How Does Agile Speed Government Development?

Using Behavioral Modeling and Simulation to Explore, Refine, and Validate Government Applications of Agile

Presenter:

Andrew P. Moore (apm@sei.cmu.edu)
Software Engineering Institute
412-268-5465

Contributors:

William E. Novak
William R. Nichols
David Zubrow

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM19-0290

Introduction

Objective: Illustrate how behavioral modeling and simulation (BModSim) can help identify key insights and answer key questions of agile concepts and application in the USG

BModSim does NOT refer to

- Traditional Model-Based Software Engineering (MBSE)
- Modeling and simulation as used in traditional war-gaming applications

We use BModSim to describe (model) software-intensive system acquisition processes

- Includes PMO management processes of the as well as contractors' software development processes
- Process execution (simulation) allows evaluating these processes and improving their performance
- Common BModSim tools support agent-based modeling, system dynamics modeling, computational game theory, dynamic network analysis

Could be called **Model-Based *Human System Analysis* (MBHSA)**

Why Does BModSim Matter for Agile?

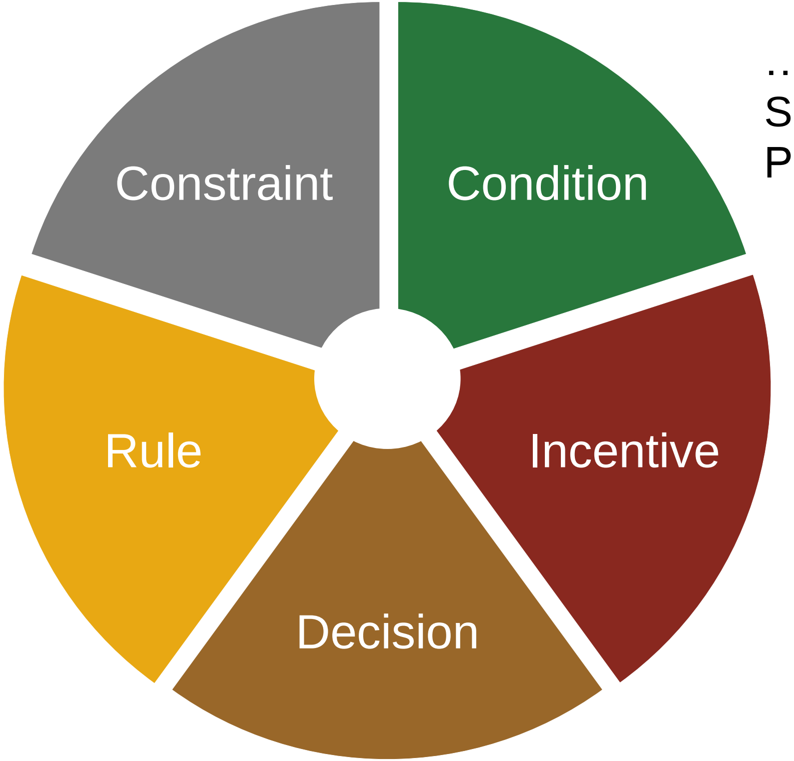
Key challenges:

- Proliferation of many variants of agile development approaches
- Impacts of refactoring large systems for incremental development
- Difficulty of conducting experiments to try out new ideas

BModSim can help to (prior to or in absence of experimentation)

- Expose operation of underlying mechanisms that make agile useful in government programs
- Understand exactly *how* different agile approaches create value, and when they do *not*
  - analyze the potential cost, schedule, and quality implications
- Test efficacy of software development policy and process through simulation
- Construct valid, coherent, and executable characterizations of agile software development
  - Models characterize the larger context of analysis – have good data on some variables, not others
  - Monte Carlo simulation allows identifying the most important places to gather more data

Components of Behavioral Models



…so with
Limited Staff
Size…

…there's
Schedule
Pressure…

More Requirements
Mean More Effort…

…but Low QA
Means More
Defects…

…so to
Maintain
Reputation…

…you Streamline QA…

Conceptual Overview of Model Development

Specific Goal:

- Explore the benefits of one particular aspect of agile over monolithic system development : ***incremental development*** (includes re-baselining to accommodate late requirements)

Model Parameters:

- size of baseline
- late requirements level
- # of increments
- rigor of unit defect removal
- realism of schedule setting
- staffing levels
- For monolithic:
    - frequency of schedule extension
    - max # of schedule extensions

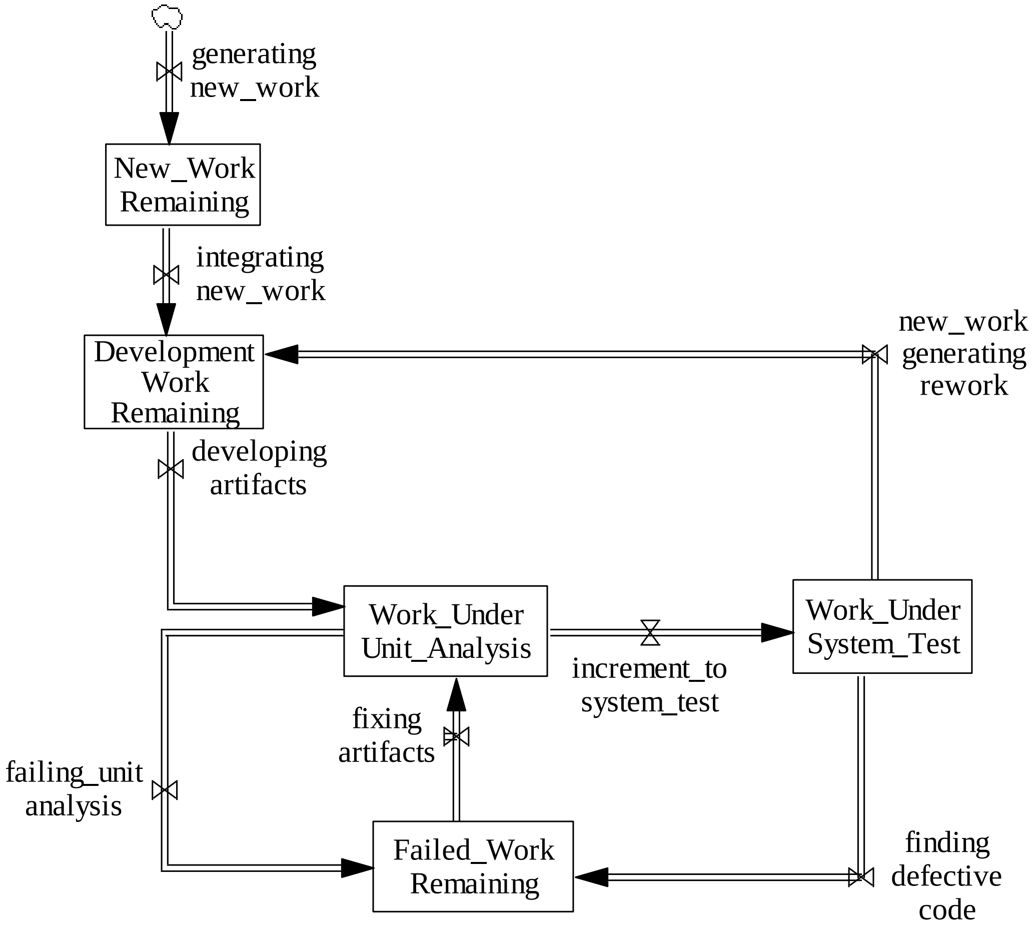
Primary Finding:

- Incremental development can reduce the impact of late requirements by introducing them into normal increment planning as they arise
    - Analysis assumes monolithic and incremental development subject to same level of late reqs

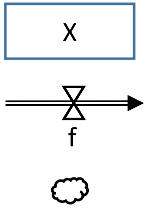
CAVEATS:

- Model is work in progress! Only models one aspect of agile development. Limited validation
- Does not solve problems due to contractual regulation rigidity inhibiting incremental development

The Basic Stock and Flow Infrastructure*



* Notation:

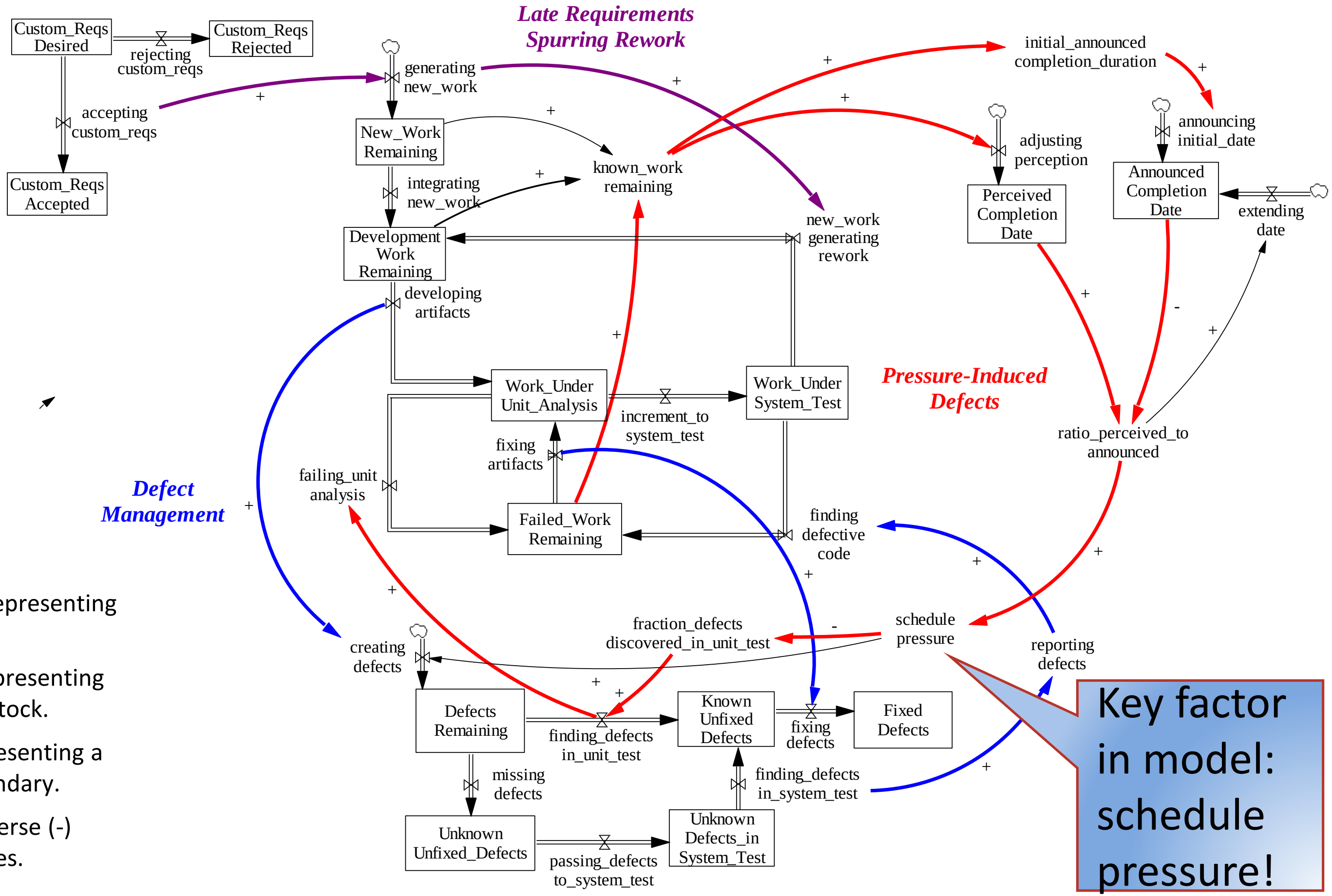


- Stock: a variable named X representing an accumulation of artifacts.

- Flow: a variable named f representing an inflow or outflow from a stock.

- Cloud: a source or sink representing a stock outside the model boundary.

Full System Dynamic Model Overview*



Key factor in model: schedule pressure!

* Notation:

X

- Stock: a variable named X representing an accumulation of artifacts.

f

- Flow: a variable named f representing an inflow or outflow from a stock.

- Cloud: a source or sink representing a stock outside the model boundary.

+/-

- Influence: a direct (+) or inverse (-) relationship between variables.

Model Baseline Common Values and Assumptions:
Parameterized for Evaluation Purposes

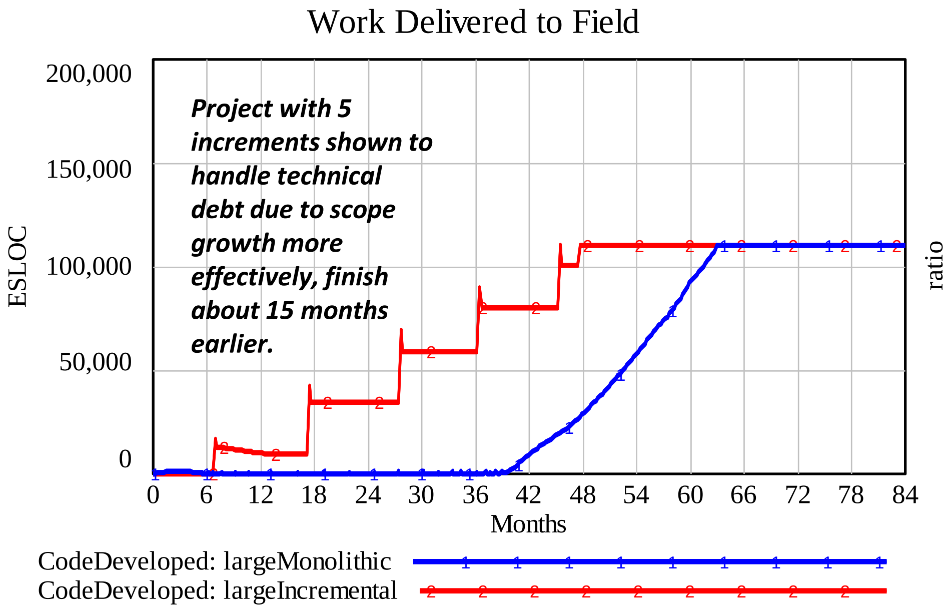
- 1026 Initial Requirements
- 74 Late Requirements
- 100 ESLOC per Requirement (on average)
- 15.5 Development Staff (FTE)
- Defects model-generated based on level of schedule pressure
  - Monolithic development granted up to 2 schedule extensions total, review every 24 months
  - Incremental development delays late requirements introduction to next increment and develops new schedule from there
- Defect Repair Time generated from
  - a log normal distribution
  - with average of 2.2 days for one person to repair a defect
- Total Rework includes 1) defect fixes and 2) rework due to late requirements

Model Calibration

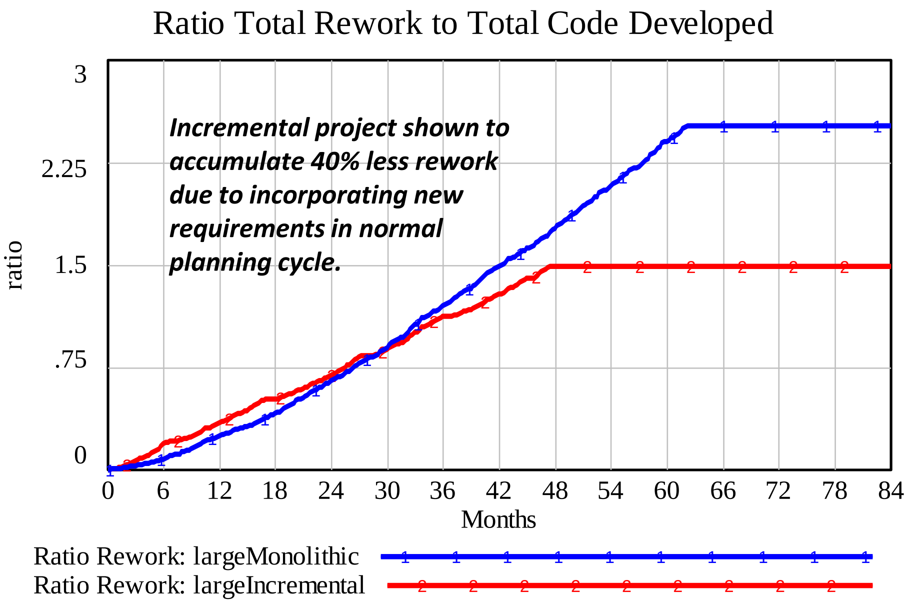
External Source	Factor calibrated	External Value (Average)	Model Generated Value (Optimized Fit)
DOD Software Factbook (July 2017) Averages for ***Large*** Projects	Baseline SW Size	110 K-ESLOC (1100 reqs)	110 K-ESLOC (1100 reqs)
	Project Duration	48.3 months	47.75 months
	Developer Effort	97K hours	73.4K hours
	Developer Staff	19.3 developers	15.5 developers
	Developer Overall Productivity	1.69 ESLOC/person-hour	1.5 ESLOC/person-hour
Estimates on Software Defects in Industry from TSP Research	Total Defect Potential	40 to 60 defects/K-ESLOC (using IDE)	37 defects/K-ESLOC
	Level of Unit Test Yield	80% defects discovered in unit test	74% defects discovered in unit test
	Defect Rates Entering System Test	2 to 10 defects/K-ESLOC	9.8 defects/K-ESLOC
	Overall Defect Repair Rate	6.9 defects/person-month	9.7 defects/person-month

- Model calibrated to provide a best fit of measures from the DoD Software Factbook and measures for industry from TSP research.
- TSP data represents higher performers.

Comparison of Large Project Development: Monolithic vs Incremental*



Project with 5 increments shown to handle technical debt due to scope growth more effectively, finish about 15 months earlier.



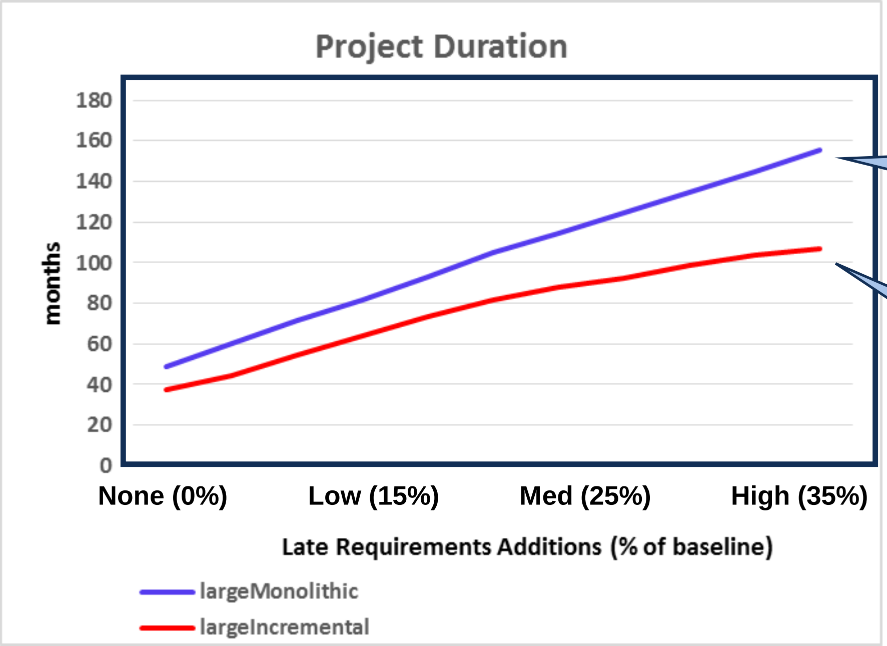
Incremental project shown to accumulate 40% less rework due to incorporating new requirements in normal planning cycle.

- Bar and behavior-over-time graphs show simulation occurring over 84 months (7 years). Each simulation run is specified with a number label as shown in the legend below the graph.

Overview Comparison of Monolithic vs Incremental

Measure	Monolithic	Incremental	Units
Overall Development Productivity	1.08	1.50	ESLOC/person-hour
Overall Defect Repair Productivity	10.6	9.8	defect/person-month
Total Defects	7039	4074	defects
Total Defect Potential	64	37	defects/K-ESLOC
Defect Rate Entering System Test	30.5	9.8	defect/K-ESLOC
Ultimate Quality Level in Terms of % Defects Fixed	Equivalent (for comparability)		
Ratio Total Rework to Total Code	2.5	1.5	ratio

Incremental Development Tolerates Scope Growth Better

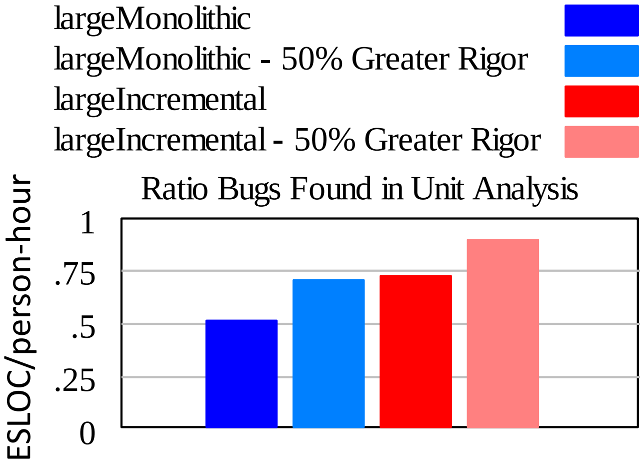


Note that for monolithic development a 35% increase in scope results in over tripling the project schedule.

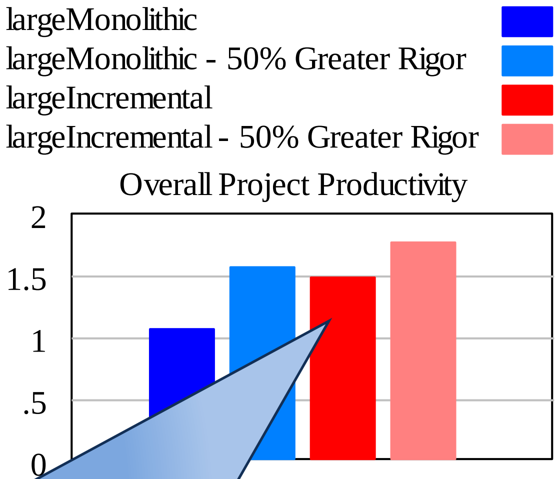
Incremental development reduces schedule growth by 40%. Reason: reduced rework by managing reqs growth in increment planning.

ModSim Also Helps Characterize/Assess Value of Processes:

Effects of Rigor of Unit Defect Removal (unit test, peer review, static analysis)

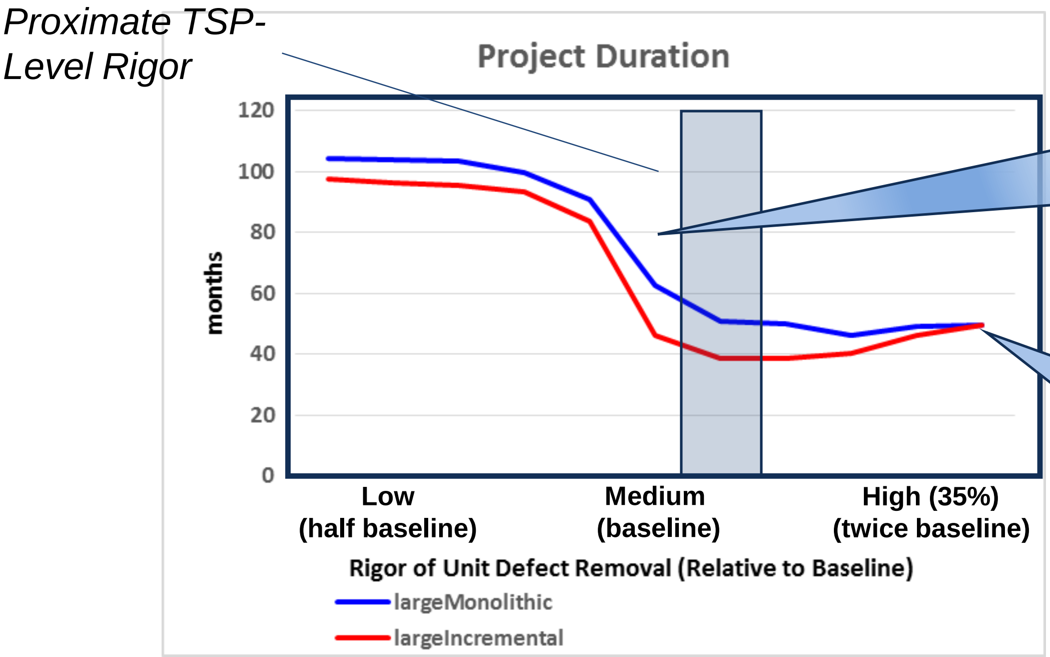


50% greater rigor of unit defect removal does increase bugs found at unit level resulting in fewer problems later in process. This enables greater overall productivity despite reduced developer productivity up front.



Greater rigor does better for monolithic than going incremental by itself, but returns on going both incremental with greater rigor are best.

Optimal Level of Rigor of Unit Defect Removal Process is Not at Either Extreme



Greater rigor of unit defect removal slows initial development productivity but also decreases the problems found later.

Diminishing returns results in slight productivity drop and increased duration. Extreme rigor reduces differences between monolithic and incremental, but incremental better at optimum.

Conclusions

Preliminary observations from the model show that incremental development accommodates requirements growth without as much disruption to schedule as monolithic system development

- Primary factors: inflexible schedule, schedule pressure, process shortcuts, rework

This is a work in progress, but demonstrates the potential value of using BModSim in combination with traditional data analytic techniques to

- Expose the operation of underlying mechanisms that make agile useful in a government program context
- Understand exactly how and when different agile approaches create value
- Test the efficacy of software development policy and process through simulation before implementation
- Compare model behavior with real world behavior

More work is needed in model refinement, validation, and calibration.

Call for volunteers as data suppliers and subject matter experts to help!

Questions and Contact

QUESTIONS?

Contact:

Andrew P. Moore
Software Engineering Institute
Carnegie Mellon Institute
4500 Fifth Avenue
Pittsburgh, PA 15213
apm@sei.cmu.edu
412-268-5465