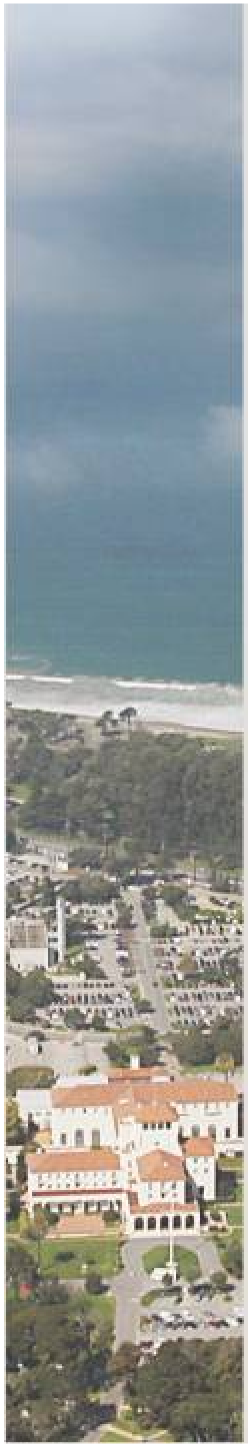


NAVAL
POSTGRADUATE
SCHOOL

An information-theoretic approach to software test-retest problems

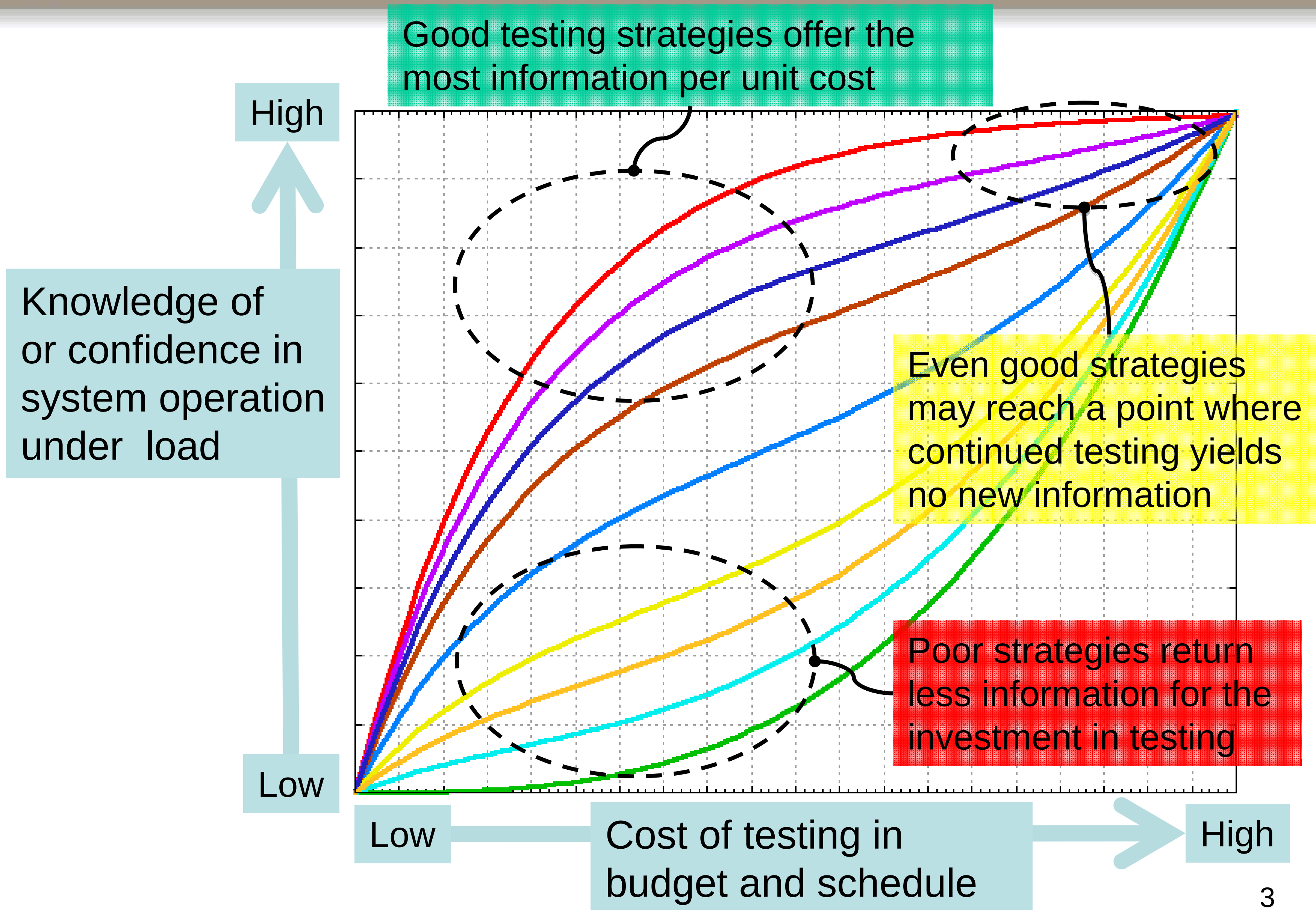
May 13, 2010

Karl D. Pfeiffer
Valery A. Kanevsky
Thomas J. Housel

Overview

- Open architectures (OA) and reusable software components offer the promise of more rapid fielding of increments in systems development
- Testing and re-testing these components requires a significant level of effort as new systems are developed and old systems are upgraded
- *How much testing is enough? When can we stop testing?*

Motivation


Good testing strategies offer the most information per unit cost
High
Knowledge of or confidence in system operation under load
Low
Even good strategies may reach a point where continued testing yields no new information
Poor strategies return less information for the investment in testing
Low
Cost of testing in budget and schedule
High

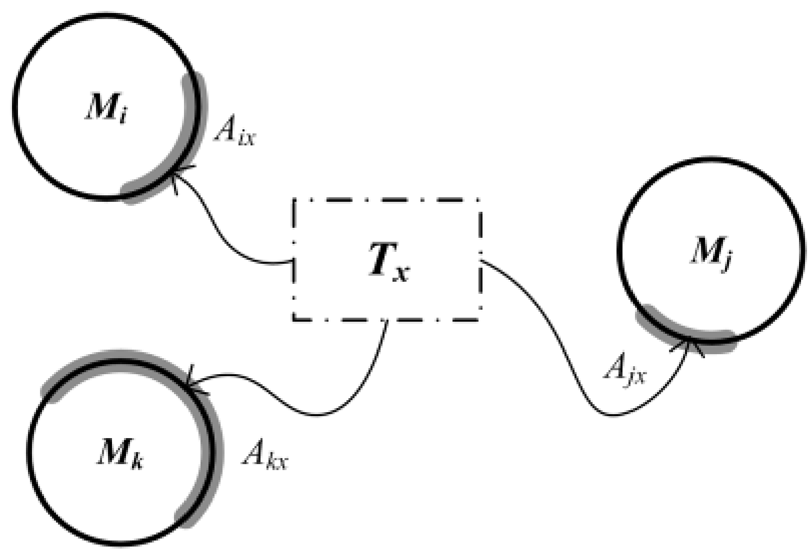
Model fundamentals

- We can identify good testing strategies by constructing a simple model of the system, its components, and its attendant test suite
- This model requires
  - Estimates of a prior probability of failure for modules within the system
  - Estimates of the coverage for each test in the suite over these modules
- These estimates need not be precise to make the model useful
  - Monte Carlo simulation can be used to sample around the estimates as means, offering some insight into model sensitivity

Model fundamentals

- This model should help answer questions like:
  - Given a desired level of confidence in system operation, how much testing should we accomplish? How much will this cost?
  - Given a fixed budget for testing, how much confidence in system performance can we achieve through testing?
  - Given a particular test suite, how much information is attainable given infinite resources?

Model fundamentals



- A module M_i is modeled as a unit circle with probability of being defective b_i
- Test T_x exercises region A_{ix} in module M_i
- In general we assume that T_x may exercise several regions across several modules

- A test has two possible outcomes:
  - *PASS* indicates that the test did not *detect* a defect in any of the exercised regions within the modules tested
  - *FAIL* indicates that at least one module exercised is defective, though we may not know which one

Model fundamentals

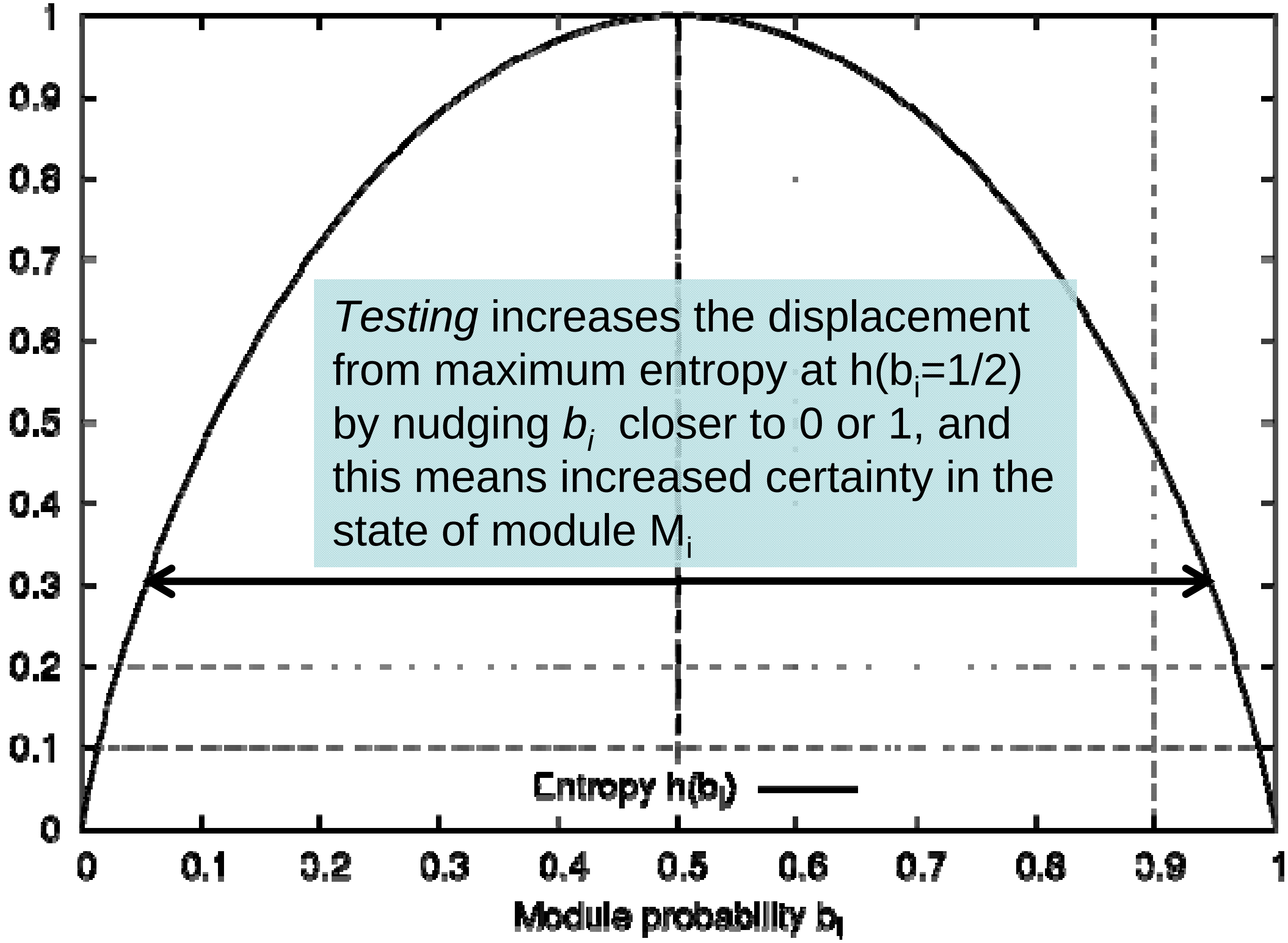
- These ambiguities offer a rich framework for modeling realistic system testing scenarios
  - We need not execute (and pay for) Tx to *forecast* information returned
  - Within this language of expression we can formulate a *quantitative* assessment of the information returned by a test sequence

- Across the system of modules M_i we can measure the information returned by a test using the classic residual entropy for a distribution of probabilities:

$$H = \sum_i h_i = \sum_i -b_i \log_2 b_i - (1-b_i)\log_2(1-b_i)$$

Model fundamentals

At maximum entropy we have a 50/50 chance that our module is good or bad—*we might as well flip a coin*



Testing increases the displacement from maximum entropy at $h(b_i=1/2)$ by nudging b_i closer to 0 or 1, and this means increased certainty in the state of module M_i

Model fundamentals

- From entropy, we derive the forecast measure:

$$Q(T_x) = \sum_i \left(\max(b_i^{fail}, 1 - b_i^{fail}) P(T_x \text{ fails}) + \max(b_i^{pass}, 1 - b_i^{pass}) P(T_x \text{ passes}) \right)$$

- Let c_x be the cost of executing test T_x in appropriate units of time or money (or both) A *good* strategy will sequence the suite of tests such that:

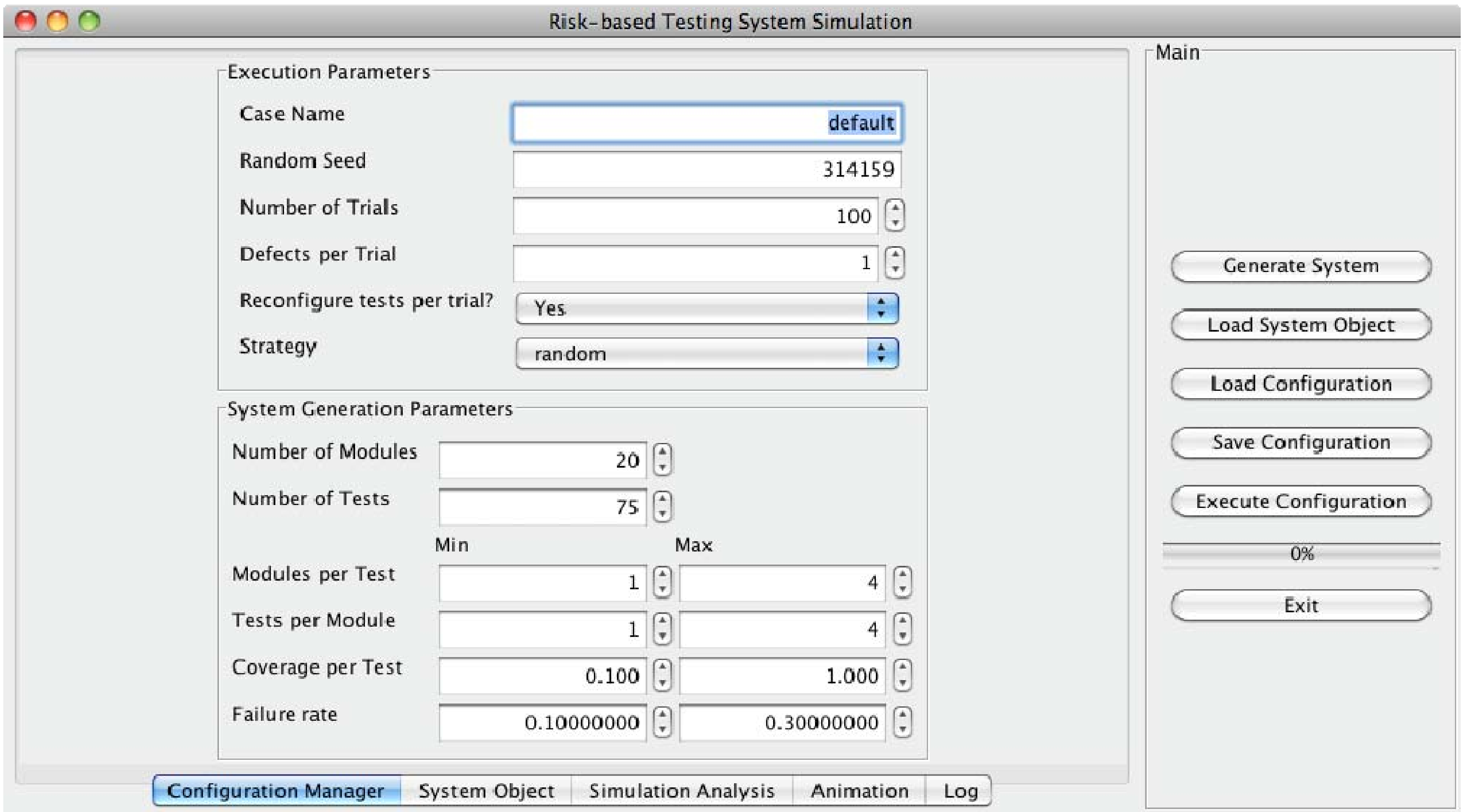
$$\frac{Q(T_{[1]})}{c_{[1]}} \geq \frac{Q(T_{[2]})}{c_{[2]}} \cdots \geq \frac{Q(T_{[m]})}{c_{[m]}}$$

- These ratios represent ***information per unit cost***

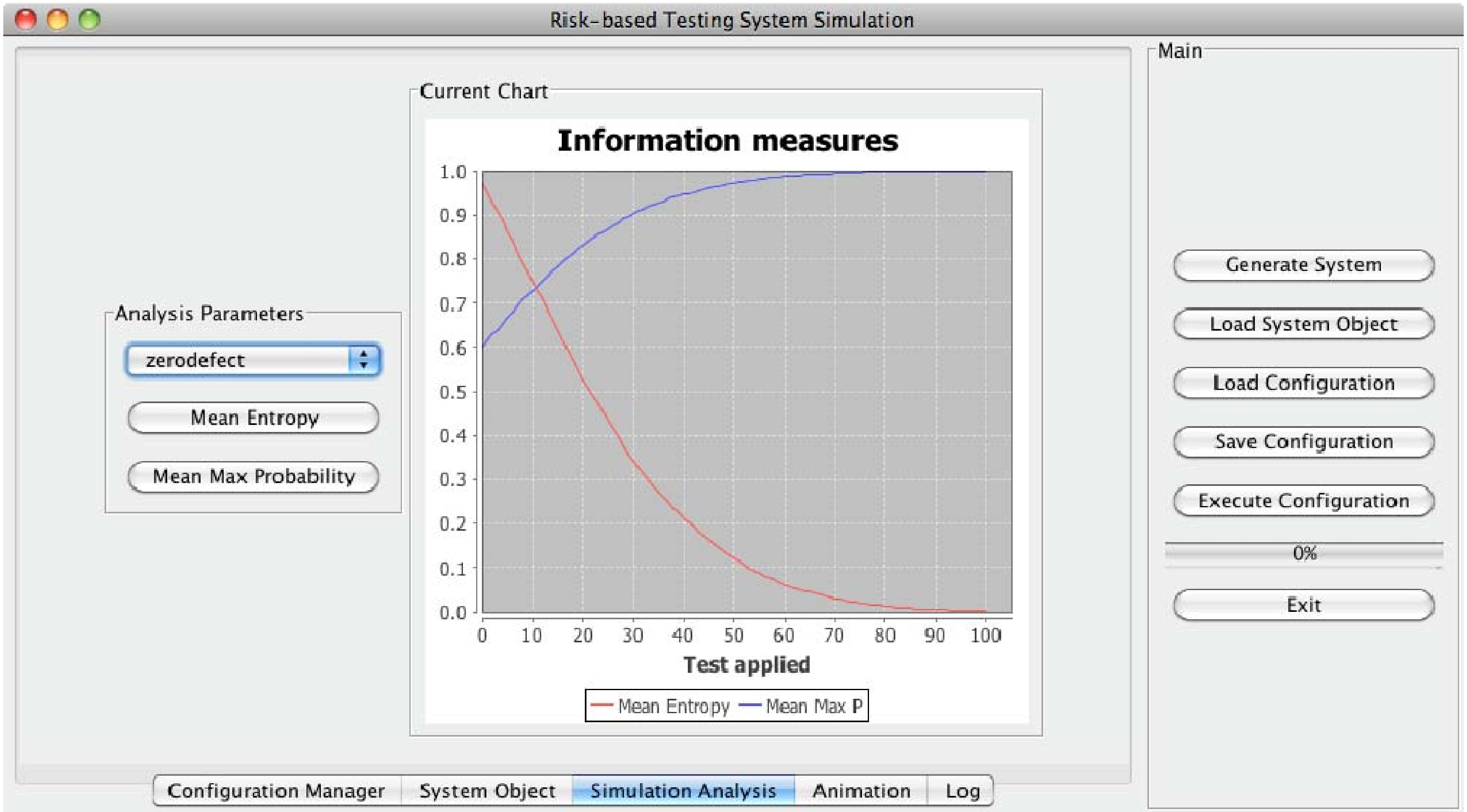
Model implementation

- Within the decision aid, for simple investigations, a fully randomized system can be created with only a few user specified constraints
- If the user has a few system details but only vague insight about others, these aspects can be augmented with randomized parameters (e.g. sizes and number of coverages)
- A system with well-documented interdependencies can be completely specified by the user in terms of modules, tests and coverages

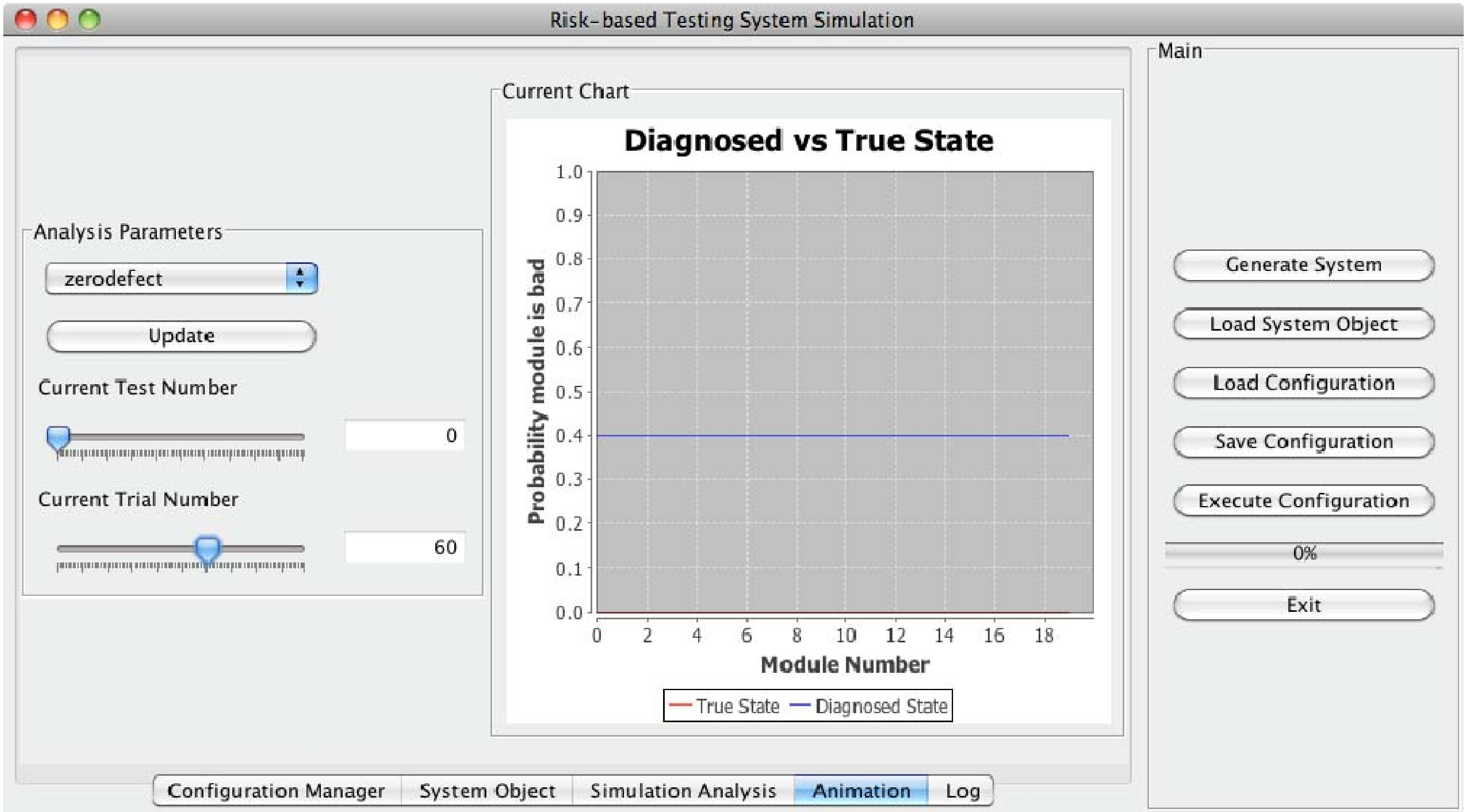
Model implementation



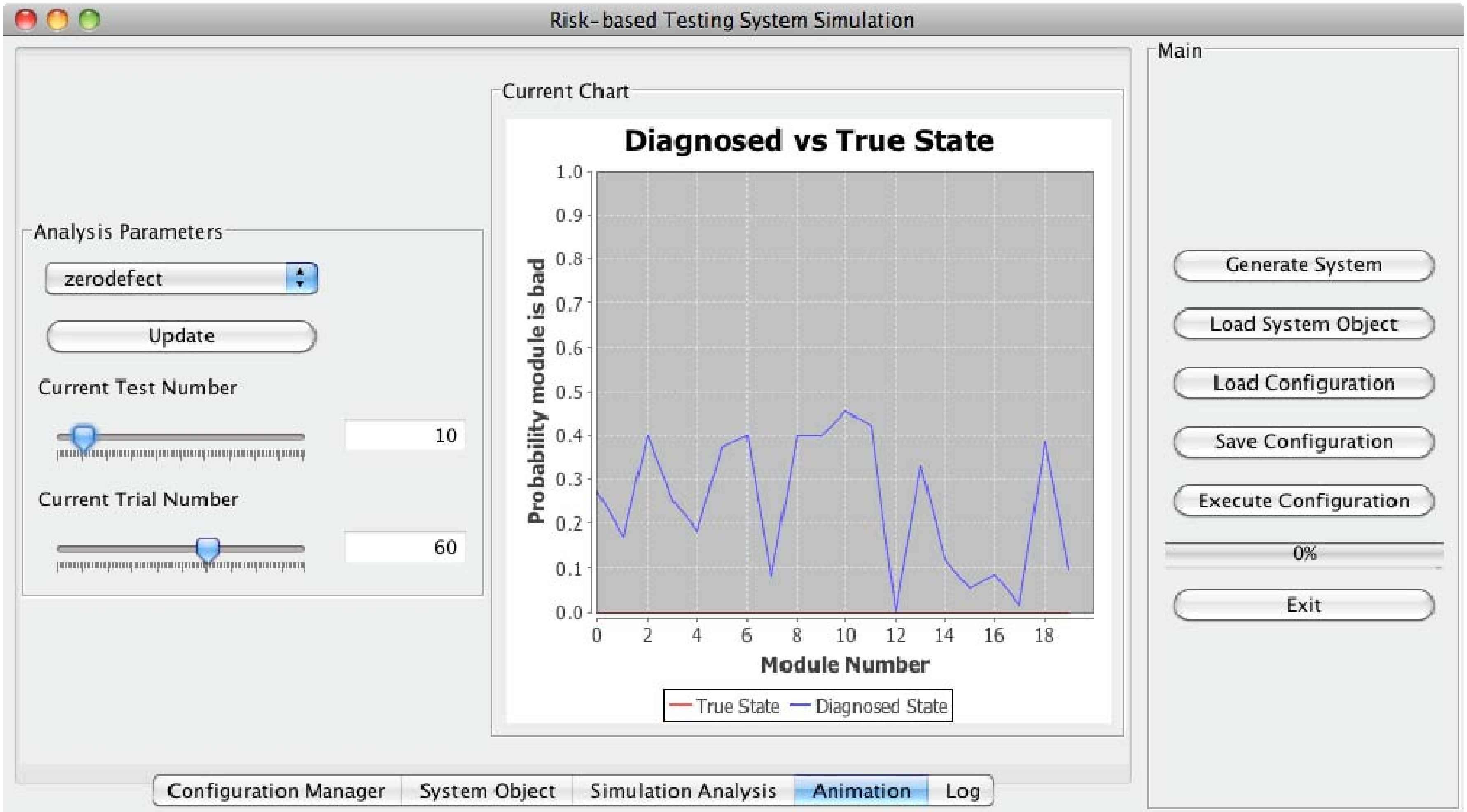
Model implementation



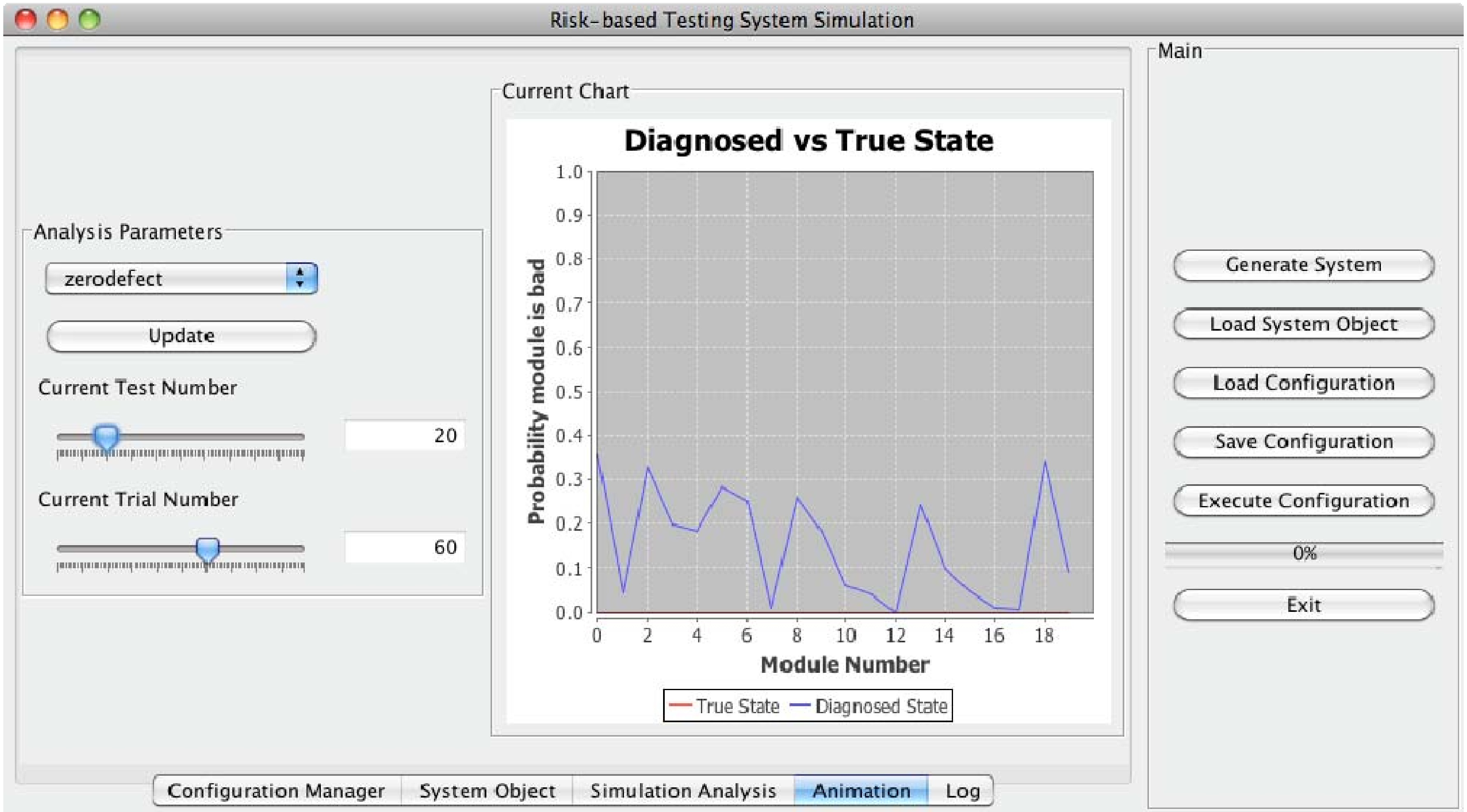
Model implementation



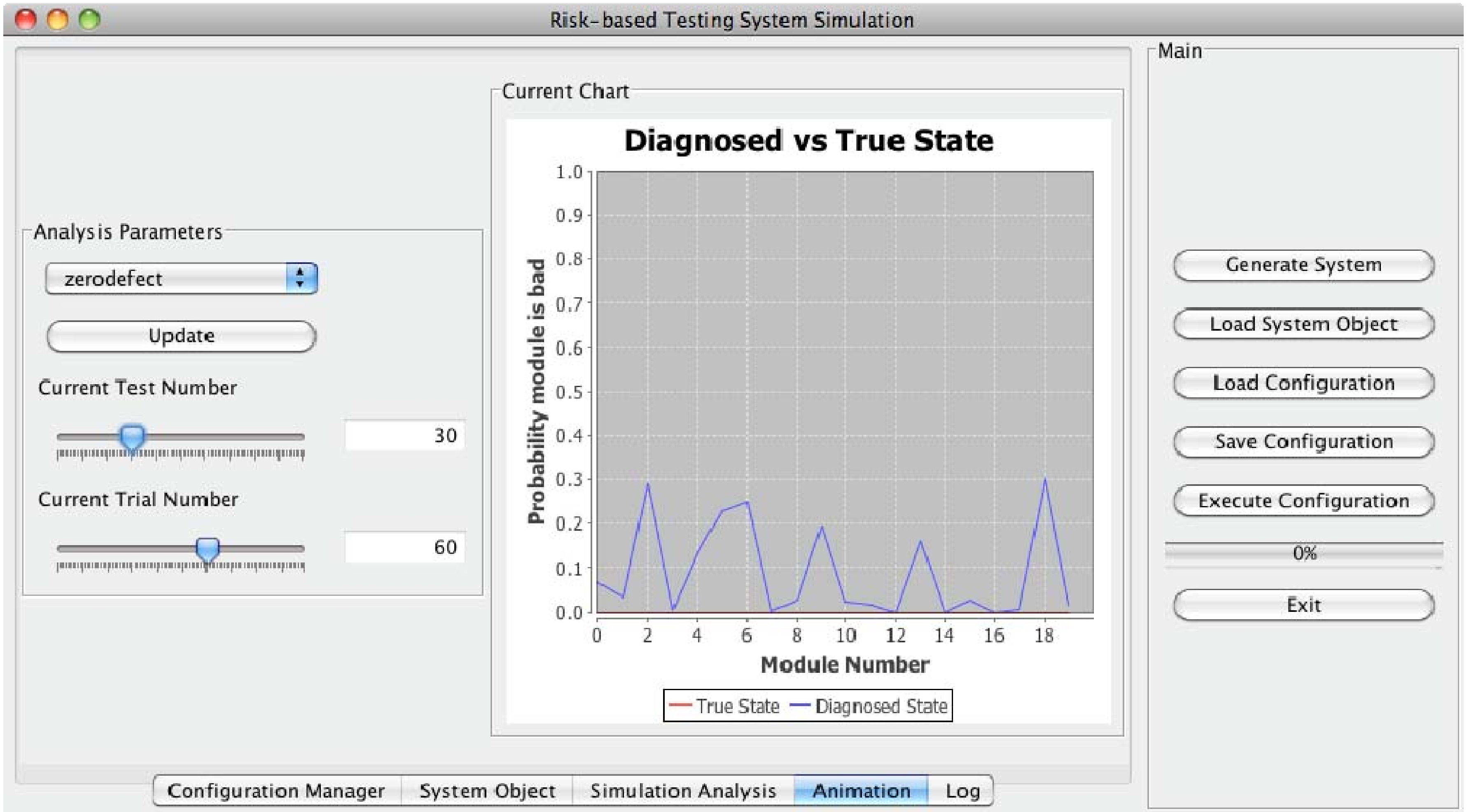
Model implementation



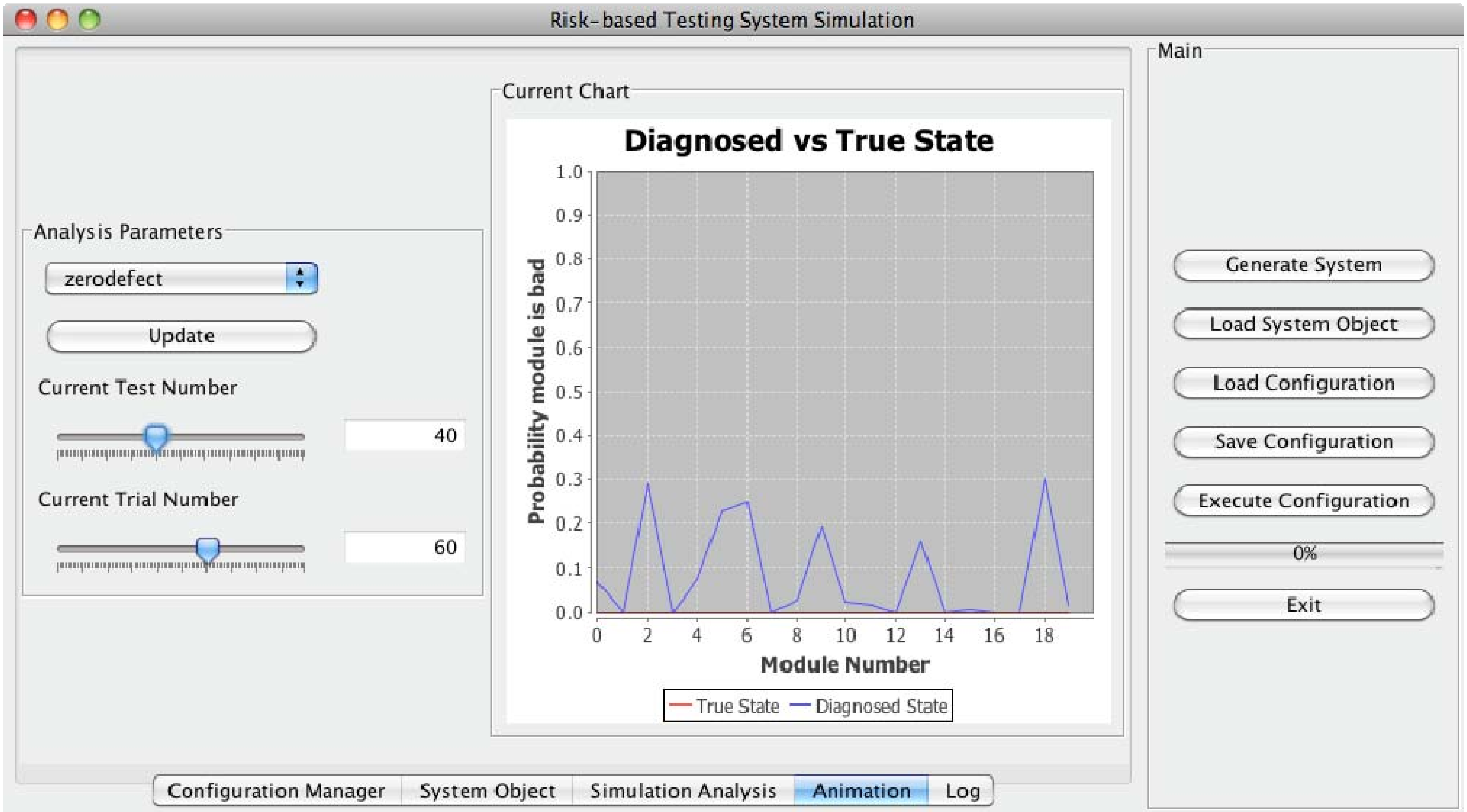
Model implementation



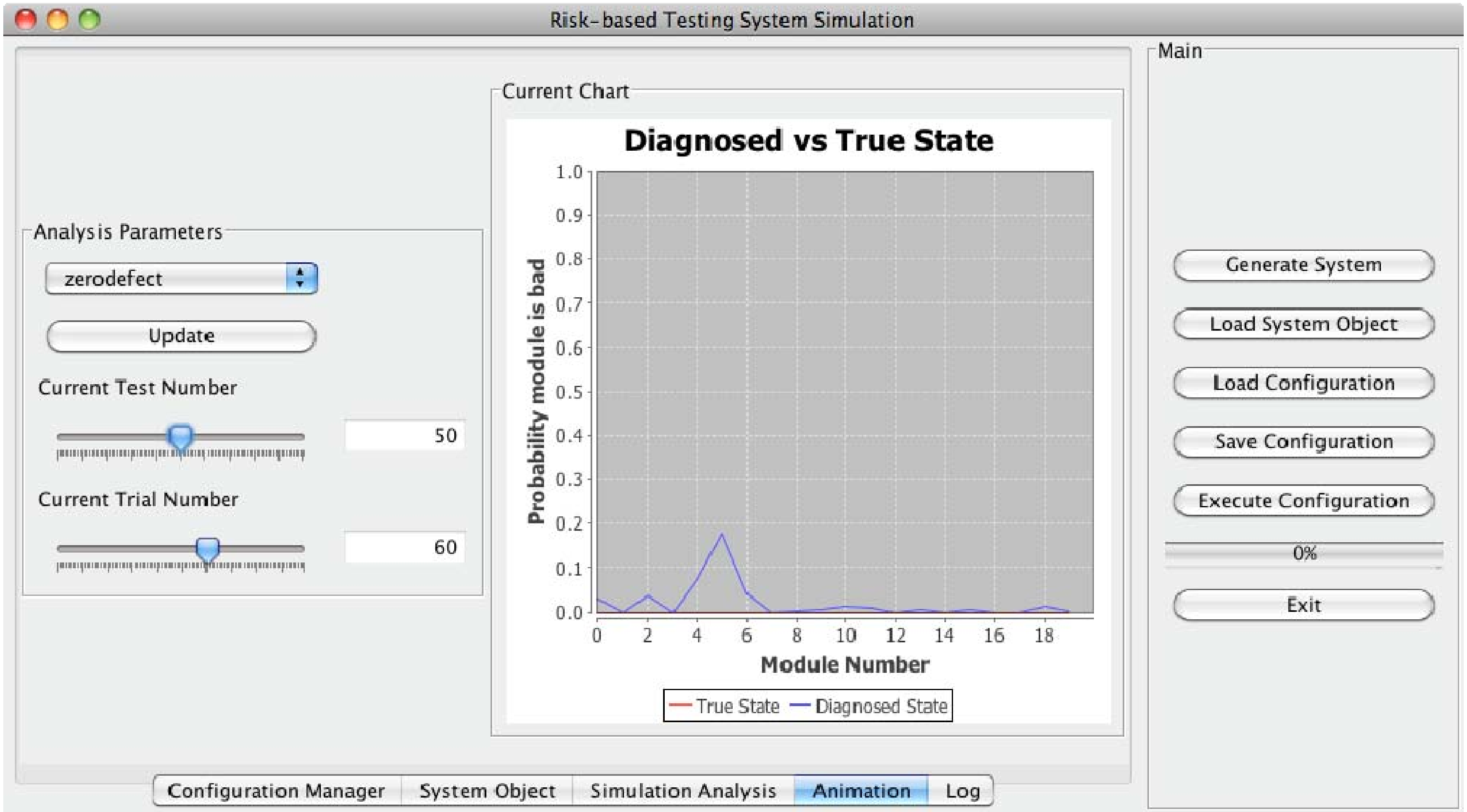
Model implementation



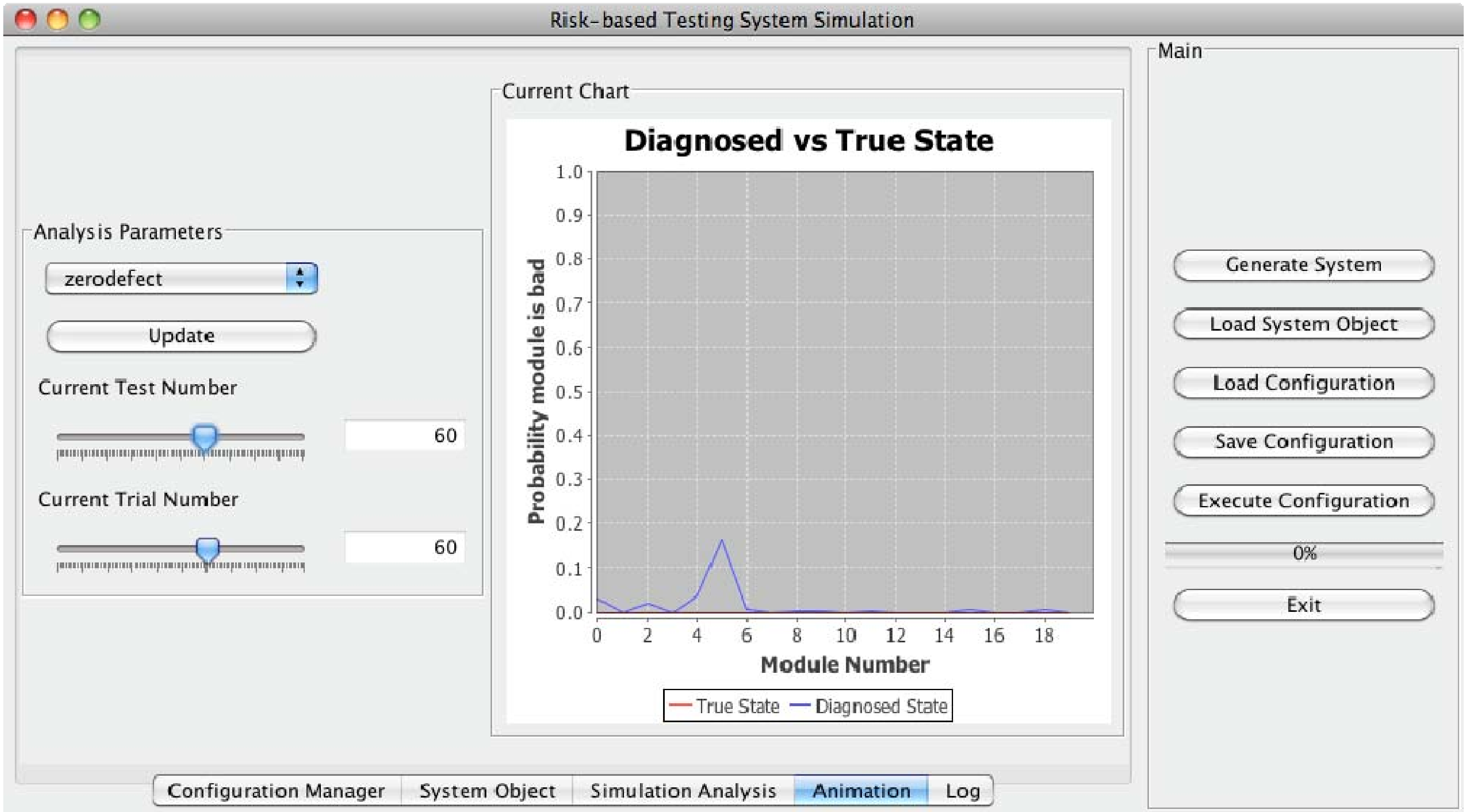
Model implementation



Model implementation

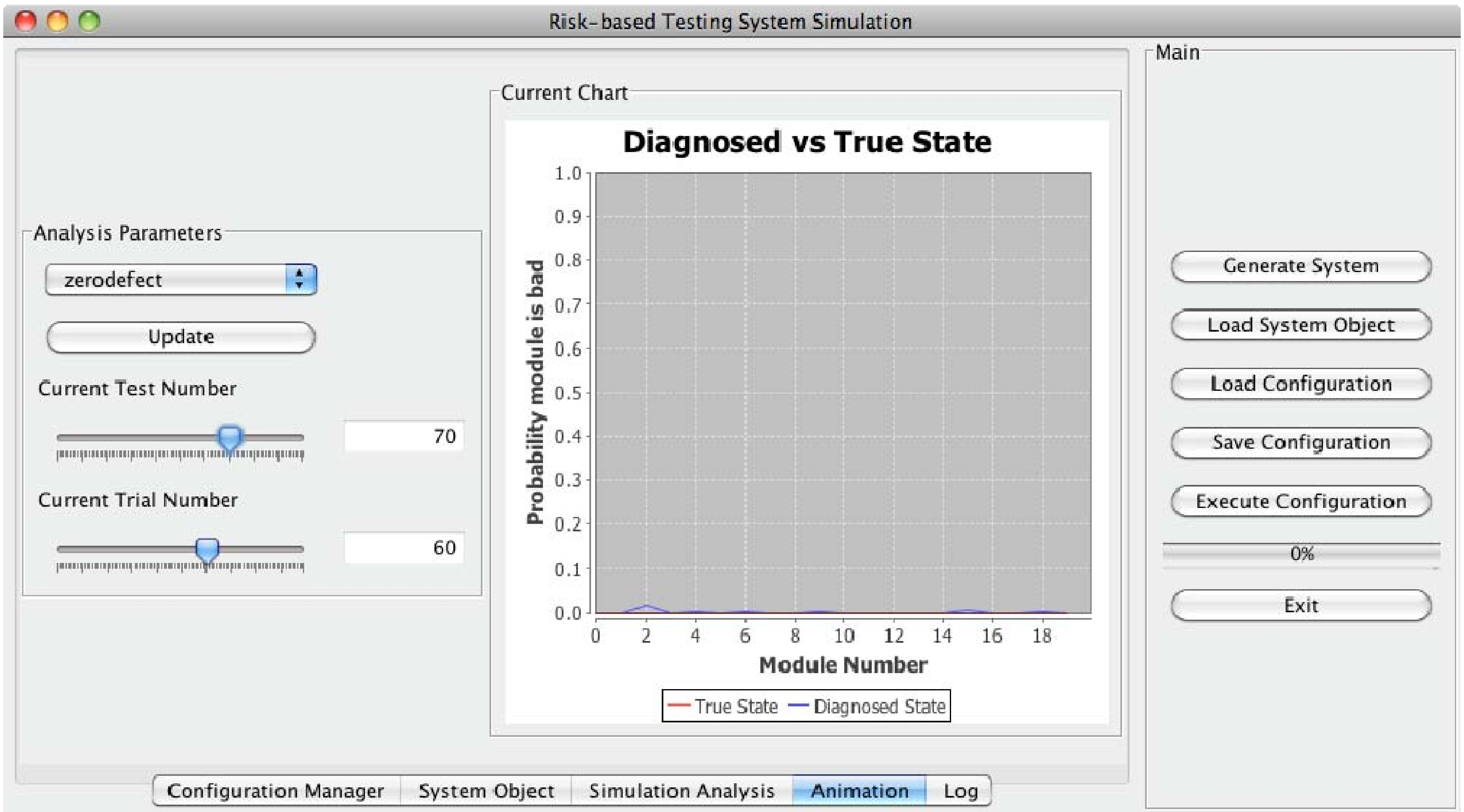


Model implementation



NAVAL POSTGRADUATE SCHOOL

Model implementation



NAVAL POSTGRADUATE SCHOOL

But, what does it all mean?

- *Effective, cost-efficient testing is critical to the long-term success of open architecture programs*

- This model and prototype decision aid provide a rigorous yet tractable way ahead to improve system testing

- Using this framework we can build the tools to:
  - Lower the testing costs for a given level of system reliability
  - Improve the use of existing suites for a given budget or schedule
  - Design better, more targeted test suites to minimize redundancy
  - Provide insight into the power or sensitivity of current test suites