CS599 Software Process Modeling
Week 3

Ray Madachy

September 14, 1999
Outline

• Software process lifecycles
• RAD hypothesis testing guidelines
• Example simulation project
• Some modeling heuristics
• Term projects
  – candidate problems
  – existing models for modification/testing
• Brookes’s Law homework review and discussion
• Homework
  – delay example
Process Lifecycle Models

- Waterfall
- Incremental
- Evolutionary
- Transform
- Reuse
- Spiral and WinWin Spiral
- MBASE/UDP
MBASE/UDP Software Process

Activities & Representative Amounts

Process Activities
- Requirements Capture
- Analysis & Design
- Implementation
- Test

Supporting Activities
- Management
- Environment
- Deployment

Time
- LCO
- LCA
- IOC

Stages
- Inception
- Elaboration
- Construction
- Transition

Iterations
- preliminary iteration(s)
- Iter. #1
- Iter. #2
- Iter. #n
- Iter. #n+1
- Iter. #n+2
- Iter. #m
- Iter. #m+1
Outline

• Software process lifecycles

• RAD hypothesis testing guidelines

• Example simulation project

• Some modeling heuristics

• Term projects
  – candidate problems
  – existing models for modification/testing

• Brookes’s Law homework review and discussion

• Homework
  – delay example
Guidelines for CORADMO

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
  – keep static parameters constant between simulation runs
  – use only a single pattern for other dynamic effects during the trials
  – only your chosen parameter can vary between runs
Guidelines for CORADMO
Parameter Hypothesis Testing (cont.)

• Model the underlying mechanics for the chosen parameter (e.g. the relevant levels and flow rates impacted by the parameter)
  – don’t simulate the behavior like COCOMO
  – you may resort to behavior simulation of other aspects of your model, as appropriate

• Instrument cost and schedule in your model

• Quantify the COCOMO rating guidelines into model input parameters
  – these will be used for experimental input
Guidelines for CORADMO

Parameter Hypothesis Testing (cont.)

• Make a best attempt to collect relevant data from literature for calibration and/or validation
  – do not use the COCOMO multipliers, since you are independently deriving them

• Interview process experts as appropriate for more insight
  – your professors can help identify experts

• Run planned experiments that vary the parameter of interest over the rating range
Guidelines for CORADMO

Parameter Hypothesis Testing (cont.)

• Independently derive effort and schedule multipliers from your trials
• Compare to CORADMO multipliers
• Write up your analysis
• It is possible that your analysis will have a change impact on CORADMO
  – parameter may appear irrelevant
  – multipliers might change
  – new parameters might be identified

See following slide for an example
## 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></td>
<td></td>
</tr>
<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>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Effort Multiplier

- **Nominal**: Black triangle (▲)
- **High**: Black square (●)
- **Very High**: Black diamond (♦)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Effort Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rqts. and Product Design</td>
<td>1.0</td>
</tr>
<tr>
<td>Detailed Design</td>
<td>1.1</td>
</tr>
<tr>
<td>Code and Unit Test</td>
<td>1.0</td>
</tr>
<tr>
<td>Integration and Test</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Outline

- Software process lifecycles
- RAD hypothesis testing guidelines
- Example simulation project
- Some modeling heuristics
- Term projects
  - candidate problems
  - existing models for modification/testing
- Brookes’ Law homework review and discussion
- Homework

cs599 9/14/99  delay example
Introduction to Madachy Inspection Model

• Research problem
  – What are the dynamic effects to the process of performing inspections?

• Abstract of work
  – A dynamic model of an inspection-based software lifecycle process has been developed to support quantitative evaluation of the process. In conjunction with a knowledge-based method for cost estimation and project risk assessment, these modeling techniques can support project planning and management, and aid in process improvement.
Process Model Summary

- 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

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

- Model validated against industrial data
System Diagram
Demonstration and Evaluation

• Effects of inspections
• Error generation
• Error multiplication
• Schedule compression
• Monte-Carlo analysis
• Learning curve
• Derivation of phase-specific cost driver
• Validation against industrial data
## Effects of Inspections

<table>
<thead>
<tr>
<th>Test Case Description</th>
<th>Effort (person-days)</th>
<th>Schedule (days)</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design</td>
<td>Coding</td>
<td>Inspection</td>
</tr>
<tr>
<td>1.1 COCOMO constant job size</td>
<td>3.6</td>
<td>32 KSLOC</td>
<td>1 simulation</td>
</tr>
<tr>
<td>1.2</td>
<td>0.5 simulation</td>
<td>2092</td>
<td>1224</td>
</tr>
<tr>
<td>1.3</td>
<td>0 simulation</td>
<td>2092</td>
<td>1224</td>
</tr>
<tr>
<td>2.1 COCOMO</td>
<td>3.6</td>
<td>64 KSLOC</td>
<td>1 simulation</td>
</tr>
<tr>
<td>2.2</td>
<td>0.5 simulation</td>
<td>4807</td>
<td>2812</td>
</tr>
<tr>
<td>2.3</td>
<td>0 simulation</td>
<td>4807</td>
<td>2812</td>
</tr>
<tr>
<td>3.1 COCOMO</td>
<td>7.2</td>
<td>32 KSLOC</td>
<td>1 simulation</td>
</tr>
<tr>
<td>3.2</td>
<td>0.5 simulation</td>
<td>4184</td>
<td>2448</td>
</tr>
<tr>
<td>3.3</td>
<td>0 simulation</td>
<td>4184</td>
<td>2448</td>
</tr>
<tr>
<td>4.1 COCOMO</td>
<td>7.2</td>
<td>64 KSLOC</td>
<td>1 simulation</td>
</tr>
<tr>
<td>4.2</td>
<td>0.5 simulation</td>
<td>9614</td>
<td>5625</td>
</tr>
<tr>
<td>4.3</td>
<td>0 simulation</td>
<td>9614</td>
<td>5625</td>
</tr>
</tbody>
</table>
Effects of Inspections (continued)

1: with inspections, 2: without inspections
Error Generation Rate Effects

![Graph showing the effect of error generation rate on total effort with and without inspections.](image)

- **Y-axis**: Total Effort (Person-days)
- **X-axis**: Defects/KSLOC
- **Legend**:
  - Blue diamonds: with inspections
  - Black squares: without inspections

The graph illustrates that the total effort required to handle defects increases as the defect generation rate increases. The effort is significantly higher without inspections compared to with inspections.
Error Multiplication Effects

![Graph showing the comparison of project effort with and without inspections across different design error multiplication values. The graph plots project effort on the y-axis and design error multiplication on the x-axis. Two lines are shown: one for with inspections and another for without inspections. The graph illustrates a linear increase in project effort as design error multiplication increases.](image-url)
Monte-Carlo Analysis

- Results of varying inspection efficiency:

![Effort Bin Frequency Chart](chart.png)
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>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase</th>
<th>Effort Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rqts. and Product Design</td>
<td>1</td>
</tr>
<tr>
<td>Detailed Design</td>
<td>1</td>
</tr>
<tr>
<td>Code and Unit Test</td>
<td>1</td>
</tr>
<tr>
<td>Integration and Test</td>
<td>1</td>
</tr>
</tbody>
</table>

Nominal  
High  
Very High
Project Data Validation

**Inspection return**

<table>
<thead>
<tr>
<th>Project/Test Case</th>
<th>Return</th>
<th>Inspection Effort</th>
<th>Benefit/Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Litton Project A</td>
<td>20206 person-hours saved in test using the average cost to fix an error during test</td>
<td>8716 person-hours</td>
<td>2.32</td>
</tr>
<tr>
<td>Test case 1.1</td>
<td>613 person-days saved in test compared to test case 1.3</td>
<td>303 person-days</td>
<td>2.02</td>
</tr>
</tbody>
</table>

**Test effort and schedule**

<table>
<thead>
<tr>
<th>Project/Test Case</th>
<th>Test Effort Reduction</th>
<th>Test Schedule Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Litton Project A compared to previous project</td>
<td>50%</td>
<td>25%</td>
</tr>
<tr>
<td>Test case 1.1 with Litton productivity constant and job size compared to test case 1.3 with Litton parameters</td>
<td>48%</td>
<td>19%</td>
</tr>
<tr>
<td>Test case 1.1 compared to test case 1.3</td>
<td>48%</td>
<td>21%</td>
</tr>
</tbody>
</table>

**Rework fraction of effort**

<table>
<thead>
<tr>
<th>Project/Test Case</th>
<th>Rework Effort</th>
<th>Preparation and Meeting Effort</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Litton Project A</td>
<td>2789 person-hours</td>
<td>5927 person-hours</td>
<td>.47</td>
</tr>
<tr>
<td>JPL (several projects)</td>
<td>.5 person-hours per defect</td>
<td>1.1 person-hours per defect</td>
<td>.45</td>
</tr>
<tr>
<td>Test case 1.1</td>
<td>100 person-days</td>
<td>203 person-days</td>
<td>.49</td>
</tr>
</tbody>
</table>
Contributions

• 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.
Outline

• Software process lifecycles
• RAD hypothesis testing guidelines
• Example simulation project

• Some modeling heuristics

• Term projects
  – candidate problems
  – existing models for modification/testing

• Brookes’s Law homework review and discussion

• Homework
  – delay example
Model Conceptualization Heuristics

• Define a clear, operational purpose of the model.
• Don’t try to model the “system”.
• Aggregate and abstract to the appropriate degree.
• Use a top-down iterative approach.
• KISS (keep it simple, stupid)
Model Formulation and Development Heuristics

- Don't enumerate all factors at first.
- Iteratively refine and slowly add relationships to model.
- Normalize when possible.
- Use relative measures.
- Don't stray too far from a simulatable model.
- Don't model in isolation; try to involve those being modeled.
Model Validation Heuristics

- Look for qualitative similarity on the first pass.
- Alter one parameter at a time at first.
- Be conscious of reality constraints.
- Model validity is a relative matter.
  - The usefulness of a mathematical simulation model should be judged in comparison with the mental image or other abstract model which would be used instead [Forrester 68]. Models are successful if they clarify our knowledge and insights into systems.
Miscellaneous Heuristics

Data Collection

• Model design should not be postponed until all pertinent parameters have been accurately measured.

Communication

• Use simple diagrams to communicate with others until they seek more detail.
General Modeling Heuristics

• No model is perfect.
  – but some are useful
• All models are incomplete.
• No model is final; it is possible to build many different models of a single process.
• All models contain hidden assumptions.
• Continually challenge the model.
Outline

• Software process lifecycles
• RAD hypothesis testing guidelines
• Example simulation project
• Some modeling heuristics

• Term projects
  – candidate problems
  – existing models for modification/testing

• Brookes’s Law homework review and discussion

• Homework
  – delay example
Candidate Term Projects

• RAD hypothesis testing per Boehm lecture (model the underlying process mechanics; don’t just simulate the behavior like high-level COCOMO):
  – speed of concurrent engineering vs. waterfall process
  – square root effort-schedule relationship for small projects
  – RVHL, BPRS, CLAB, RESL, PPOS cost and schedule effects
Candidate Term Projects (cont.)

• COQUALMO pipe and tank model of software defect levels
  – experiment with different project types, and validate the defect patterns
  – many other possibilities for defect modeling

• Model MBASE/UDP process
  – optionally focus on 577ab development process

• Adapt Burke process improvement model

  “Adapt”: modify, calibrate and test with local or other available data
Candidate Term Projects (cont.)

• Model collaboration effects between stakeholders, such as when using WinWin
  – See Christie et al. for an example approximation of collaboration effects. Simplify or otherwise adapt this model, or develop your own

• Adapt the iterative Ford-Sterman model with process concurrency and calibrate it for nominal MBASE phase and activity distributions. Make it scalable for size and number of development iterations.
Candidate Term Projects (cont.)

• Use process concurrence to model different processes for Rapid Application Development (RAD). Enable tradeoff studies.

• In a sort of reverse Brooks's Law effect, model the effect of extra work required when someone leaves a project
  - There is often additional coordination work to sort out the lost person's tasks. Since there are fewer workers and the project is slowed down, management may increase the pressure. This may then lead to others leaving.
Candidate Term Projects (cont.)

- Develop a long time horizon model to simulate productivity and estimation accuracy trends
  - Consider the Figure 6.1 in Boehm et al. regarding productivity and estimation accuracy trends. Consider this reference behavior for the cost estimation process. It should include the effects of technology innovations over time, organizational learning curves, possibly delayed flows of information to the software estimation process, and facilities to calculate estimation error. Augment or refine the reference behavior as appropriate.
Candidate Term Projects (cont.)

• Develop models that address any of the recommended future directions in combining static and dynamic modeling
  – Dynamic COCOMO variants
  – Derive phase-sensitive multipliers for selected cost drivers

• Investigate the schedule relationships in COCOMO with system dynamics
  – Model the software process effects of schedule compression strategies without any COCOMO formulas and compare the simulation model behavior against COCOMO predictions
Candidate Term Projects (cont.)

- Model the Rechtin rubber-band schedule heuristic
- Elaborate chosen Weinberg causal loop diagrams into simulation models
- Product Line reuse modeling
  – see Madachy first-cut for Litton core software reuse
- Dynamic effects of tools across lifecycle
- Many other possibilities
## Available Extensive Models for Testing and Modification

<table>
<thead>
<tr>
<th>Model Filename</th>
<th>Author(s) / Source</th>
<th>Major focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEASauth.itm</td>
<td>Steven Burke</td>
<td>organizational process improvement</td>
</tr>
<tr>
<td>inspect.itm</td>
<td>Ray Madachy</td>
<td>dynamic project effects of incorporating inspections</td>
</tr>
<tr>
<td>abdel_hamid.itm</td>
<td>Margaret Johnson</td>
<td>an lthink version of Abdel-Hamid's software project dynamics model</td>
</tr>
<tr>
<td>tvedt.itm</td>
<td>John Tvedt</td>
<td></td>
</tr>
<tr>
<td>project.itm</td>
<td>Doug Sycamore</td>
<td>three increment project model</td>
</tr>
<tr>
<td></td>
<td>Iona Rus</td>
<td></td>
</tr>
<tr>
<td>1phase.itm</td>
<td>David Ford</td>
<td>single phase dynamic development project model (with process concurrency and iteration)</td>
</tr>
<tr>
<td>DNFProjProcess.mdl</td>
<td>David Ford</td>
<td>multiple phase dynamic development project model (with process concurrency and iteration)</td>
</tr>
</tbody>
</table>
Outline

• Software process lifecycles
• RAD hypothesis testing guidelines
• Example simulation project
• Some modeling heuristics
• Term projects
  – candidate problems
  – existing models for modification/testing
• Brookes’s Law homework review and discussion
• Homework
  – delay example
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

\[ \text{developed\_software}(t) = \text{developed\_software}(t-1) + (\text{software\_development\_rate} \times dt) \]
\[ \text{INIT\_developed\_software} = 0 \]

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

**INFLUENCES:**
- **software\_development\_rate**
  - nominal\_productivity\( \times (1 - \text{communication\_overhead\%}) \times (\text{new\_project\_personnel} \times (0.1 \& \text{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**
  - experienced\_personnel\( \times (t - dt) \times (\text{assimilation\_rate} \times dt) \]
  - **INIT\_experienced\_personnel\_needed\_for\_training = 100**

**DOCUMENT:** The number of experienced personnel needed for training.

**INFLUENCES:**
- **assimilation\_rate**
  - new\_project\_personnel\( \times 20 \)

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

- **new\_project\_personnel**
  - new\_project\_personnel\( \times (\text{personnel\_allocation\_rate} \times \text{assimilation\_rate}) \]
  - **INIT\_new\_project\_personnel = 0**

**DOCUMENT:** The number of new project personnel.

**INFLUENCES:**
- **personnel\_allocation\_rate**
  - pulse\( (0, 100, 0.999) \)

**OUTFLUENCES:**
- **assimilation\_rate**
  - new\_project\_personnel\( \times 20 \)

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

**INFLUENCES:**
- **requirements\( \times t\) = requirements(t-1) + (software\_development\_rate \times dt) \]
  - **INIT\_requirements = 500**

**DOCUMENT:** The project size is 600 function points. This level represents the number left to be implemented.

**OUTFLUENCES:**
- **software\_development\_rate**
  - nominal\_productivity\( \times (1 - \text{communication\_overhead\%}) \times (\text{new\_project\_personnel} \times (0.1 \& \text{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.

**INFLUENCES:**
- **experienced\_personnel\_needed\_for\_training**
  - new\_project\_personnel\( \times \text{training\_overhead\%} \times \text{FTE\_experienced}\)

**DOCUMENT:** Training overhead is the effort expended by experienced personnel to bring new people up to speed. It is the number of new personnel \( t \) the percent of an experienced person's time dedicated to training.

- **nominal\_productivity**
  - nominal\( (\text{adjusted\_productivity\%})\)

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

- **total\_personnel**
  - experienced\_personnel\( \times \text{new\_project\_personnel} \)

**DOCUMENT:** Percent of FTE equivalent personnel experienced personnel's time dedicated to training new hires.

- **communication\_overhead\%**
  - (experienced\_personnel\( \times \text{new\_project\_personnel} \))

**DOCUMENT:** Percent of time spent communicating with other team members as a function of team size. This graph represents the \( \sqrt{t} \) 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:
  – *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
Outline

• Software process lifecycles
• RAD hypothesis testing guidelines
• Example simulation project
• Some modeling heuristics
• Term projects
  – candidate problems
  – existing models for modification/testing
• Brookes’s Law homework review and discussion

• Homework
  – delay example
Homework

• Complete Brooks’s Law model

• Start deciding on a term project and writing your proposal
  – turn in draft in 2 weeks

• Modeling assignments (both due next week)
  1: model the first two items in the DPRS rating scale: number of approvals per task and time taken per approval
    • show results from different cases; i.e. heavily burdened vs. little bureaucracy
    • augment for probability of delays
Homework (cont.)

2: develop a very simple model of software production
   • include levels for personnel and work artifacts, and have a productivity variable
   • calibrate parameters as you see fit, and document your rationale
   • show simulation runs

• Readings:
  – Introduction to Systems Thinking and Ithink
    • Complete Part 1 through Chapter 6
  – Software Process Dynamics
    • handout for sections 1.2.2 -1.3.2