Cost/Schedule/Process Modeling via System Dynamics

Ray Madachy
Litton Guidance and Control Systems
USC Center for Software Engineering

14th International Forum on COCOMO and Software Cost Modeling
October 26, 1999

Outline

- Introduction to Process Modeling and System Dynamics
- *Brooks's Law Demonstration*
- Model Structures and System Behaviors
- Overview of Past Applications
- *Earned Value Demonstration*
- Rapid Application Development (RAD) Modeling and Process Concurrence
- *Rayleigh Curve and Dynamic COCOMO Demonstration*
Terminology

- **System**: a grouping of parts that operate together for a common purpose; a subset of reality that is a focus of analysis
  - open, closed
- **Software Process**: a set of activities, methods, practices and transformations used by people to develop software.
- **Model**: an abstract representation of reality.
  - static, dynamic, continuous, discrete
- **Simulation**: the numerical evaluation of a mathematical model.
- **System dynamics**: a simulation methodology for modeling continuous systems. Quantities are expressed as levels, rates and information links representing feedback loops.

A Software Process
Software Process Models

- Used to quantitatively evaluate the software process.
- Demonstrate effects of process strategies on cost, schedule and quality throughout lifecycle.
- Enable tradeoff analyses and process optimization.
- Can experiment with changed processes via simulation before committing project resources.
- Provide interactive training for software managers; "process flight simulation".
- Encapsulate our understanding of development processes (and support organizational learning).
- Benchmark process improvement when model parameters are calibrated to organizational data.
- Process modeling techniques can be used to evaluate other existing descriptive theories/models.
  - force clarifications, reveal discrepancies, unify fields

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
Systems Thinking

- A way to realize the structure of a system that leads to its behavior
- Systems thinking involves:
  - thinking in circles and considering interdependencies
  - closed-loop causality vs. straight-line thinking
  - seeing the system as a cause rather than effect
  - internal vs. external orientation
  - thinking dynamically rather than statically
  - operational vs. correlational orientation
- Improvement through organizational learning takes place via shared mental models
- The power of models increase as they become more explicit and commonly understood by people
  - a context for interpreting and acting on data
- System dynamics is a methodology to implement systems thinking and leverage learning efforts

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
Process Modeling Characterization
Matrix and Examples

<table>
<thead>
<tr>
<th>Script/Purpose</th>
<th>Planning</th>
<th>Development</th>
<th>Multiple, concurrent projects</th>
<th>Long-term product evolution</th>
<th>Long-term organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategic management</td>
<td>stage-based, non-schedule driven</td>
<td>project, overall, strategy, quality, optimization</td>
<td>product-line, revenue strategy, real costs</td>
<td>projected, handoff, business growth</td>
<td></td>
</tr>
<tr>
<td>Planning</td>
<td>stage-based, non-schedule driven</td>
<td>project, overall, strategy, quality, optimization</td>
<td>product-line, revenue strategy, real costs</td>
<td>projected, handoff, business growth</td>
<td></td>
</tr>
<tr>
<td>Control and operational management</td>
<td>stage tracking, control</td>
<td>overall, value, tracking, change, control</td>
<td>product-line, revenue strategy, real costs</td>
<td>projected, handoff, business growth</td>
<td></td>
</tr>
<tr>
<td>Process improvement and technology adoption</td>
<td>RAD process, tradeoffs</td>
<td>project, review, processes, change, control</td>
<td>project, review, processes, change, control</td>
<td>product-line, revenue strategy, real costs</td>
<td></td>
</tr>
<tr>
<td>Understanding</td>
<td>process, concurrence, network, level</td>
<td>resource sharing, tradeoffs, cycle times</td>
<td>sustainable, size and effort, trends</td>
<td>organizational behavior</td>
<td></td>
</tr>
<tr>
<td>Training and learning</td>
<td>managerial, training</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Continuous View

- Individual events are not tracked
- Entities are treated as aggregate quantities that flow through a system
  - can be described through differential equations
- Discrete approaches usually lack feedback, internal dynamics
System Dynamics Notation

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

- Legend:
  - level
  - source/sink
  - rate
  - information link
  - auxiliary variable

- Example system:

Modeling Process Overview

- Iterative, cyclic
  - policy implementation
  - system understandings
  - policy analysis
  - problem definition
  - simulation
  - model conceptualization
  - model formulation
Modeling Stages and Concerns

- problem definition
  - context, symptoms
  - reference behavior modes
- model conceptualization
  - model purpose
  - system boundary
- model formulation
  - feedback structure
  - model representation
- simulation
  - model behavior
- evaluation
  - reference behavior modes

Modeling Tools

- DYNAMO
  - Fortran-like programming language
  - modeler writes difference equations
- ITHINK and Stella
  - visual programming, levels of abstraction
  - some utilities for discrete components
- Powersim
  - graphical interface
  - web-enabled simulations
- Vensim
  - comprehensive package with graphical interface
  - statistical estimation and model calibration facilities
- Extend
  - iconic environment supports continuous, discrete-event and mixed mode simulation
  - extensible with source code access
- Others
Outline

• Introduction to Process Modeling and System Dynamics
  * Brooks's Law Demonstration
  * Model Structures and System Behaviors
  * Overview of Past Applications
  * Earned Value Demonstration
  * Rapid Application Development (RAD) Modeling and Process Concurrence
  * Rayleigh Curve and Dynamic COCOMO Demonstration

Brooks's 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 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)
Outline

- Introduction to Process Modeling and System Dynamics
- *Brooks’s Law Demonstration*
- Model Structures and System Behaviors
  - Overview of Past Applications
  - *Earned Value Demonstration*
  - Rapid Application Development (RAD) Modeling and Process Concurrence
  - *Rayleigh Curve and Dynamic COCOMO Demonstration*

Model Structure and Behavior

- Description of levels, flows, feedback loops
- Model building blocks
- Basic flow processes and infrastructures
- Summary of general system behaviors
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
    • Revenue
    • Schedule date
    • Others

Model Components (continued)

• Rates
  – flows; the “actions” in a system that often represent policies
  – inseparable from levels
  – effect the changes in levels

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

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

• Connectors
  – information linkages
Feedback Loops

- A feedback loop is a closed path connecting an action decision that affects a level, then information on the level being returned to the decision making point to act on.

![Diagram of a feedback loop](image)

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)
Error Co-flows

- tasks designed
- design rate
- design errors
- design error generation rate
- design error density

Error Detection and Rework

- errors
- undetected errors
- error generation rate
- error escape rate
- error detection rate
- detected errors
- rework rate
- reworked errors
Personnel Pool

newly hired workforce  experienced workforce

hiring rate  workforce assimilation rate  quit rate

Learning Curve

tasks completed

development rate  job size

productivity  percentage complete

manpower rate  learning
Software Production Structure

- Combines task development and personnel chains.
- Production constrained by productivity and applied personnel resources.

![Diagram of software production structure]

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.

![Diagram of cost/schedule/quality tradeoffs]
General System Behaviors

- Behaviors are representative of many known types of systems.
- Knowing how systems respond to given inputs is valuable intuition for the modeler.
- Can be used during model assessment
  - use test inputs to stimulate the system behavioral modes

System Order

- The order of a system refers to the number of levels contained.
- A single level system cannot oscillate, but a system with at least two levels can oscillate because one part of the system can be in disequilibrium.
Example System Behaviors

- Delays
- Goal-seeking Negative Feedback
  - First-order Negative Feedback
  - Second-order Negative Feedback
- Positive Feedback Growth or Decline
- S-curves

Delays

- Time delays are ubiquitous in processes
- They are important structural components of feedback systems.
- Example: hiring delays in software development.
  - the average hiring delay represents the time that a personnel requisition remains open before a new hire comes on board

```
import pandas as pd
import numpy as np

data = {'process': ['personnel requisition', 'new hire'], 'time_delay': [5, 10]}

df = pd.DataFrame(data)
```

33

34
Third Order Delay

- A series of 1st order delays
- Graphs show water levels over time in each tank

Tank 1 starts full

Delay Summary

<table>
<thead>
<tr>
<th>Delay order</th>
<th>Pulse input</th>
<th>Step input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1" alt="Pulse Input 1" /></td>
<td><img src="image2" alt="Step Input 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image3" alt="Pulse Input 2" /></td>
<td><img src="image4" alt="Step Input 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image5" alt="Pulse Input 3" /></td>
<td><img src="image6" alt="Step Input 3" /></td>
</tr>
<tr>
<td>Infinite</td>
<td><img src="image7" alt="Pulse Input Infinite" /></td>
<td><img src="image8" alt="Step Input Infinite" /></td>
</tr>
</tbody>
</table>

(pipeline)
Negative Feedback

- Negative feedback exhibits goal seeking behavior, or sometimes instability.
- May represent hiring increase towards a staffing goal. The change is more rapid at first and slows down as the discrepancy between desired and perceived decreases. Also a good trend for residual defect levels.

\[ \text{rate} = \frac{(\text{goal} - \text{present level})}{\text{time constant}} \]

Analytically:
\[ \text{Level} = \text{Goal} - (\text{Level} - \text{Goal})e^{-\frac{t}{\text{time constant}}} \]

Orders of Negative Feedback

- **First-order Negative Feedback**
- **Second-order Negative Feedback**
  - Oscillating behavior may start out with exponential growth and level out. It could represent the early sales growth of a software product that stagnates due to satisfied market demand, competition or declining product quality.
Positive Feedback

- Positive feedback produces a growth process
- Exponential growth may represent sales growth (up to a point), Internet traffic, defect fixing costs over time
- rate = present level*constant

Analytically:
- exponential growth: Level = Level_0 e^{at}
- exponential decay: Level = Level_0 e^{-rt}

S-Curves

- S-curve: graphic display of a quantity like progress or cumulative effort plotted against time that exhibits an s-shaped curve. It is flatter at the beginning and end, and steeper in the middle. It is produced on a project that starts slowly, accelerates and then tails off as work tapers off
- S-curves are also observed in the ROI curve of technology adoption, either time-based return or in production functions that relate ROI to investment.
Outline

- Introduction to Process Modeling and System Dynamics
- Brooks’s Law Demonstration
- Model Structures and System Behaviors
- Overview of Past Applications
- Earned Value Demonstration
- Rapid Application Development (RAD) Modeling and Process Concurrence
- Rayleigh Curve and Dynamic COCOMO Demonstration

Brief History

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1961</td>
<td>Jay Forrester publishes Industrial Dynamics</td>
</tr>
<tr>
<td>1984</td>
<td>Tarek Abdel-Hamid completes Ph.D. dissertation at MIT</td>
</tr>
<tr>
<td>late 1980's</td>
<td>NASA JPL and a few others begin research with system dynamics</td>
</tr>
<tr>
<td>1991</td>
<td>Abdel-Hamid and Stuart Madnick publish Software Project Dynamics</td>
</tr>
<tr>
<td>1991-1999</td>
<td>Many industrial and academic implementations, including the effects of process improvement initiatives</td>
</tr>
</tbody>
</table>
Model Implementations

- **Industry/government**: AT&T, Bellcore, Draper Labs, Fedex, Hughes, Litton, Mitre, NASA, Siemens, others
- **Academic**: ASU, Imperial College, Stanford, MIT, Naval Postgraduate School, USC, others
- **Tool vendors/workshops**: Bartz Associates, Dynamica, Rubin Systems
- Many other companies are evaluating system dynamics for process improvement
- Several academic research projects in proposal stage or dissertations being written

---

Process Evaluation

- Investigating the dynamic effects of inspections [Madachy 94], [Tvedt 95]
- Incremental development [Tvedt 95]
- Unit testing phase [Collafello et al. 96]
- Requirements phase (several)
- Investigating software reuse from a macro-inventory perspective [Abdel-Hamid 93a]
- Software outsourcing [Collafello et al. 99]
- Process model tradeoffs
Process Evaluation (continued)

- Organizational CMM-based process improvement (Burke 96)
- Other process improvement investments
  - staffing policies
  - work environment investments
  - computer aided tool investments
  - staff training investments
  - metrics, reuse, risk management and others
- Global software process feedback, stability and product evolution [Lehman 98]

Flight Simulators

- Personnel training
  - graduate software project management (ASU)
  - vendor tools (Rubin et al.)
- Navigating new skies
  - process maturity initiatives
- Stimulate dialogues for shared mental models
- Virtual reality for court cases
Other Applications

- Integration with cost estimation models
  - improving on static assumptions [Madachy 95], [Rubin et al. 95]
  - calibrations between [Madachy 95]
  - deriving static parameters with dynamic experiments [Madachy 95]

- Knowledge-based assistance/expert systems
  - heuristic project risk analysis and input checking [Madachy 94]
  - input evaluation and change recommendation [Lin et al. 92]
  - QA expert simulator

- Examining heuristics
  - Brookes's Law (several)
  - cost estimation correction processes [Abdel-Hamid 93]
  - others

Sample Insights

- Inspection policy tradeoff analysis - diminishing returns from inspections as a function of error generation rates [Madachy 94]
- QA policy tradeoff analysis - finding the optimal QA effort [Abdel-Hamid/Madnick 91]
- Rework staffing allocation [Tvedt 95]
- Organizational process improvement transition requires temporary productivity setbacks [Rubin, Johnson, Yourdon 95]
- Maximize your pro-SPI people (Burke 96)
Sample Insights (continued)

- Leverage of experienced staff (several)
- Internal workings of Brookes’s Law - training and communication losses
- Schedule compression not a static decision [Abdel-Hamid 90]
- Anchor-dragging in project control [Abdel-Hamid 93]
- Competing feedback loops in software reuse factory [Abdel-Hamid 93b]
- Many others

Example Product Chains

Abdel-Hamid/Madnick 91

Madachy 94

Tvedt 95
Example Defect Chains
Abdel-Hamid/Madnick 91

Example Personnel Chains
Madachy 94
Abdel-Hamid/Madnick 91
Abdel-Hamid Model Subsystems

Abdel-Hamid Model Behavior

- Underestimation factor = 0.67
Abdel-Hamid Model Critique

Advantages
- The model includes a good deal of important dynamic effects. It does well to illustrate some methods of poor software management (this can also be a downfall if certain policies are emulated, or the model becomes prescriptive instead of descriptive).
- It uses the very realistic notion that management perceptions, rather than true conditions, dictate actions.
- Delays in action are also important realistic considerations that the model covers well.
- The inclusion of important personnel attributes like motivation, exhaustion and schedule pressure effects.
- The model is strong in terms of planning and control structures.

Downfalls
- The model contains too many elements for managers to understand and use, and requires initialization of many parameters and functions.
- The software production model is too simplistic for many purposes. Design and code are aggregated together.
- The model may be wrongly used or overly relied on to perpetuate poor management practices, such as not being able to status progress early on.
- The definition of QA is non-standard. The activities modeled as QA would be considered part of standard development activities in most installations, and performed by the same people who develop the software.
- The control function uses a misleading indicator of progress.
Introduction to Madachy Inspection Model

- Research problem addressed
  - What are the dynamic effects to the process of performing inspections?

- Model used to evaluate process quantitatively
  - Demonstrates effects of inspection practices on cost, schedule and quality throughout lifecycle
  - Can experiment with changed processes before committing project resources
  - Benchmark process improvement
  - Support project planning and management

- Model parameters calibrated to Litton data
  - Error generation rates, inspection effort, efficiency, COCOMO constant, others

- Model validated against industrial data

System Diagram
System Diagram (continued)

Effects of Inspections

1: with inspections, 2: without inspections

* Qualitatively matches generalized effort curves from Michael Fagan (Advances in software inspections, IEEE Transactions on Software Engineering, July 1986), which were used in problem definition.
Sample Project Progress Trends

- From [Madachy 94]

![Graph showing project progress trends.]

Inspection Policy Tradeoff Analysis

- Varying error generation rates shows diminishing returns from inspections [Madachy 94]:

![Graph showing inspection policy tradeoff analysis.]

[Graphs and data points illustrating trends and analysis.]

[References and citations included as appropriate.]
Error Multiplication Effects

![Error Multiplication Effects graph]

Derivation of Phase Specific Cost Driver

<table>
<thead>
<tr>
<th>Simulation Parameters</th>
<th>COCOMO Rating for Use of Inspections</th>
</tr>
</thead>
<tbody>
<tr>
<td>design inspection practice</td>
<td>code inspection practice</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

![Derivation of Phase Specific Cost Driver table and graph]

Nominal
High
Very High

Phase

32
Risk Analysis

- A deterministic point estimate from a simulation run is only one of many actual possibilities
- Simulation models are ideal for exploring risk
  - test the impact of input parameters
  - test the impact of different policies
- Monte-Carlo analysis takes random samples from an input probability distribution

Monte-Carlo Example

- Results of varying inspection efficiency:
Contributions of Madachy Inspection Model

- Demonstrated dynamic effects of performing inspections.
- New knowledge regarding interrelated factors of inspection effectiveness.
- Demonstrated complementary features of static and dynamic models.
- Techniques being adopted in industry.

CS599 Software Process Modeling Student Term Projects

- Dynamics of architecture development process in MBASE inception and elaboration phases
- Application of RAD techniques to pre-IPO internet companies
- COTS glue-code development and integration dynamics
- Reuse and language-level effects in software development
- CMM-based process improvement strategies
Outline

- Introduction to Process Modeling and System Dynamics
- Brooks's Law Demonstration
- Model Structures and System Behaviors
- Overview of Past Applications
- Earned Value Demonstration
- Rapid Application Development (RAD) Modeling and Process Concurrence
- Rayleigh Curve and Dynamic COCOMO Demonstration

Earned Value

- Earned value is a method for measuring project performance.
- It compares the amount of work that was planned with what was actually accomplished to determine if cost and schedule performance is as planned.
- Cost performance index (CPI) is the ratio of budgeted costs to actual costs (BCWP/ACWP)
- Schedule performance index (SPI) is the ratio of work performed to work scheduled (BCWP/BCWS)
Earned Value Model

Outline

- Introduction to Process Modeling and System Dynamics
- *Brooks' Law Demonstration*
- Model Structures and System Behaviors
- Overview of Past Applications
- *Earned Value Demonstration*
- Rapid Application Development (RAD) Modeling and Process Concurrence
- *Rayleigh Curve and Dynamic COCOMO Demonstration*
Context for RAD Parameter Hypothesis Testing

- Develop a model that isolates the effects of the factor of interest (e.g. RVHL, BPRS)
- Eliminate other confounding effects besides chosen parameter
- Model the underlying mechanics for the chosen parameter
- Instrument cost and schedule in the model
- Run planned experiments that vary the parameter of interest over the desired range
- Independently derive effort and schedule multipliers from the trials

RAD Modeling Example: Process Reengineering and Streamlining

- Run this model at different settings for number of approvals per task and time taken per approval
- Derive schedule multipliers from experiments
**Process Concurrence**

- Process concurrence refers to interdependency constraints between tasks, both within and between phases.
- Describes how much work becomes available for completion based on previous work accomplished.
- Bottlenecks on the availability of work
- Concurrence relations can be sequential, parallel, partially concurrent, or other dependent relationships.

**Trying to Accelerate Software Production**

[picture of a funnel diagram with software tasks, tasks to develop, and completed tasks]
Limited Parallelism of Software Activities

- There are always sequential constraints independent of phase:
  - analysis and specification
  - figure out what you're supposed to do
  - development of something (architecture, design, code, test plan, etc.)
  - assessment
  - verify/validate/review/debug
  - possible rework recycle of previous activities

- These can't be done totally in parallel with more applied people

Different people can perform the different activities with limited parallelism, but downstream activities will always have to follow some of the upstream.

Funnel View of Limited Parallelism
Lessons from Brooks in *The Mythical Man-Month*

- Sequential constraints imply tasks cannot be partitioned.
  - Applying more people has no effect on schedule
- Men and months are interchangeable only when tasks can be partitioned with no communication among them.

Internal Process Concurrence

- Internal process concurrence relationship shows how much work can be done based on the percent of work already done.
- The relationships represent the degree of sequentiality or concurrence of the tasks aggregated within a phase.
Linear and Non-linear Internal Process Concurrence

A lockstep relationship.

The overarching segments of software must be completed before other parts can begin.

Internal Concurrence Examples

Simple conversion task where tasks can be partitioned with no communication

Complex system development where tasks are dependent due to required inter-task communication.
External Process Concurrence

- External process concurrence relationships describe constraints on amount of work that can be done in a downstream phase based on the percent of work released by an upstream phase.

- See examples on following slides
  - More concurrent processes have curves near the upper left axes, and less concurrent processes have curves near the lower and right axes.

External Process Concurrence Example

- Typical non-linear external process concurrence relationship
No Inter-phase Relationship

- No dependencies between the phases.
- The downstream phase can progress independently of the upstream phase.
- The entire downstream work is available to be completed with none of the upstream work released.

Sequential Inter-phase Relationship

- None of the downstream phase can occur until the upstream phase is totally complete.
- Like a theoretical waterfall development process where no phase can start until the previous phase is completed and verified.
- Same as a finish-stop relationship in a critical path network.
Parallel Inter-phase Relationship

- The two phases can be implemented completely in parallel.
- The downstream phase can be completed as soon as the upstream phase is started.

Delayed Start Inter-phase Relationship

- The downstream phase must wait until a major portion of the upstream phase is completed, then it can be completed in its entirety.
- Like a start-start relationship in a critical path network.
Lockstep Inter-phase Relationship

- The downstream phase can progress at the same speed as the upstream phase; thus they are in lockstep with each other.
- Like stovepipe components stacked on top of each other, similar to building the stories of a skyscraper.
- This relationship is not available in PERT/CPM methods.

Delay with Partially Concurrent Inter-phase Relationship

- The downstream phase has to wait until a certain percentage of upstream tasks have been released, and then can proceed at varying degrees of concurrence per the graph.
- Representative of much software development work.
- This relationship is not available in PERT/CPM.
Roles Have Different Mental Models

- Differing perceptions upstream and downstream (Ford-Sterman 97)

Four Estimates of External Process Concurrence Relationship between the Product Definition and Design Phases

RAD Awareness

- **Hypothesis**: to optimize schedule on a complex project with partial inter-phase concurrency, the optimal systems engineering staffing is front-loaded vs. constant level-of-effort
  - downstream development is constrained by the specifications available

Case 1: not RAD aware

Case 2: RAD aware
External Concurrence Model

Diagram

specified tasks

development rate

percent of specs available to develop

specification rate

development personnel

productivity

deployed tasks

total tasks

specification constraint flag

External Concurrence Model

Equations

\[ \text{INIT} \text{developed_tasks} = 0 \]

\[ \text{INFLOWS:} \]

- development_rate \times \text{specification_rate} \times \text{specification_constraint_flag} = 0 \text{then development_personnel} \times \text{productivity} \times \text{specification_rate} \times \text{specification_constraint_flag} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks that have been developed} \]

\[ \text{OUTLETS:} \]

- development_rate \times \text{specified_tasks} \times \text{specification_constraint_flag} = 0 \text{then development_personnel} \times \text{productivity} \times \text{specified_tasks} \times \text{specification_constraint_flag} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks that have been specified but not developed} \]

- specification_rate \times \text{percent_of_specs_available_to_develop} \times \text{developed_tasks} \times \text{total_tasks} = 0 \text{then \text{development_personnel} \times \text{productivity} \times \text{percent_of_specs_available_to_develop} \times \text{developed_tasks} \times \text{total_tasks} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks to be developed} \]

- development_rate \times \text{total_tasks} = 0 \text{then development_personnel} \times \text{productivity} \times \text{total_tasks} = 0 \]

\[ \text{DOCUMENT:} \text{Total tasks to be developed} \]

- development_rate \times \text{total_tasks} = 0 \text{then development_personnel} \times \text{productivity} \times \text{total_tasks} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks to be specified and developed} \]

- development_rate \times \text{total_tasks} = 0 \text{then development_personnel} \times \text{productivity} \times \text{total_tasks} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks to be specified and developed} \]

- development_rate \times \text{total_tasks} = 0 \text{then development_personnel} \times \text{productivity} \times \text{total_tasks} = 0 \]

\[ \text{DOCUMENT:} \text{Tasks to be specified and developed} \]

- development_rate \times \text{total_tasks} = 0 \text{then development_personnel} \times \text{productivity} \times \text{total_tasks} = 0 \]
Rayleigh Manpower Distribution

- Rayleigh curve is a popular model of personnel loading
- Assumptions:
  - Only a small number of people are needed at the beginning of a project to carry out planning and specification. As the project progresses and more detailed work is required, the number of staff builds up to a peak. After implementation and unit testing is complete, the number of staff required starts to fall until the product is delivered.
  - The number of people working on a project is approximately proportional to the number of problems ready for solution at that time
Rayleigh Formula

A Rayleigh curve describes the rate of change of manpower effort per the following first order differential equation:

\[ \frac{dC(t)}{dt} = p(t)(K - C(t)) \]

where \( C(t) \) is the cumulative effort at time \( t \), \( K \) is the total effort, and \( p(t) \) is a learning function. The learning function is linear and can be represented by

\[ p(t) = at \]

where \( a \) is a positive number.

- The manpower rate of change represents the number of people involved in development at any time (staffing profile).
- The \( a \) parameter is an important determinant of the peak personnel loading called the manpower buildup parameter.

Rayleigh Model
Interactive Rayleigh Model Demo

- Vary manpower buildup parameter
- Demonstrate S-curve
- Show effect of midstream added requirements

References

- Abdel-Hamid T, Adapting, correcting, and perfecting software estimates: a maintenance metaphor, IEEE Computer, March 1993
- Abdel-Hamid T, Modeling the dynamics of software reuse: an integrating systems dynamics perspective, Presented at the Sixth Annual Workshop on Software Reuse, Chicago, NY, November 1991
- Burke S, Radical Improvements Require Radical Actions: Simulating a High-Maturity Organization, Software Engineering Institute, CMU/SEI-96-TR-024, June 1997
- Brooks F, The Mythical Man-Month, Reading, MA, Addison-Wesley, 1975
- Malekly R, A software project dynamics model for process cost, schedule and risk assessment, Ph.D. dissertation, Department of Industrial and Systems Engineering, USC, December 1994
References (cont.)

- Rubin H, Johnson M, Yourdon E, Software process flight simulation: dynamic modeling tools and metrics, Information Systems Management, Summer 1995
- Tvedt J, Collofello J, Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modeling, University of Arizona, 1995

USC-CSE Web Sites

- http://sunset.usc.edu/research_group/ray/spd
- http://sunset.usc.edu/classes/cs599_99
  - USC-CSE Software Process Modeling Course (include other system dynamics links)