



University of Southern California  
Center for Software Engineering

---

# **CS599 Software Process Modeling**

## **Week 2**

**Ray Madachy**

**September 7, 1999**



# Outline

- Boehm: *Using Process Models to Analyze Rapid Application Development*
- Software Process Modeling Overview  
(continued from last week)
- System Dynamics Fundamentals
- Brookes's Law Introduction
- Brookes's Law Model
  - overview
  - experimentation
  - homework



# Software Process Models

- Enable process experimentation via simulation without tying up valuable resources
  - tradeoff analyses and process optimization
- Models can be used to quantitatively evaluate the software process
  - demonstrate effects of process strategies on cost, schedule and quality throughout lifecycle
  - can experiment with changed processes before committing project resources
  - interactive training for software managers; “process flight simulation”
  - implement process re-engineering
  - continually calibrate models with latest data



# Software Process Models (cont.)

- Encapsulate our understanding of development processes (and support organizational learning).
- Benchmark process improvement since model parameters are calibrated to organizational data.



# System Dynamics Approach

- Involves following concepts [Richardson 91]
  - defining problems dynamically, in terms of graphs over time
  - striving for an endogenous, behavioral view of the significant dynamics of a system
  - thinking of all real systems concepts as continuous quantities interconnected in information feedback loops and circular causality
  - identifying independent levels in the system and their inflow and outflow rates
  - formulating a model capable of reproducing the dynamic problem of concern by itself
  - deriving understandings and applicable policy insights from the resulting model
  - implementing changes resulting from model-based understandings and insights.
- Dynamic behavior is a consequence of system structure 5



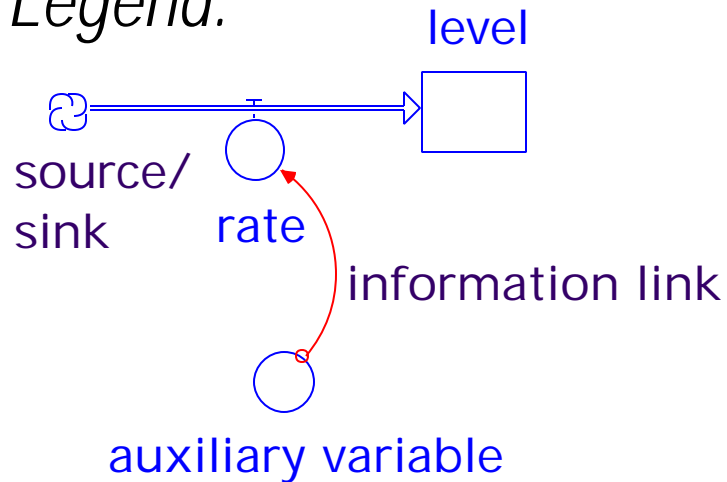
# Applicability to Software Processes

- Since software development is a dynamic and complex process with many factors, system dynamics is well-suited to analysis of software process improvement strategies
  - global system perspective
  - accounts for process feedback effects
  - can model inherent tradeoffs between schedule, cost and quality
  - accounts for critical path flows to analyze schedule as opposed to traditional cost reduction analyses
  - enables low cost process experimentation

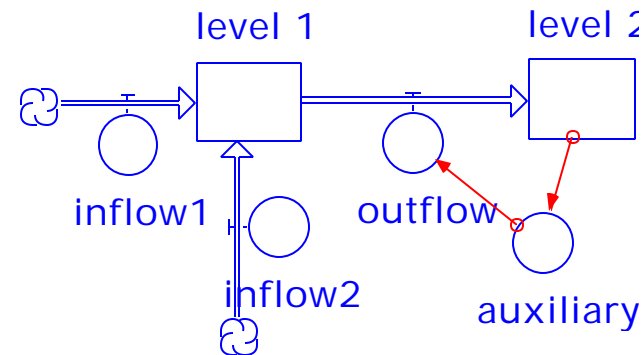
# System Dynamics Notation

- System represented by  $x'(t) = f(x, p)$ .
  - $x$ : vector of levels (state variables),  $p$ : set of parameters

- Legend:



- Example system:





# Model Components

- Level
  - An accumulation over time, also called stock or state variable. A storage device for material, energy, information.
  - Snapshot test: stop time and freeze flows in actual system. Level variables are those that still exist and have meaning in snapshot; the accumulations can be measured.
  - Software process level instances:
    - Work artifacts (requirements, tasks, lines of code, documentation pages)
    - Defect levels
    - Personnel levels
    - Effort expenditure
    - Schedule date
    - Others



# Model Components (continued)

- Rates

- flows; the “actions” in a system
- inseparable from levels
- effect the changes in levels

- Sources and sinks

- their presence indicates that the real-world accumulations occur outside boundary of the system being modeled
- represent infinite supplies or repositories

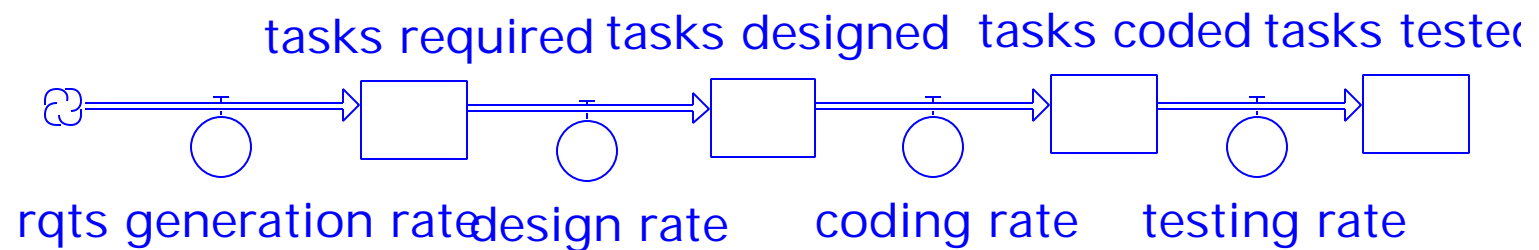
- Auxiliaries

- converters of input to output
- they elaborate detail of stock and flow structure
- often represent “score-keeping” variables

- Connectors

- information linkages

# Software Product Transformations

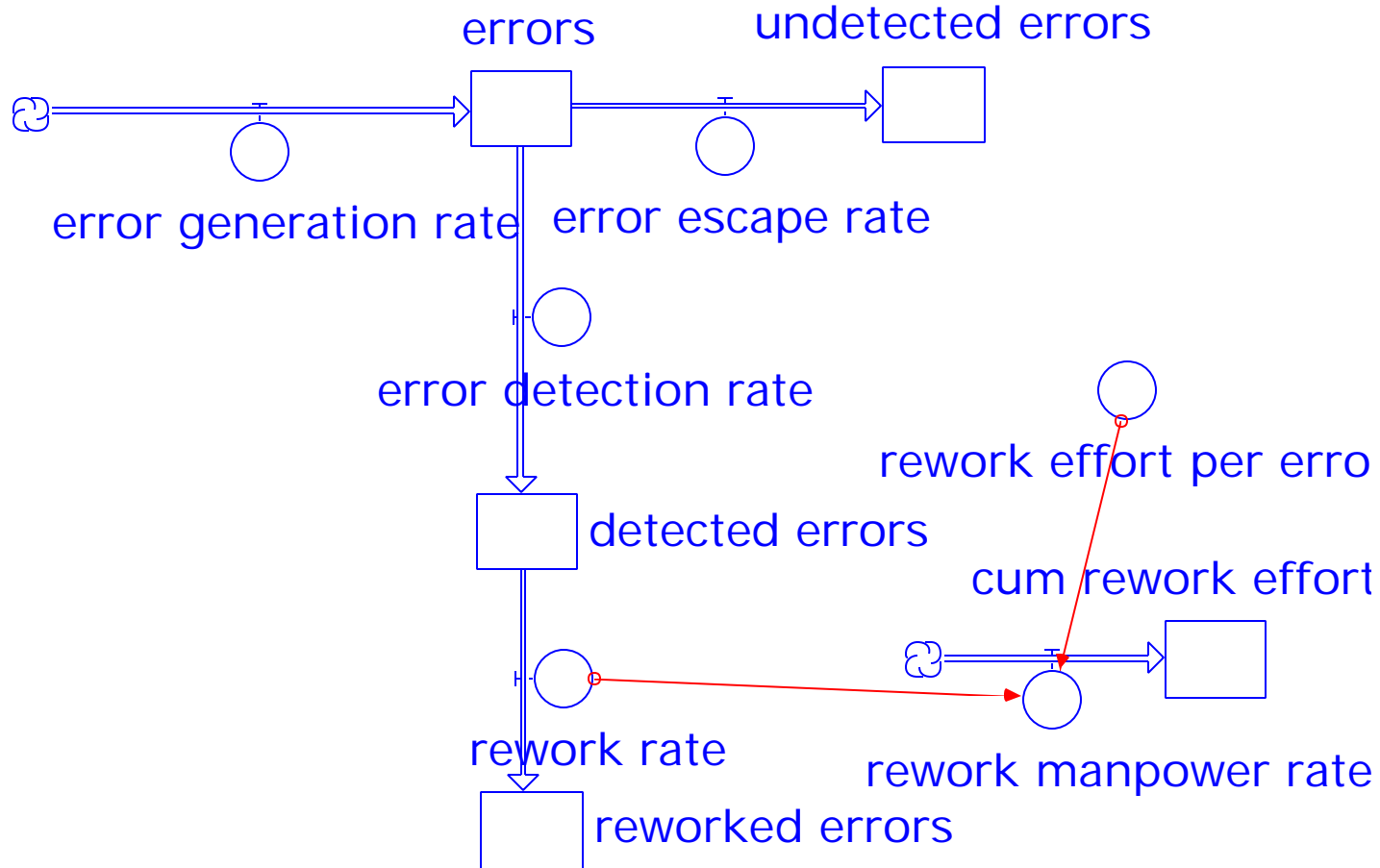


*Cycle time per phase =  
start time of first flowed entity - completion time of last flowed entity*

*Cycle time per task = transit time through relevant phase(s)*

# Cost/Schedule/Quality Tradeoffs

- Inherent in system dynamics models that represent defects as levels, and include the associated variable effort and cycle time for rework and testing as a function of those levels



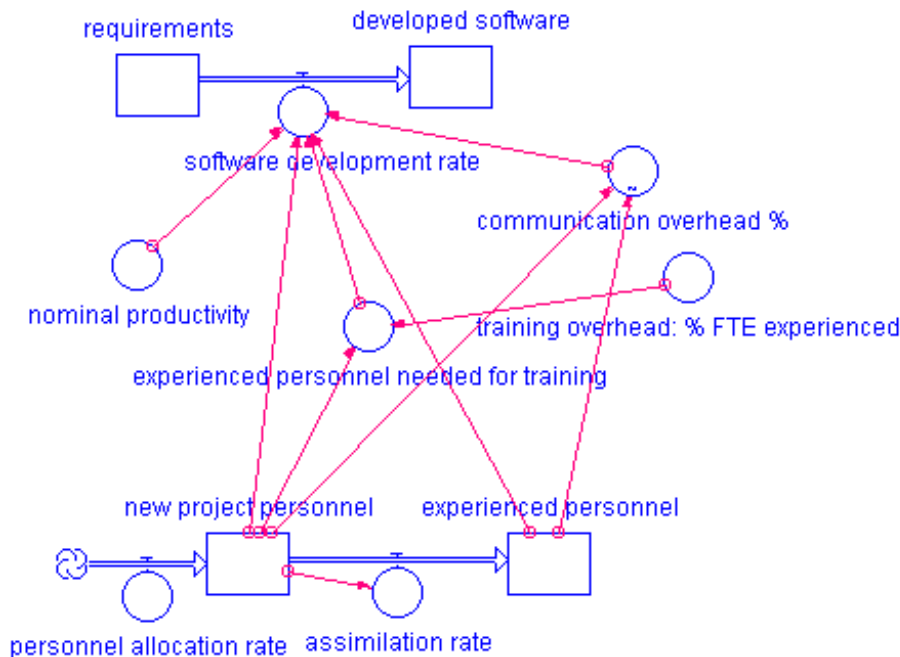


# Brooks' Law Modeling Example

- “Adding manpower to a late software project makes it later” [Brooks 75].
- We will test the law using a simple model based on the following assumptions:
  - new personnel require training by experienced personnel to come up to speed
  - more people on a project entail more communication overhead
  - experienced personnel are more productive than new personnel, on average.



# Model Diagram and Equations



developed\_software(t) = developed\_software(t - dt) + (software\_development\_rate) \* dt  
INIT developed\_software = 0

DOCUMENT: This level represents software function points that have been implemented.  
INFLOWS:

software\_development\_rate = nominal\_productivity\*(1-communication\_overhead\_/100.)\*(8\*new\_project\_personnel+1.2\*(experienced\_personnel-experienced\_personnel\_needed\_for\_training))  
DOCUMENT: The development rate represents productivity adjusted for communication overhead, weighting factors for the varying mix of personnel, and the effective number of experienced personnel.

experienced\_personnel(t) = experienced\_personnel(t - dt) + (assimilation\_rate) \* dt  
INIT experienced\_personnel = 20

DOCUMENT: The number of experienced personnel.

INFLOWS:

assimilation\_rate = new\_project\_personnel/20  
DOCUMENT: The average assimilation time for new personnel is 20 days.

new\_project\_personnel(t) = new\_project\_personnel(t - dt) + (personnel\_allocation\_rate - assimilation\_rate) \* dt  
INIT new\_project\_personnel = 0

DOCUMENT: The number of new project personnel.

INFLOWS:

personnel\_allocation\_rate = pulse(10,100,999)

OUTFLOWS:

assimilation\_rate = new\_project\_personnel/20

DOCUMENT: The average assimilation time for new personnel is 20 days.

requirements(t) = requirements(t - dt) + (- software\_development\_rate) \* dt  
INIT requirements = 500

DOCUMENT: The project size is 500 function points. This level represents the number left to be implemented.

OUTFLOWS:

software\_development\_rate = nominal\_productivity\*(1-communication\_overhead\_/100.)\*(8\*new\_project\_personnel+1.2\*(experienced\_personnel-experienced\_personnel\_needed\_for\_training))  
DOCUMENT: The development rate represents productivity adjusted for communication overhead, weighting factors for the varying mix of personnel, and the effective number of experienced personnel.

experienced\_personnel\_needed\_for\_training = new\_project\_personnel\*training\_overhead:\_%\_FTE\_experienced/100

DOCUMENT: Training overhead is the effort expended by experienced personnel to bring new people up to speed. It is the number new personnel \* the percent of an experienced person's time dedicated to training.

nominal\_productivity = .1

DOCUMENT: The nominal (unadjusted) productivity is .1 function points/person-day.

total\_personnel = experienced\_personnel+new\_project\_personnel

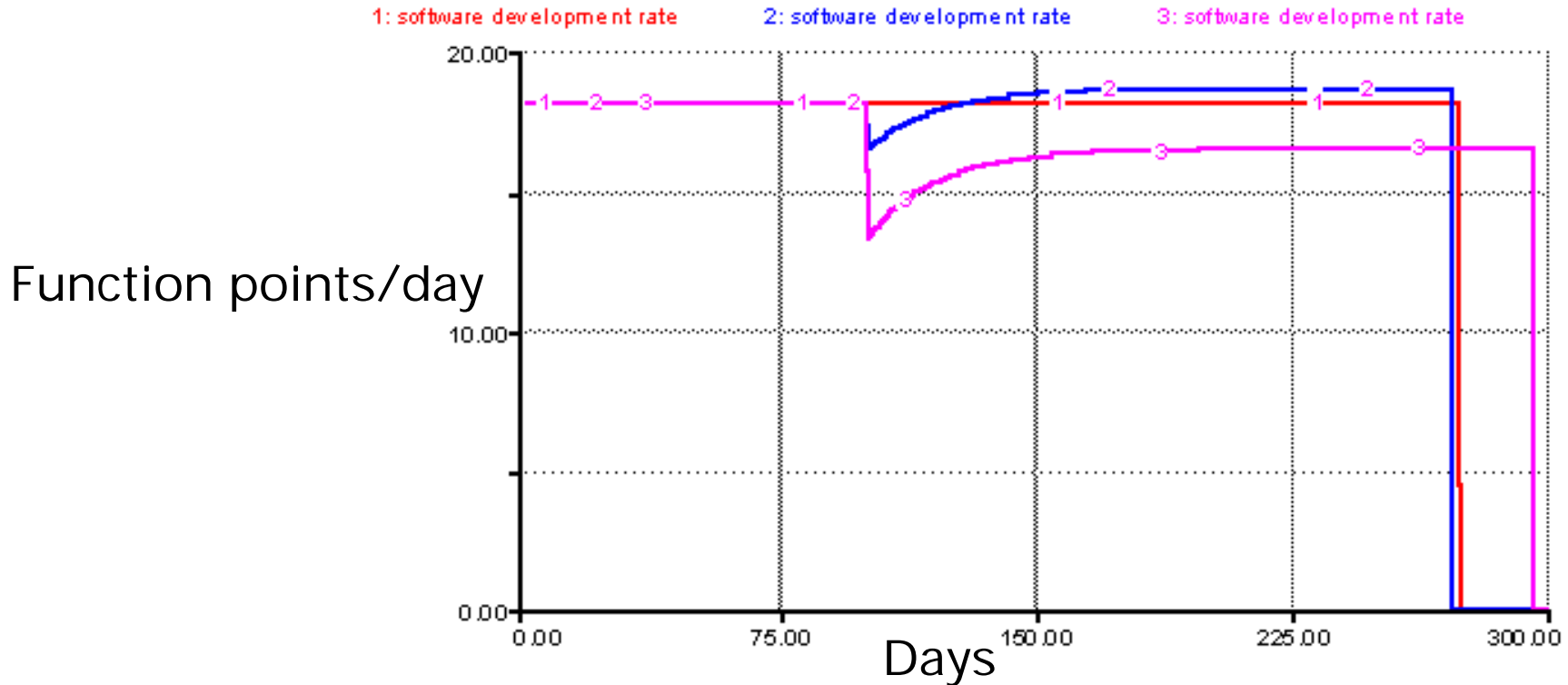
training\_overhead:\_%\_FTE\_experienced = 25

DOCUMENT: Percent of full-time equivalent experienced person's time dedicated to training new hires.

communication\_overhead\_% = GRAPH((experienced\_personnel+new\_project\_personnel))  
(0.00, 0.00), (5.00, 1.50), (10.0, 6.00), (15.0, 13.5), (20.0, 24.0), (25.0, 37.5), (30.0, 54.0)

DOCUMENT: Percent of time spent communicating with other team members as a function of team size. This graph represents the n<sup>2</sup> law in this size region, and was used in the Abdel-Hamid model.

# Model Output for Varying Additions



## Sensitivity of Software Development Rate to Varying Personnel Allocation Pulses

(1: no extra hiring, 2: add 5 people on 100th day, 3: add 10 people on 100th day)



# Brooks's Law Homework

- Preliminary reading for homework problem:
  - Brooks: *The Mythical Man-Month*, pp. 16-26, 231-232, 273-275
  - *Software Process Dynamics*, Section 1.4 Brooks's Law Example
  - Briand et al.: *Explaining the Cost of European Space and Military Projects* (focus on team size effects only)
  - Conte et al.: *Software Engineering Metrics and Models*, Section 5.8 (team size data and partitioning modeling)



# Brooks's Law Homework (cont.)

- Problem due in two weeks:
  - Use the existing Brooks's Law model as a basis or create your own similar version for the homework enhancements
  - Part 1: add a pause to the simulation when all requirements are developed
    - this will correct the model from running overtime
  - Part 2: make the model scalable for larger team sizes up to 60 people
    - make several runs to test the model and show your results
  - Part 3: add a simple feedback loop that controls personnel allocation rate by comparing actual production to planned production
    - the existing model covers actual production
    - the planned production assumes a constant development rate, with all 500 function points completed at 200 days
    - add logic for a one-time only correction when the difference between actual and planned is 65 function points
    - run the model and show the results for adding 0, 5, 10 and 20 people



# Brooks's Law Homework (cont.)

- Part 4: add the effects of partitioning to the resulting model in part 3
  - you may use the handout data from Conte et al. and Briand et al. to help develop and/or test your model
  - make several runs to test the model and show your results
  - now what is the optimal addition of people?
  - your model is now the world's best illustration of Brooks's Law
- **Fully document your model enhancements, your validation results, and any lessons learned about modeling and/or software process dynamics**



# Other Homework

- Start brainstorming about possible term projects
  - discuss with professors/employers etc.
- Next week we will discuss more example projects