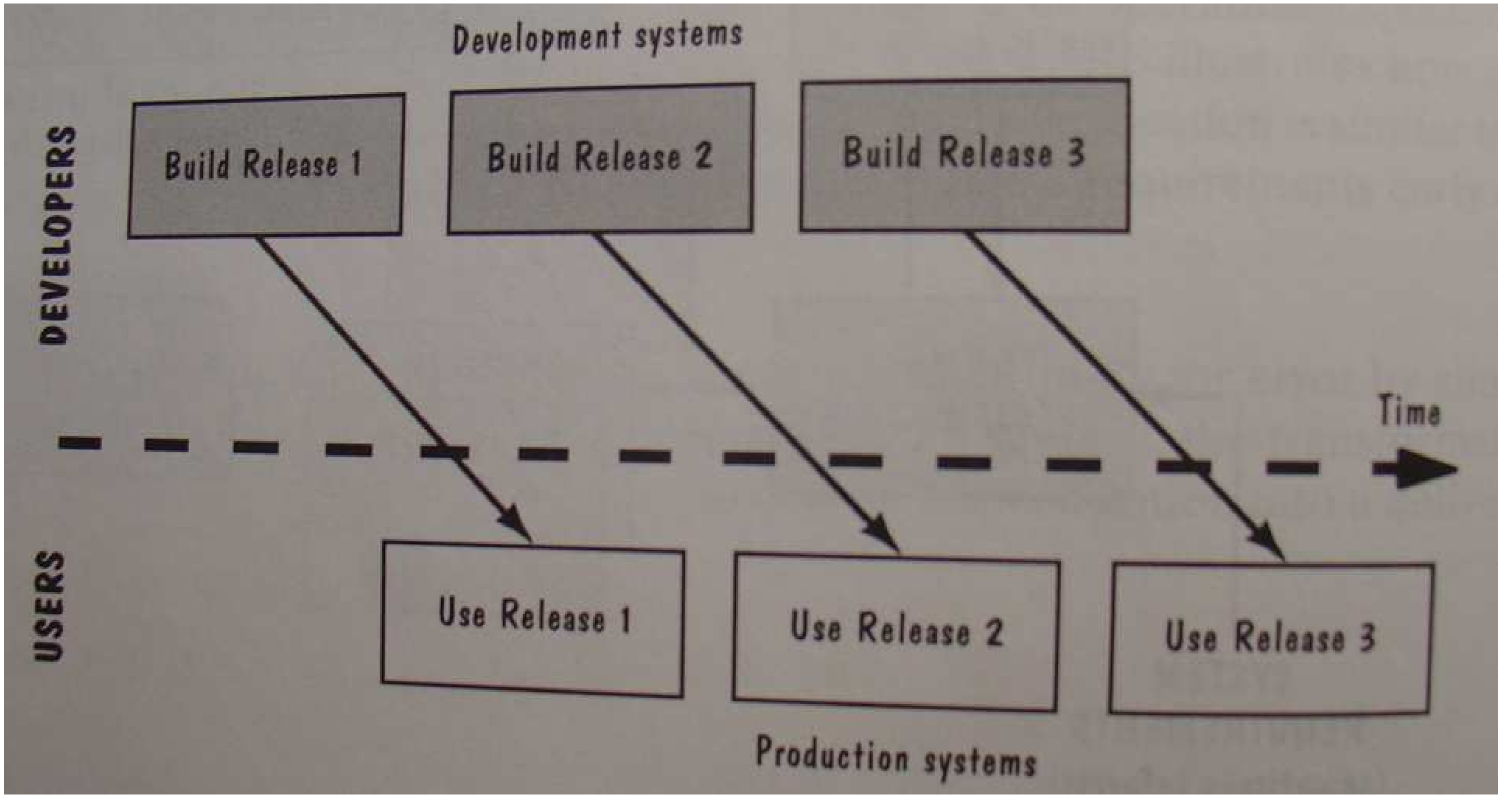
Operational Specification Process (Zave)


Execute and Revise
OPERATIONAL SPECIFICATION (problem-oriented)
TRANSFORMED SPECIFICATION (implementation-oriented)
TEST
DELIVERED SYSTEM
SYSTEM REQUIREMENTS (sometimes informal or incomplete)

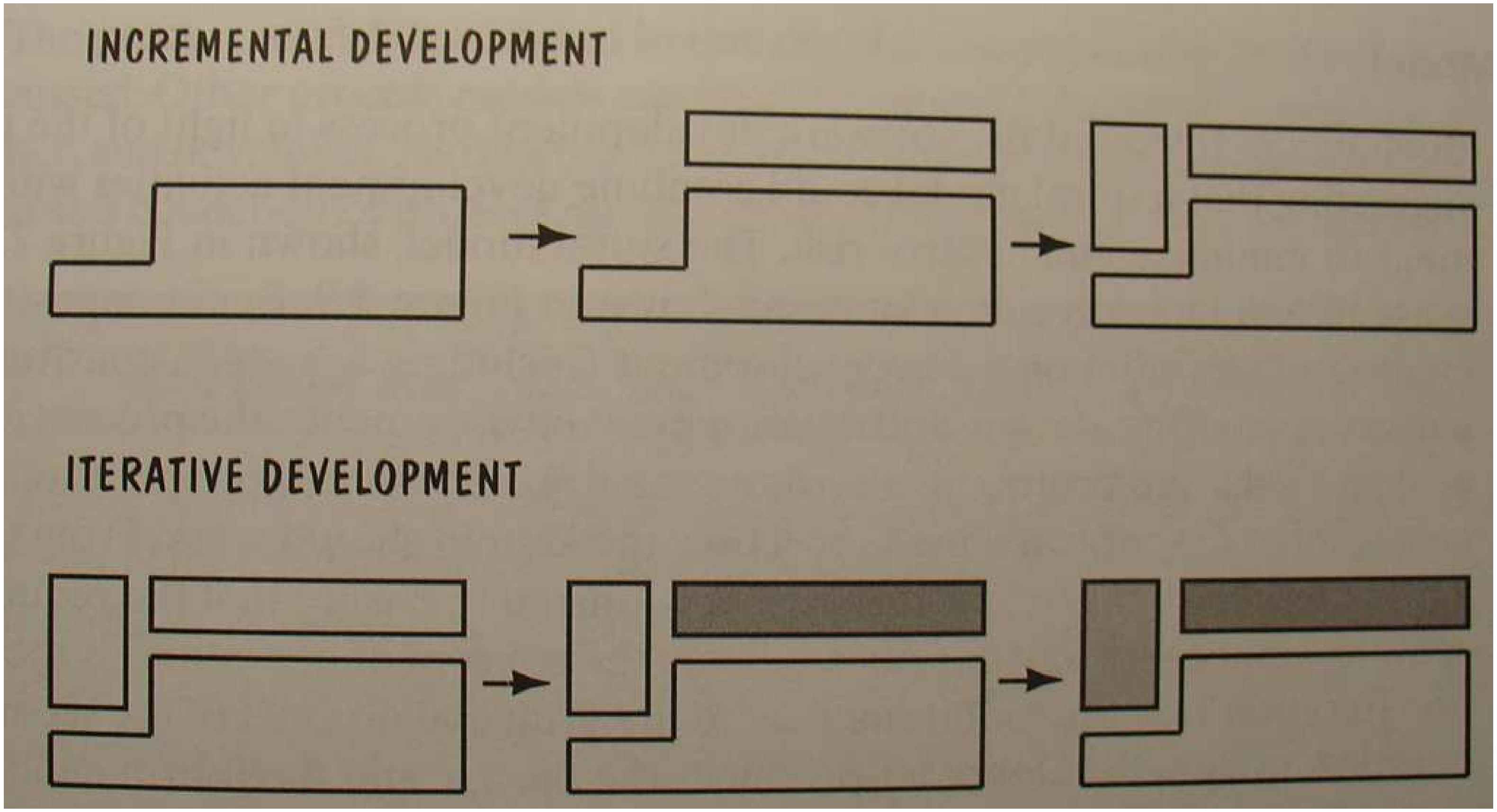
Transformational Process


Compare with requirements; update as needed
FORMAL DEVELOPMENT RECORD
Sequence of transformations plus rationale for them
TRANSFORM N
FORMAL SPECIFICATION
TRANSFORM 2
TRANSFORM 1
TEST
DELIVERED SYSTEM
SYSTEM REQUIREMENTS (sometimes informal or incomplete)

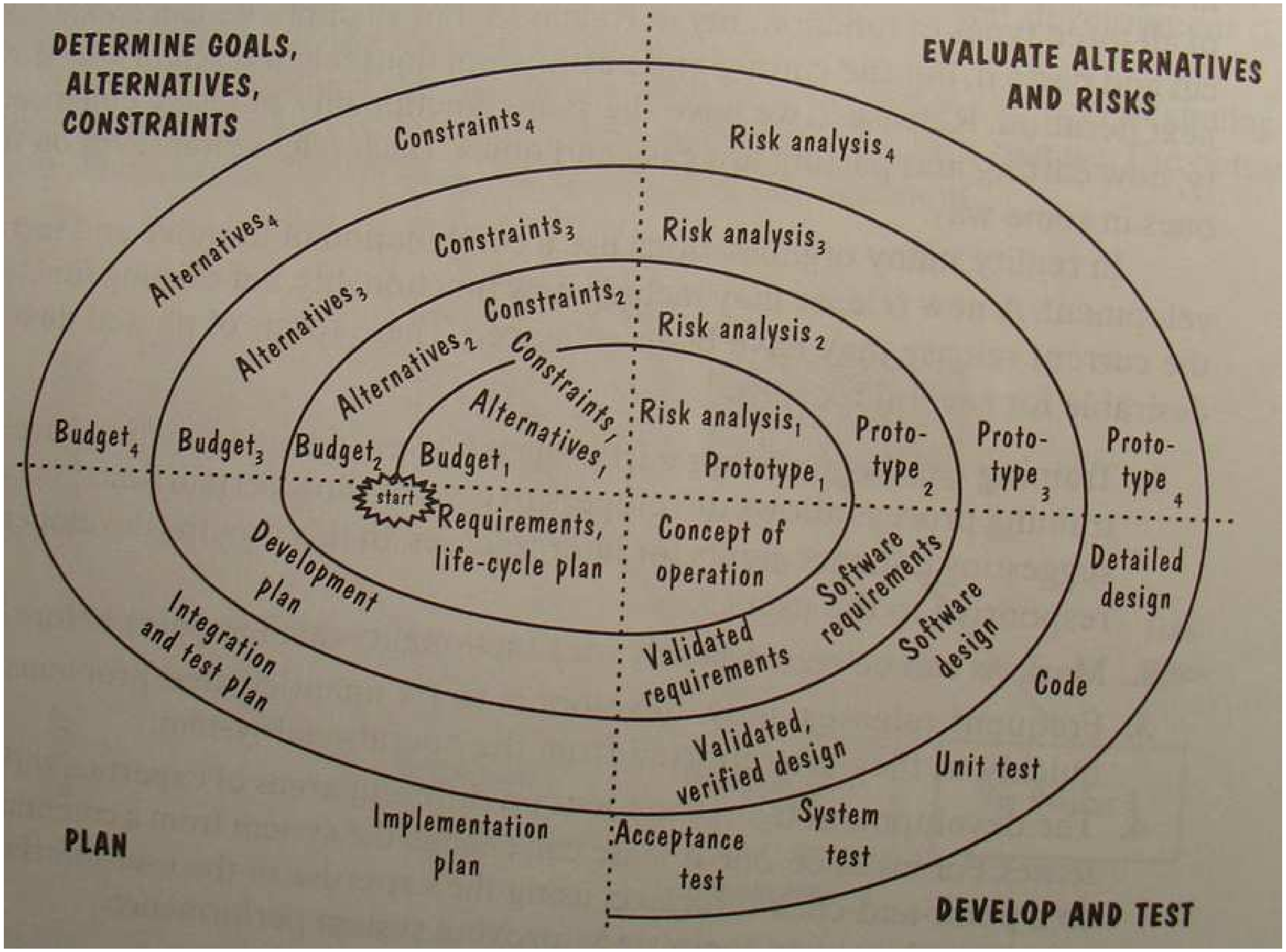
Phased Development Process



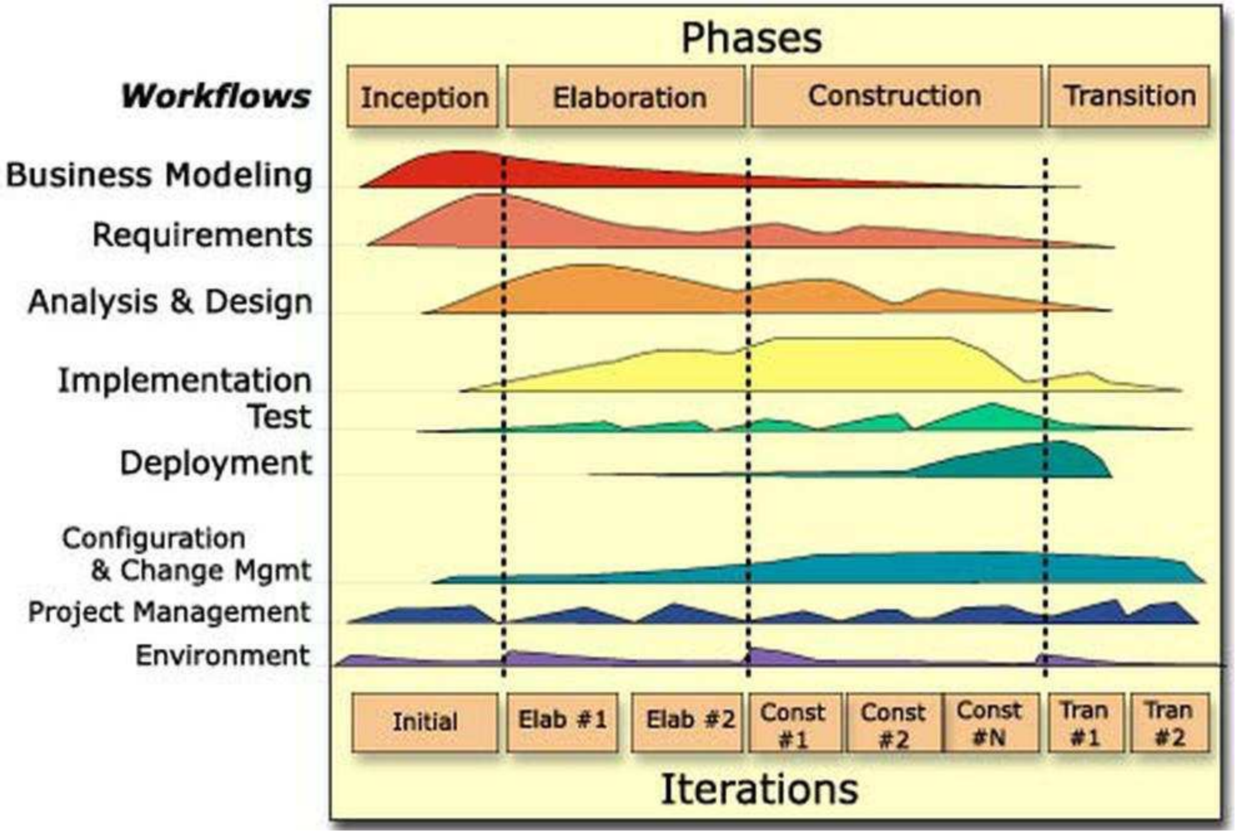
Phased Development: Incremental vs. Iterative



Spiral Model (Boehm)

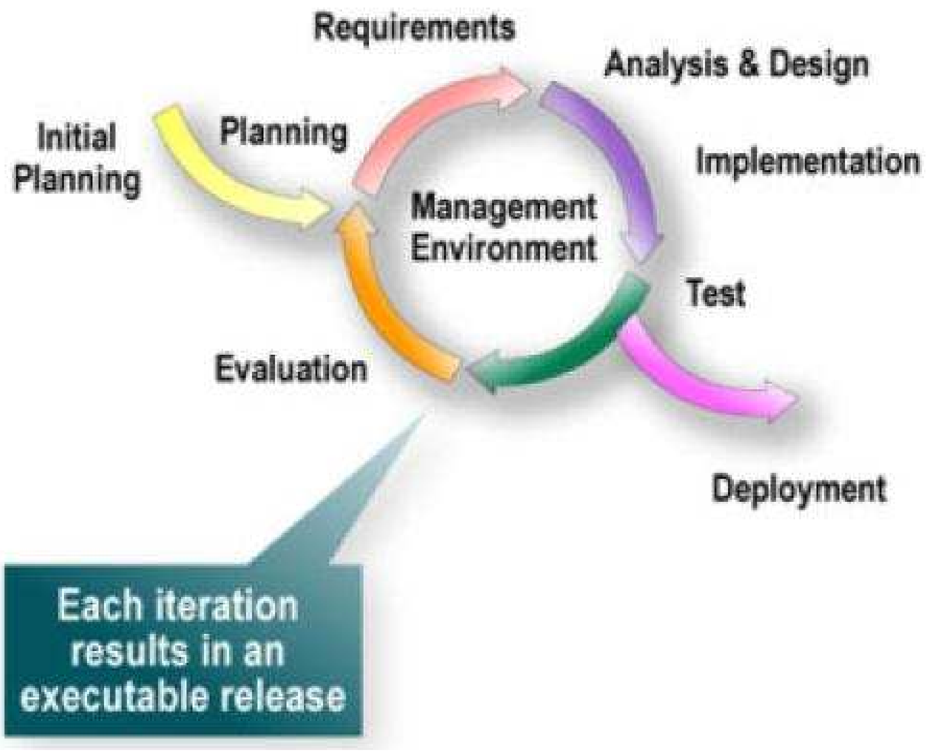


The Rational Unified Process (RUP): Activity Workload as Function of Time



The (Rational) Unified Process ((R)UP): Empirical Observations

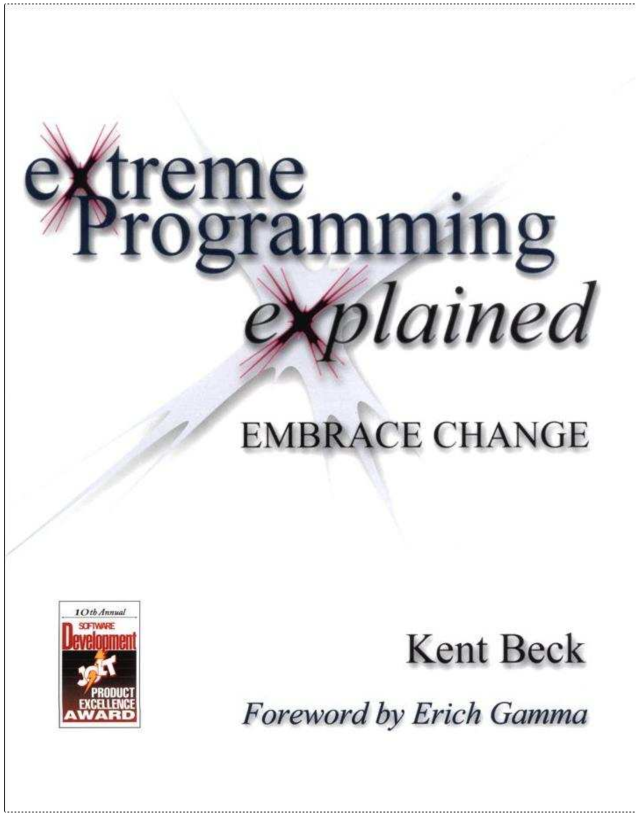
1. Waterfall-like **sequence** of
   Requirements, Design, Implementation, Testing.
2. Not pure waterfall:
   - **Phased Development** (**iterative**)
   - Overlap (**concurrency**) between activities
3. Testing:
   - **Regression** (test not only newly developed, but also previously developed code)
   - Testing starts **before** design and coding (Extreme Programming)



Use:

- descriptive
- prescriptive
- proscriptive

Extreme Programming (XP)



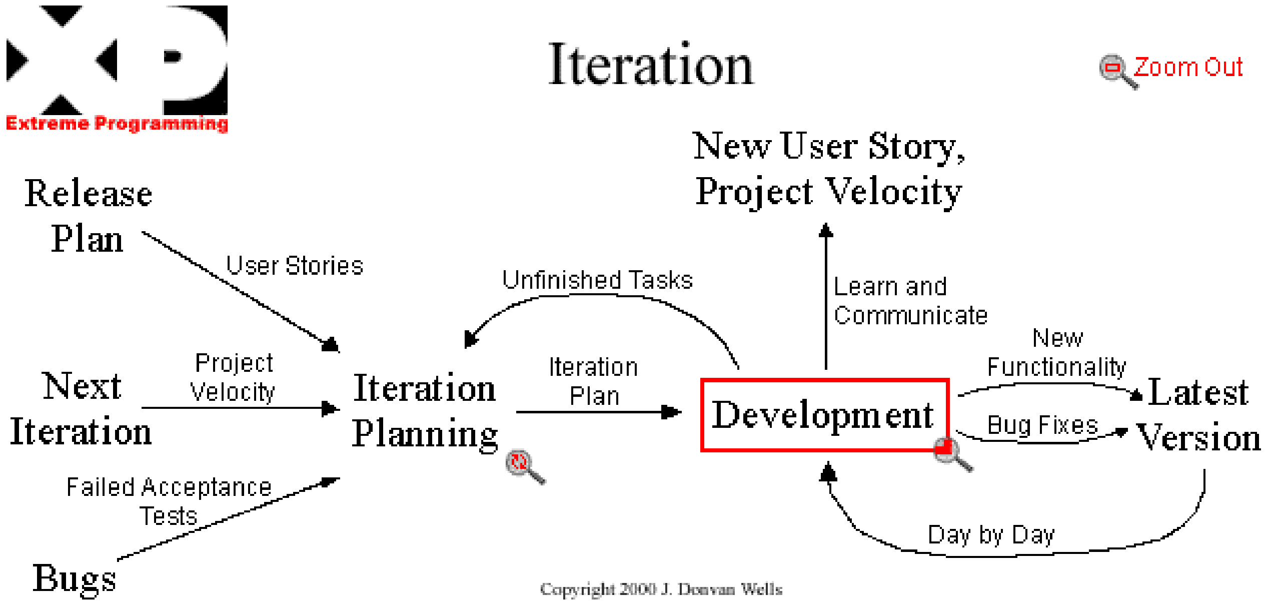
(`www.extremeprogramming.org`)

Extreme Programming (XP) highlights

User Stories are written by the customers as things that the system needs to do for them (requirements). They drive the creation of acceptance **tests**.

Extreme Programming (XP) Process

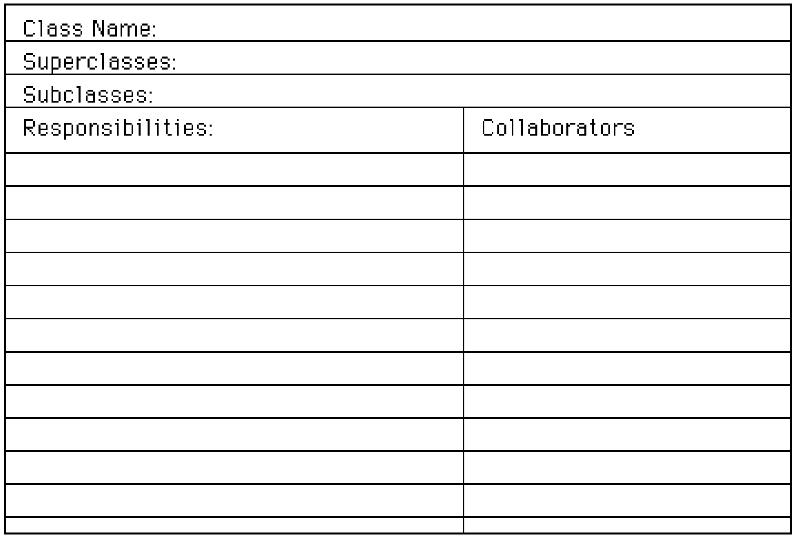
The project is divided into **Iterations**.



The "inner loop" is a **daily** cycle!

Extreme Programming (XP) highlights

Use Class, Responsibilities, and Collaboration **(CRC) Cards** to **design** the system.



Class Name:	
Superclasses:	
Subclasses:	
Responsibilities:	Collaborators

Extreme Programming (XP) highlights

- Code the Unit Test **first** (from requirements/user stories).
- **All code** must have Unit Tests; All code must pass **all** unit tests before it can be released.

Extreme Programming (XP) highlights

Refactor whenever and wherever possible.

- for readability ($\sim$ maintanability)
- for re-use
- for optimization
- ...

Refactoring **code** or **design**.

Catalog of Refactoring Patterns (rules):
`http://www.refactoring.com/catalog/`

Refactoring Pattern: Reverse Conditional

- Motivation: increase clarity.
- Mechanics: (1) remove negative from conditional; (2) Switch clauses.
- Example:

```
if ( !isSummer( date ):
 charge = winterCharge( quantity )
else:
 charge = summerCharge( quantity )
```

⇒

```
if ( isSummer( date ) ):
 charge = summerCharge( quantity )
else:
 charge = winterCharge( quantity )
```

Refactoring Pattern: Consolidate Duplicate Conditional Fragments

- Motivation: increase clarity, performance optimization.
- Mechanics: lift commonality out of conditional.
- Example:

```
if (isSpecialDeal()):
  total = price * 0.95
  send()
else:
  total = price * 0.98
  send()
```

⇒

```
if (isSpecialDeal()):
  total = price * 0.95
else:
  total = price * 0.98
send()
```

Refactoring Pattern: Split Loop

- Motivation: increase clarity (**not** performance optimization (yet)).
- Mechanics: lift commonality out of conditional.
- Example:

```
def printValues:

 averageAge = 0
 totalSalary = 0
 for person in people:
   averageAge  += person.age
   totalSalary += person.salary
 averageAge = averageAge / people.length
 print averageAge
 print totalSalary
```

⇒

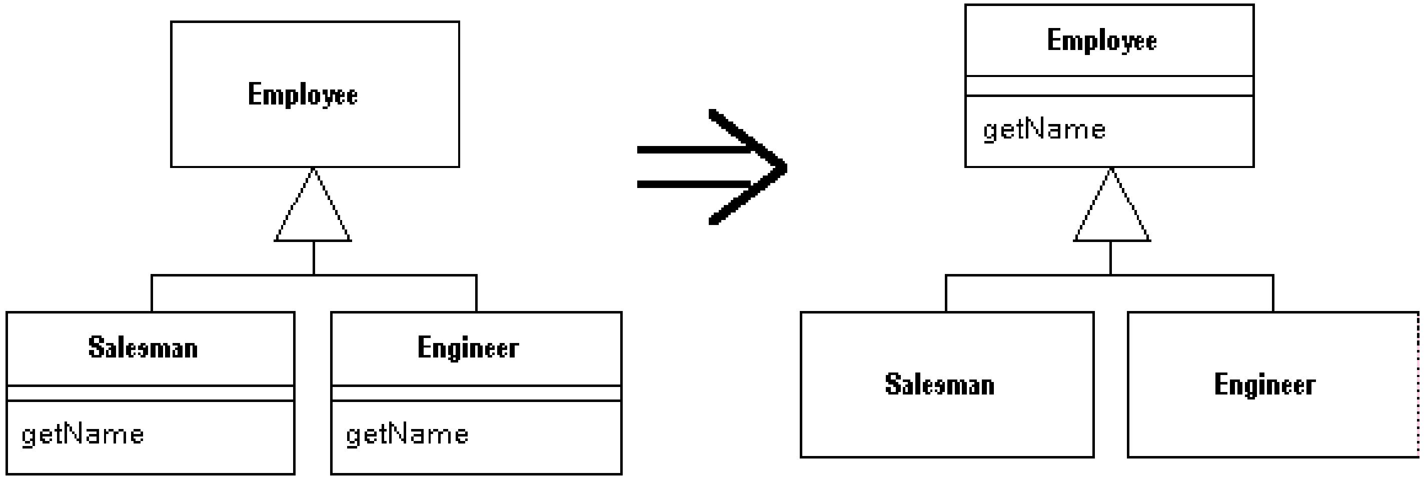
```
def printValues:

 averageAge = 0
 for person in people:
   averageAge  += person.age
 averageAge = averageAge / people.length
 print averageAge

 totalSalary = 0
 for person in people:
   totalSalary += person.salary
 print totalSalary
```

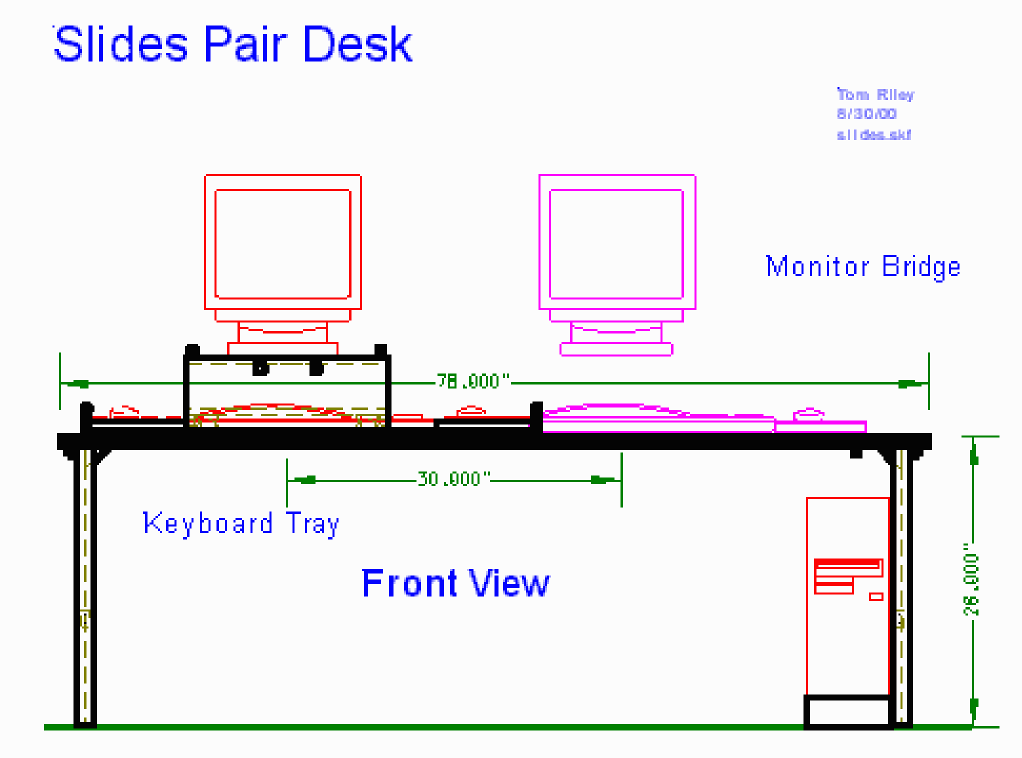
Refactoring Pattern: Pull Up Method

- Motivation: re-use.
- Mechanics: pull up identical (type-wise) methods from (all) sub-classes.
- Example:



Extreme Programming (XP) highlights

Pair Programming

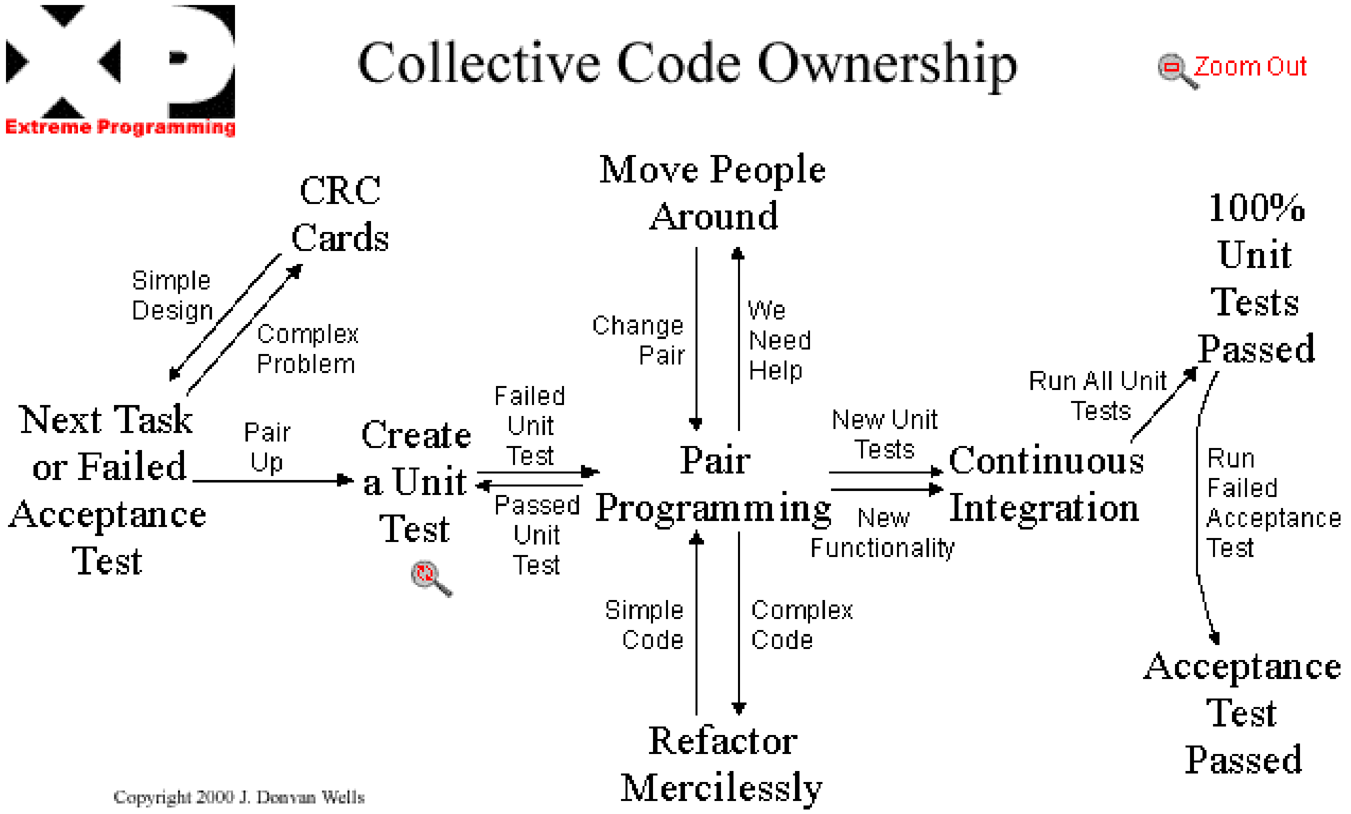


(`www.charm.net/~jriley/pairall.html`)

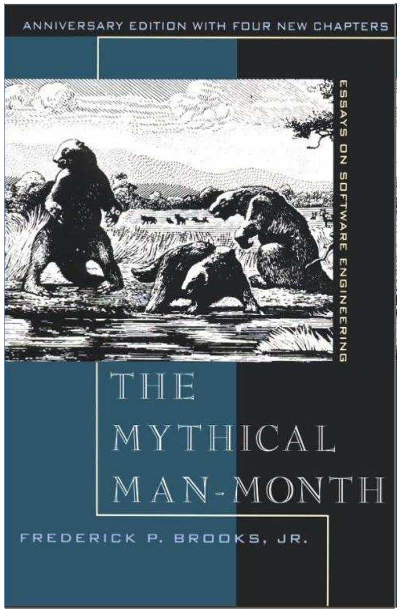
Advantages:

- Higher Quality
- Collective Ownership of code/design
- Productivity Increase (“flow”) thanks to programmer/backseat pair
- Learning/Training
- . . .

Extreme Programming (XP) Process



The Process influences Productivity

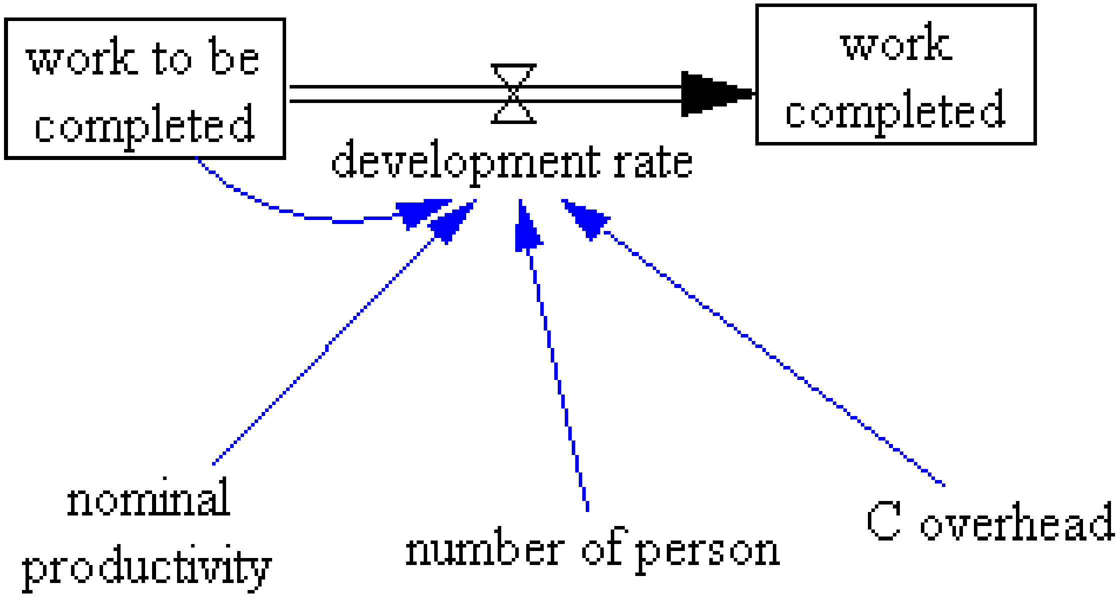

ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

“Adding manpower to a late software project makes it later”

Fred Brooks. The Mythical Man-Month.

(`www.ercb.com/feature/feature.0001.html`)

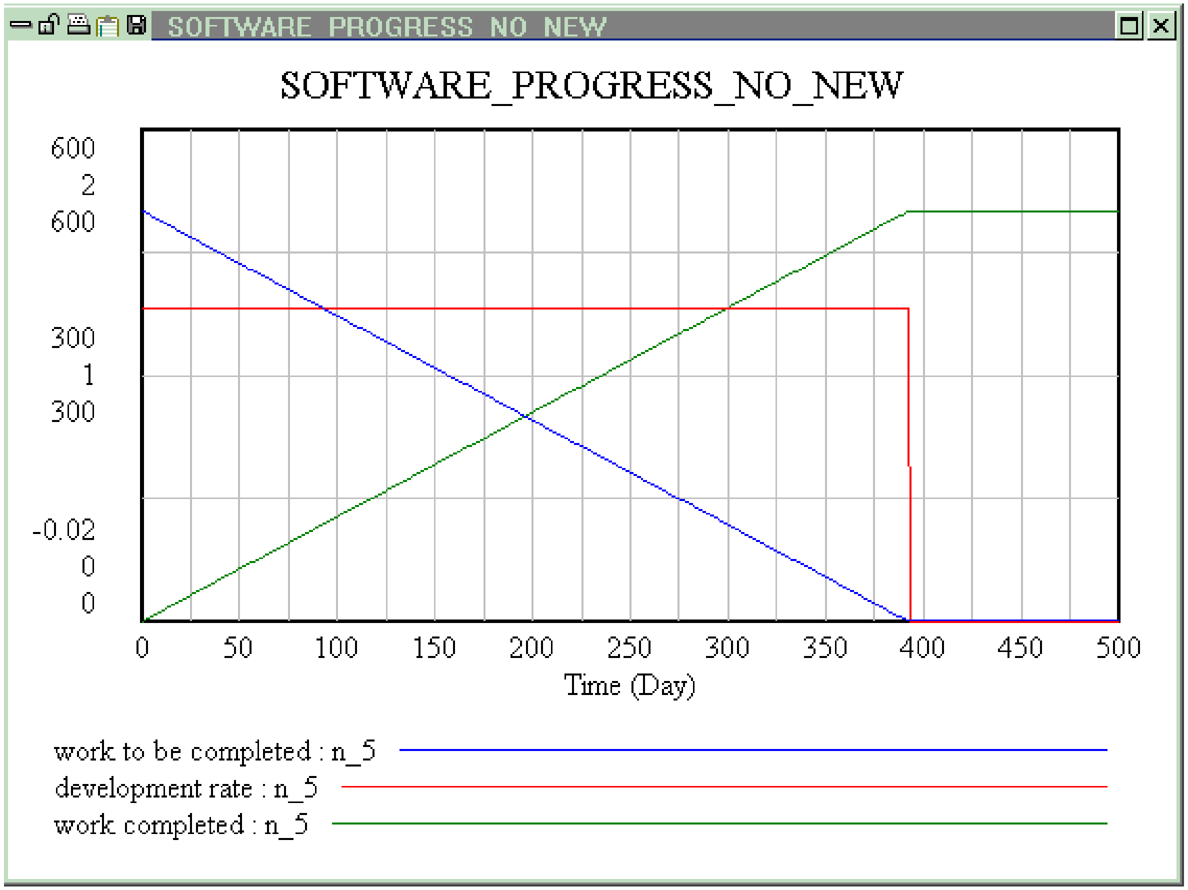
Why Brooks' Law ? Team Size.



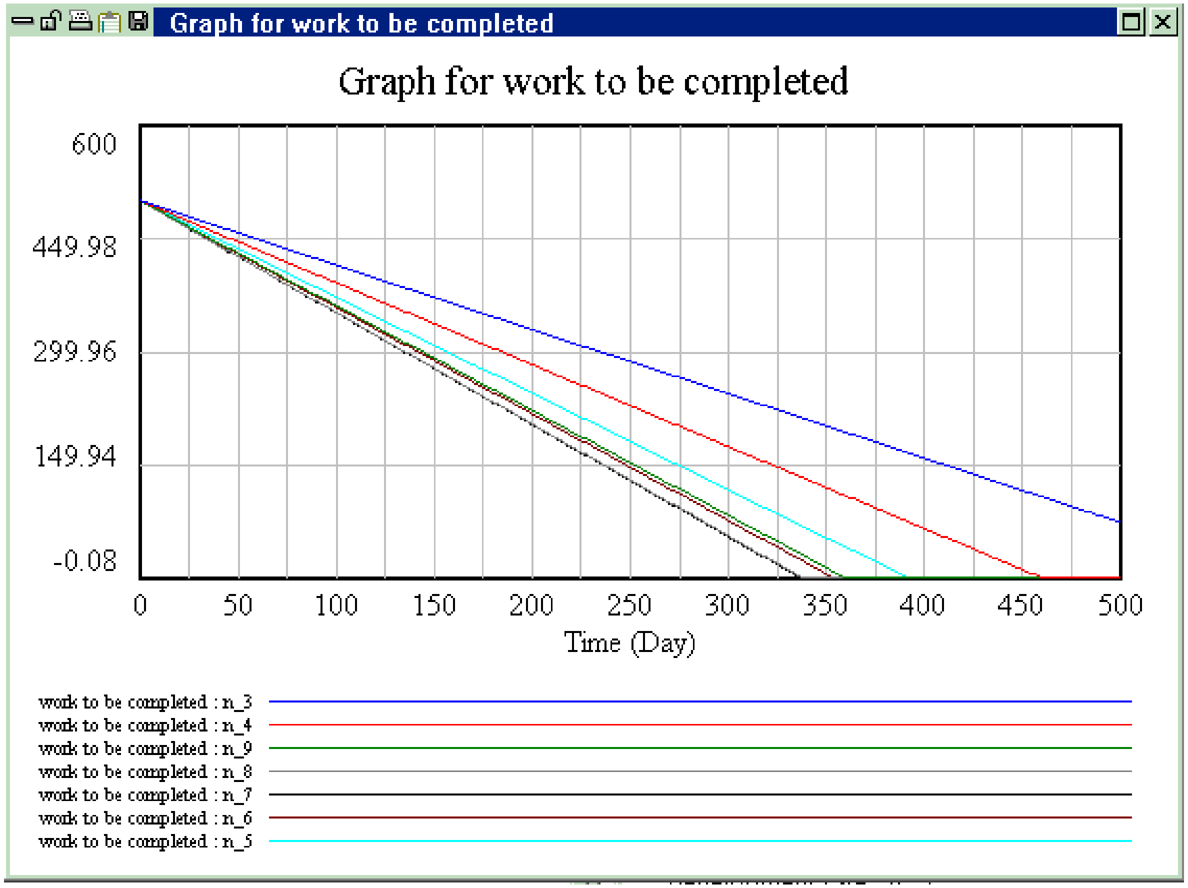
Model in **Forrester System Dynamics**
using `Vensim PLE` (`www.vensim.com`)

```
development rate =
 nominal_productivity* (1-C_overhead*(N*(N-1)))*N
```

Team Size N = 5

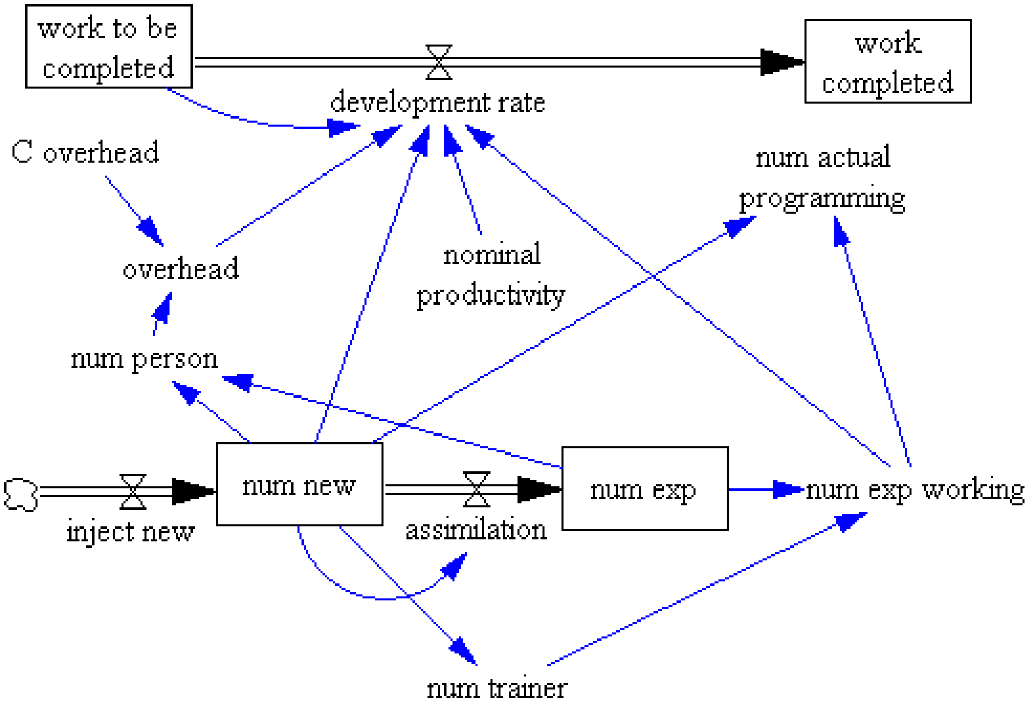

SOFTWARE_PROGRESS_NO_NEW
SOFTWARE_PROGRESS_NO_NEW
600
2
600
300
1
300
-0.02
0
0
0
50
100
150
200
250
300
350
400
450
500
Time (Day)
work to be completed : n_5
development rate : n_5
work completed : n_5

Team Size N = 3 . . . 9



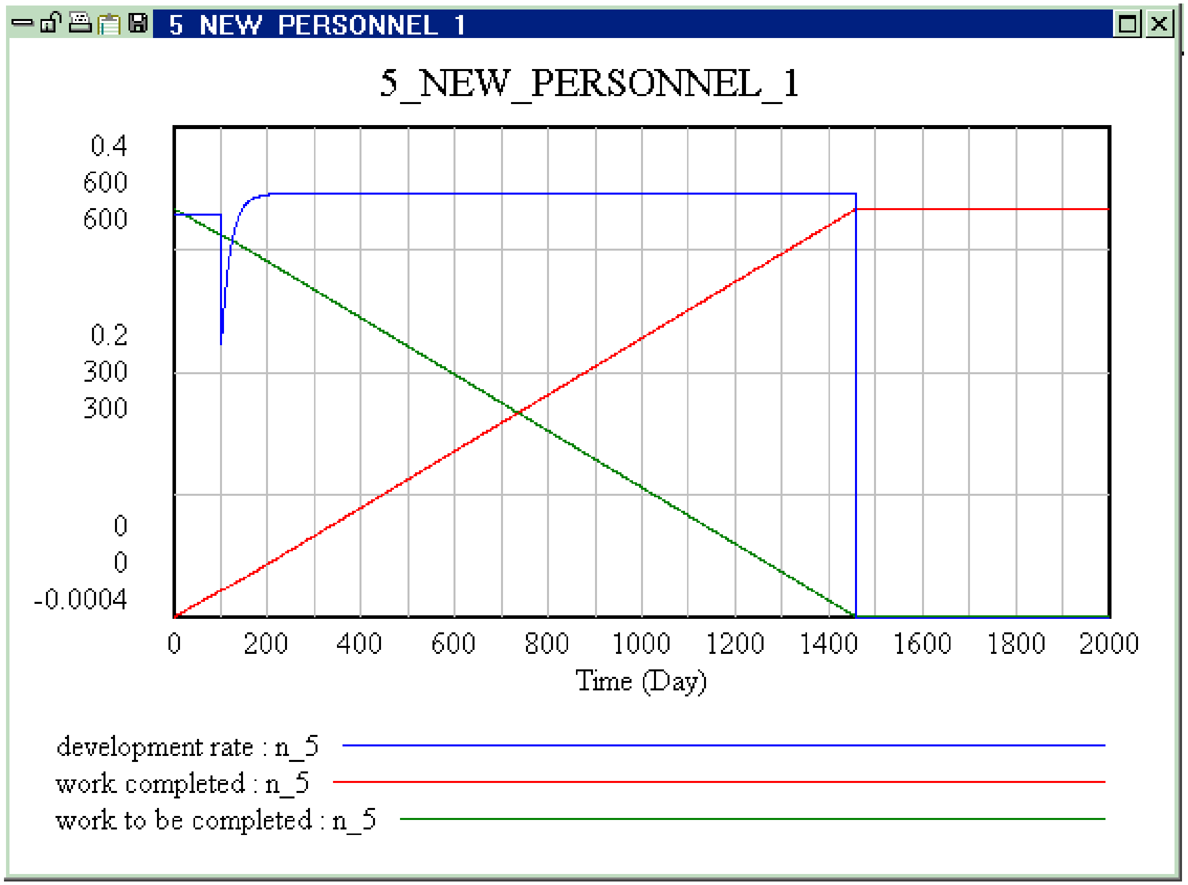
Optimal Team Size between 7 and 8

The Effect of Adding New Personnel (FSD model)

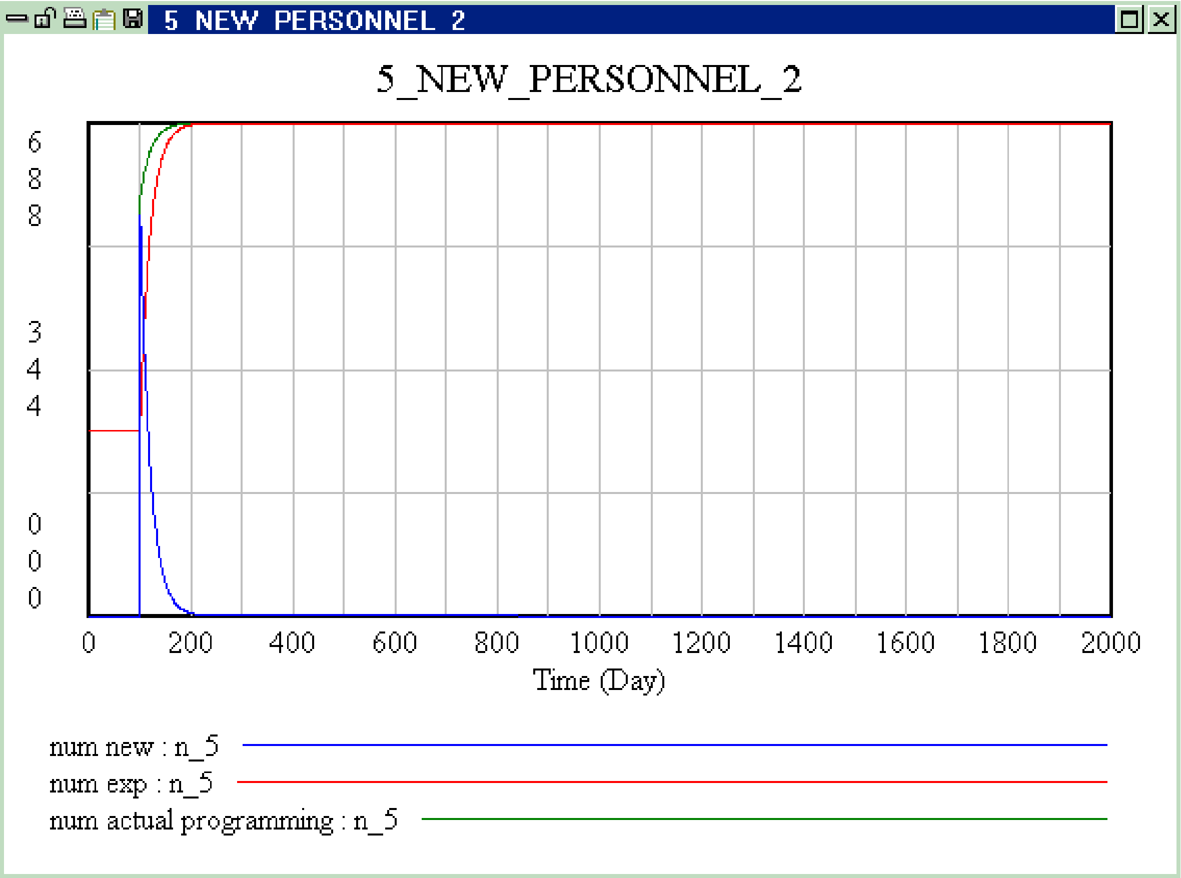


```
development rate = nominal_productivity*
 (1-C_overhead*(N*(N-1)))* (1.2*num_exp_working + 0.8*num_new)
```

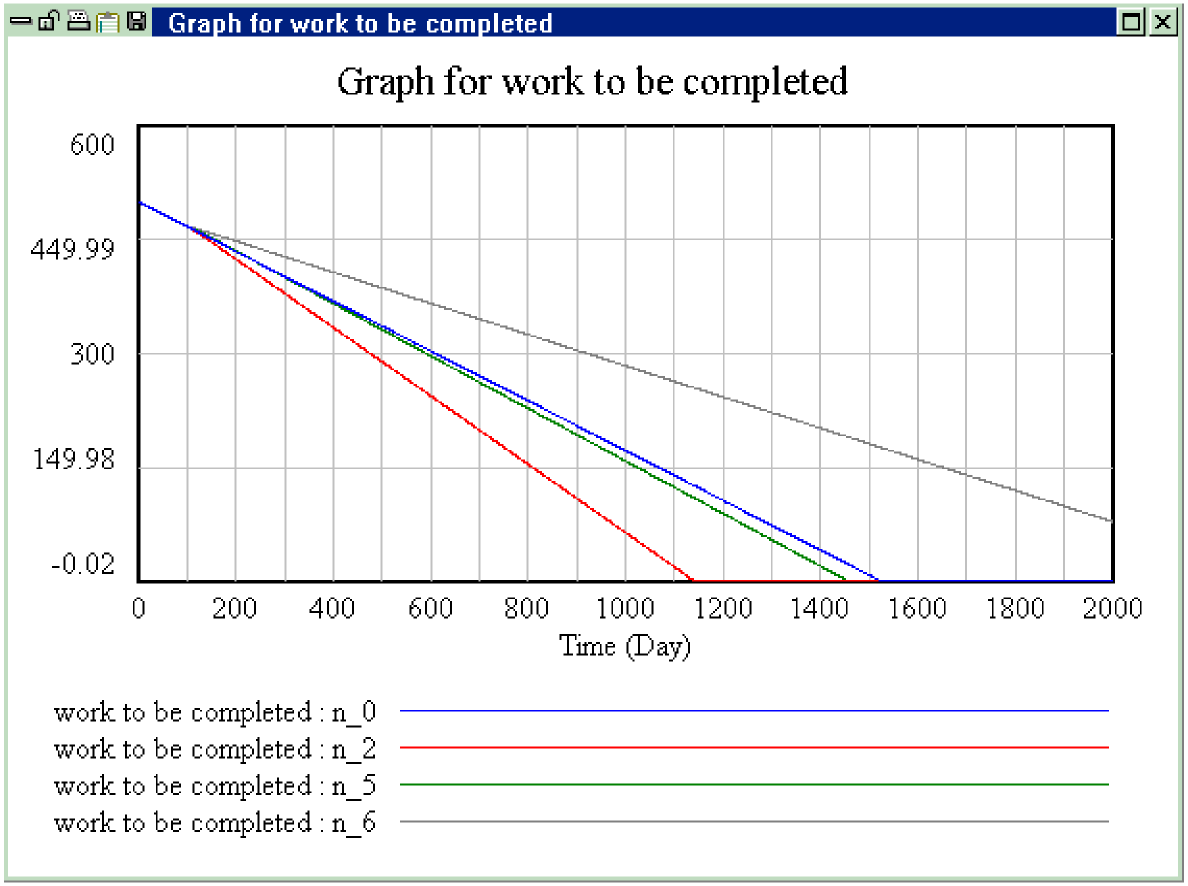
5 New Programmers after 100 days



5 New Programmers after 100 days



0 . . . 6 New Programmers after 100 days


Graph for work to be completed
Graph for work to be completed
Time (Day)
work to be completed : n_0
work to be completed : n_2
work to be completed : n_5
work to be completed : n_6