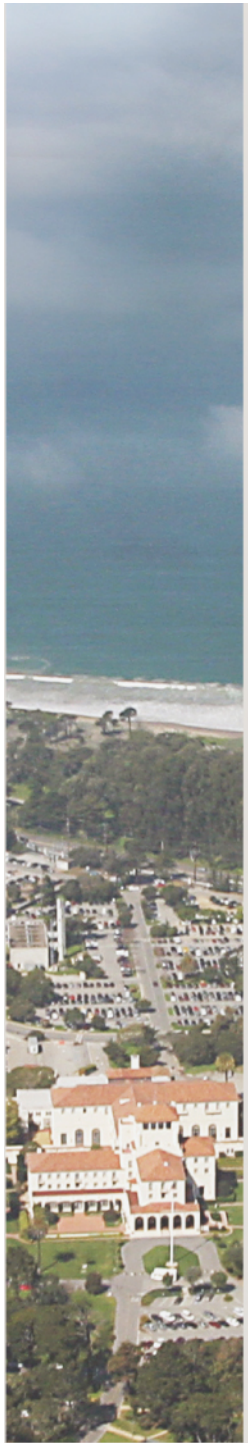


NAVAL
POSTGRADUATE
SCHOOL

Math modeling for risk-based testing:

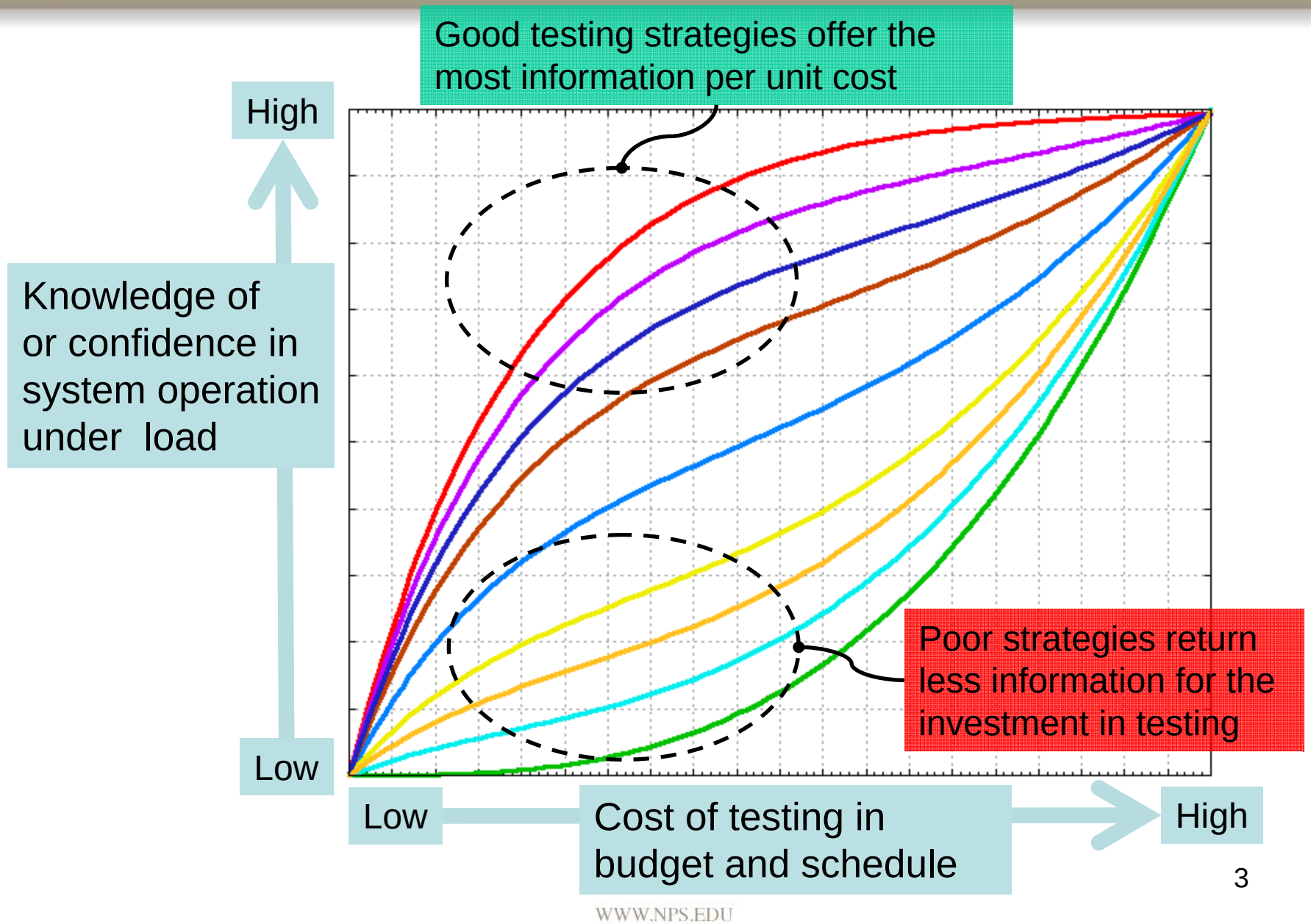
Information-driven strategies
to reduce cost and improve reliability

Karl D. Pfeiffer
Valery A. Kanevsky
Thomas J. Housel
Monterey, California
WWW.NPS.EDU

Overview

- This project seeks to provide a prototype decision aid to help control the cost of testing in an open architecture (OA) environment
- Implementation of OA can lead to more rapid fielding of increments in systems development
  - *However, frequent fielding requires frequent testing*
- This is one of two efforts funded by PEO-IWS 7 that seek to provide a rigorous basis for controlling spiraling cost of testing

Background

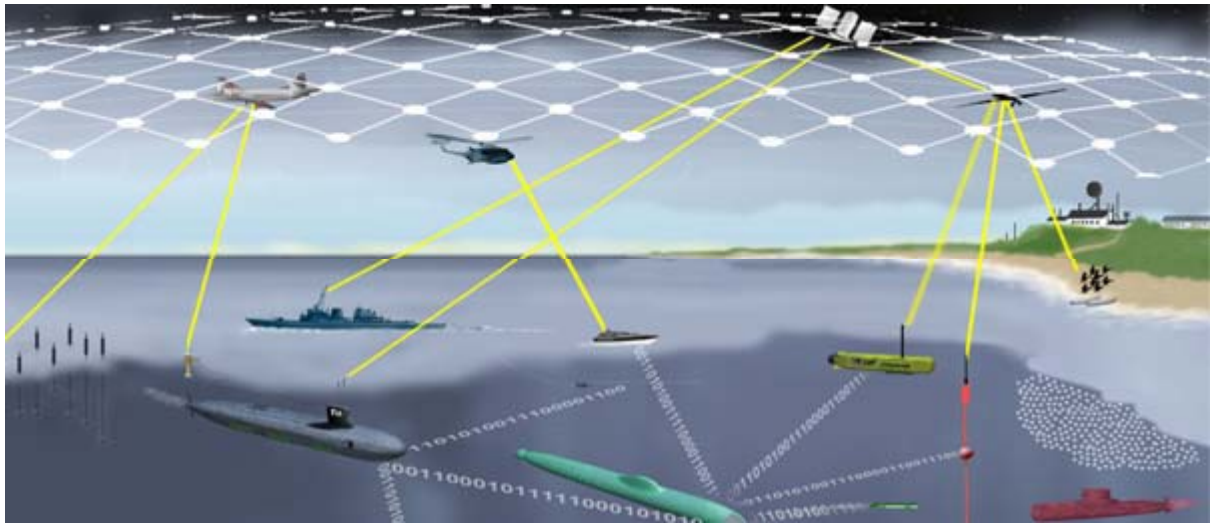

Good testing strategies offer the most information per unit cost
High
Knowledge of or confidence in system operation under load
Low
Poor strategies return less information for the investment in testing
Low
Cost of testing in budget and schedule
High

Model approach

- Classic approaches to optimal testing focus on the modules or components to be tested
- This is similar to building optimal search strategies for submarines using *only aircraft* as the reconnaissance platform, focusing on *the differences among submarines*
- While it's important to understand the components being tested, or the targets of our search, this can only take us so far—*and sometimes our targets are black boxes*

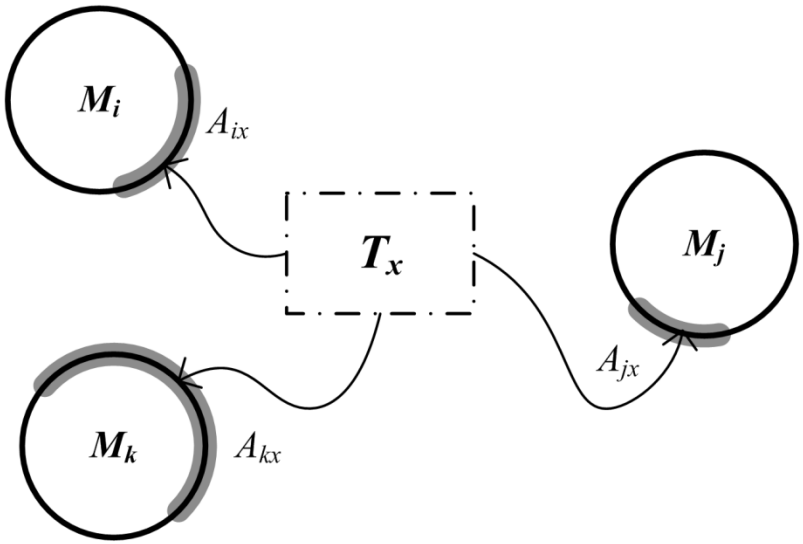
Model approach

- In the present work, we treat both *tests* and *components* explicitly, using prior knowledge of both our system and its diagnostic test suite to build an optimal test strategy



- This is similar to looking at all available platforms for the best mix of sensors (*tests*) to match the most probable or most lethal targets (*faulty components*)

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

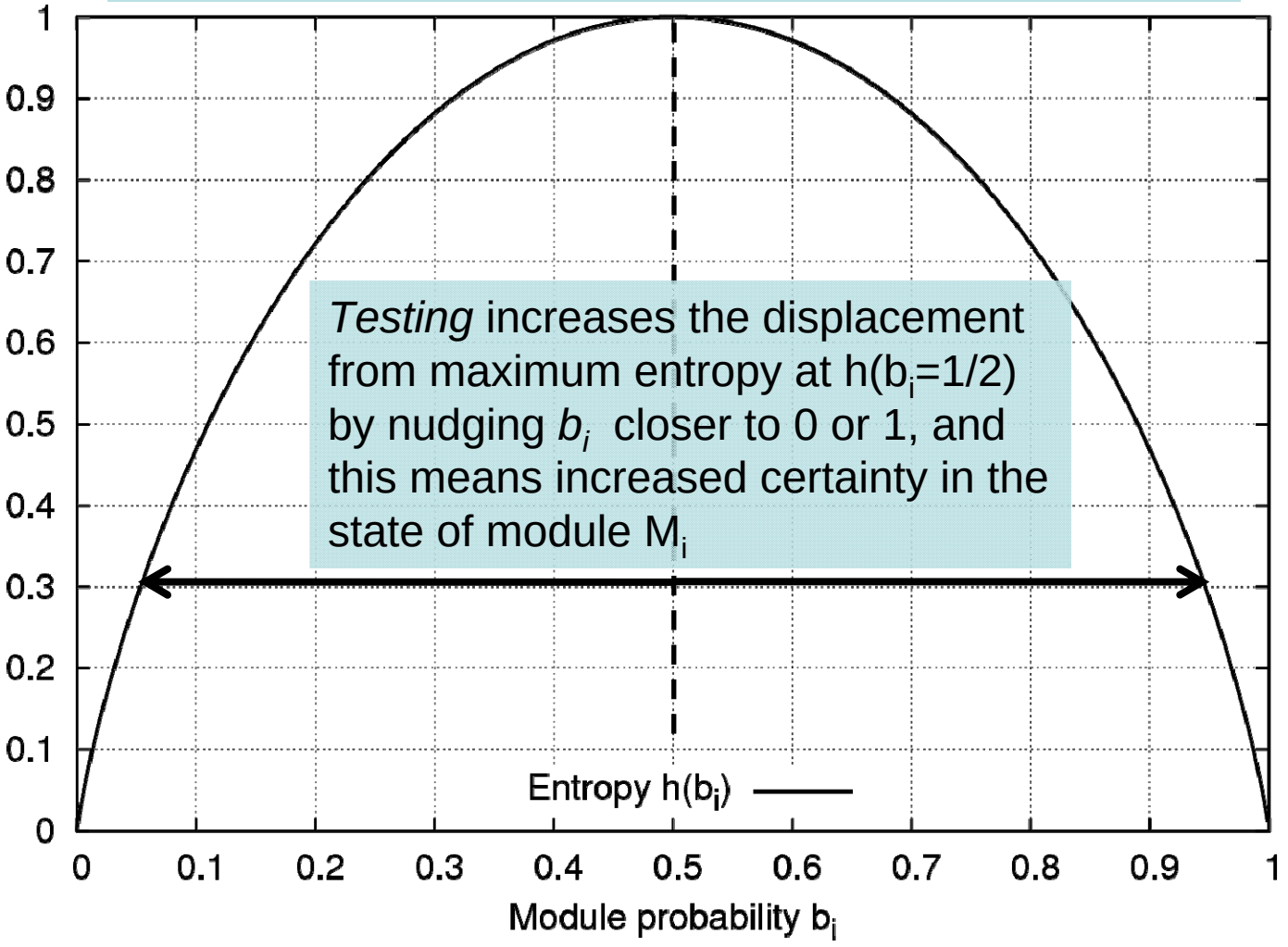
- These ambiguities offer a rich framework for modeling realistic system testing scenarios
  - We do not need to execute (and pay for) Tx to *forecast* the information returned by this test
  - Within this language of expression we can formulate a *quantitative* assessment of the information returned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

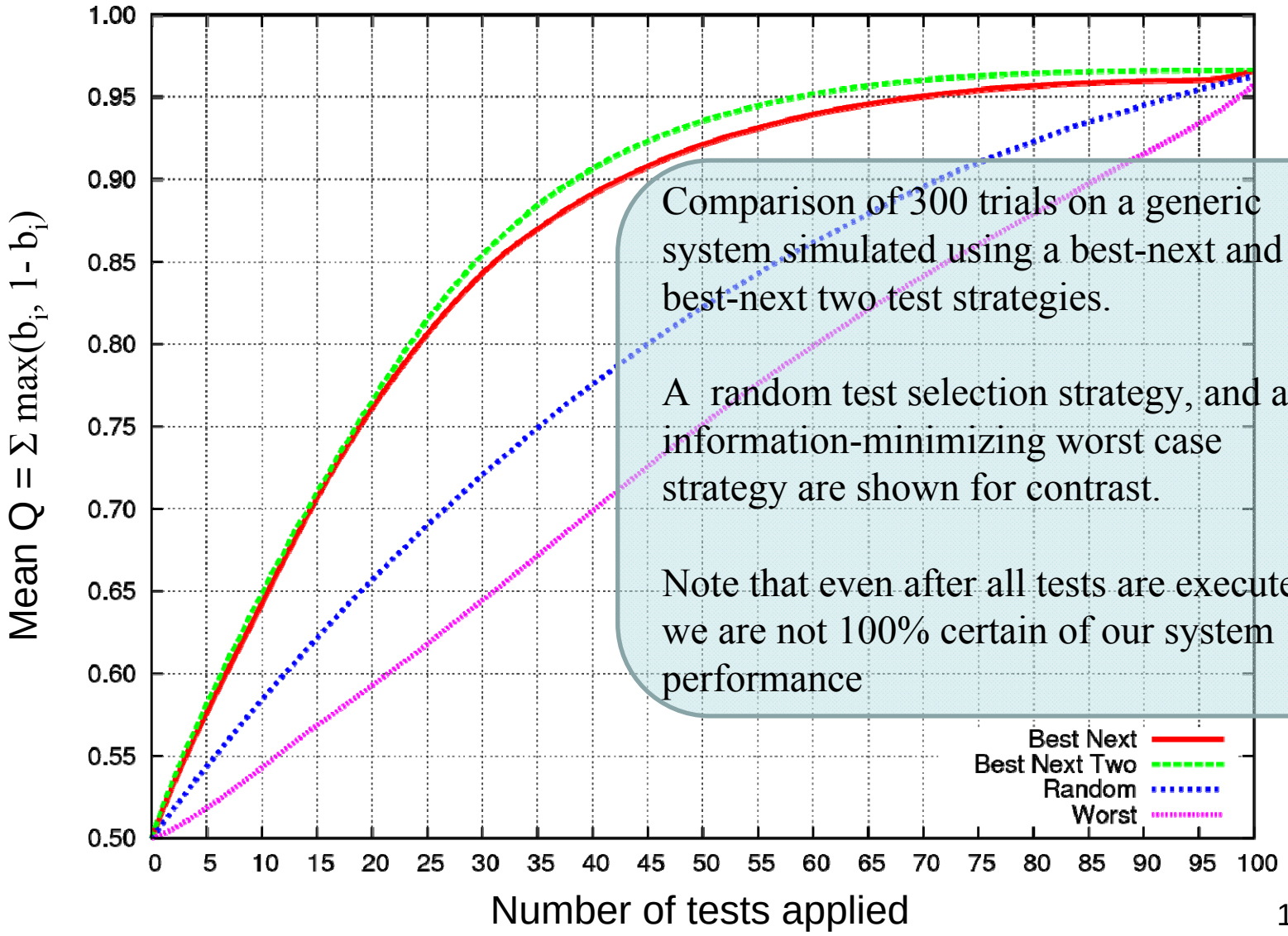
Model implementation

- A prototype decision aid was crafted from this mathematical model for desktop simulation
  - Development in platform-independent, compact Java
  - Configuration files and simulation output maintained as well-formed XML files for experimentation and analysis
- Within the simulated system, zero or more defects can be planted within the set of modules
  - With planted defects, we can examine the best test sequences to isolate faults in a system down for repair
  - With zero defect runs, we can examine the information return on a test suite for use in regression or post-maintenance analysis to verify that the system is mission capable

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

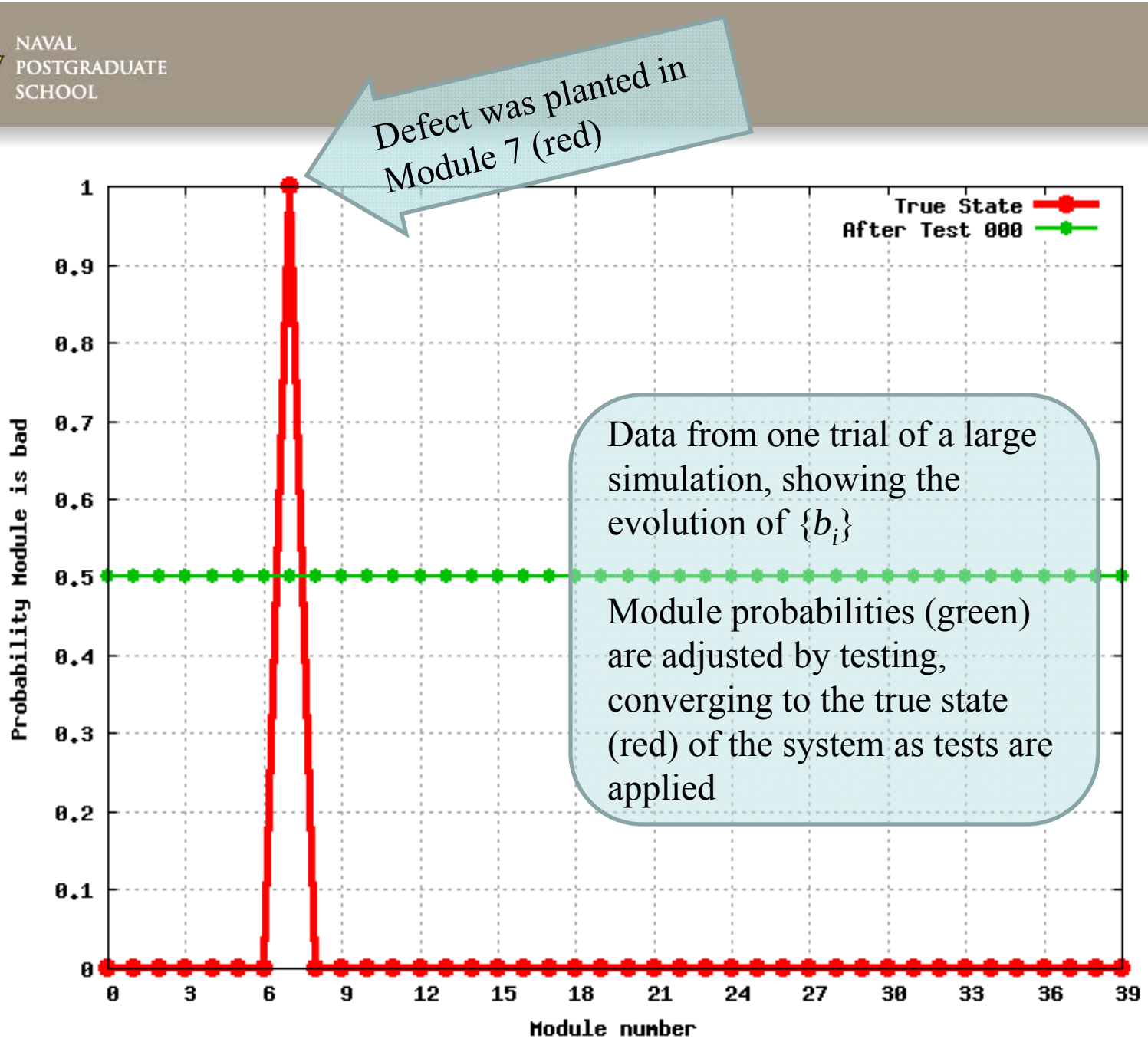
Preliminary results



Comparison of 300 trials on a generic system simulated using a best-next and best-next two test strategies.

A random test selection strategy, and an information-minimizing worst case strategy are shown for contrast.

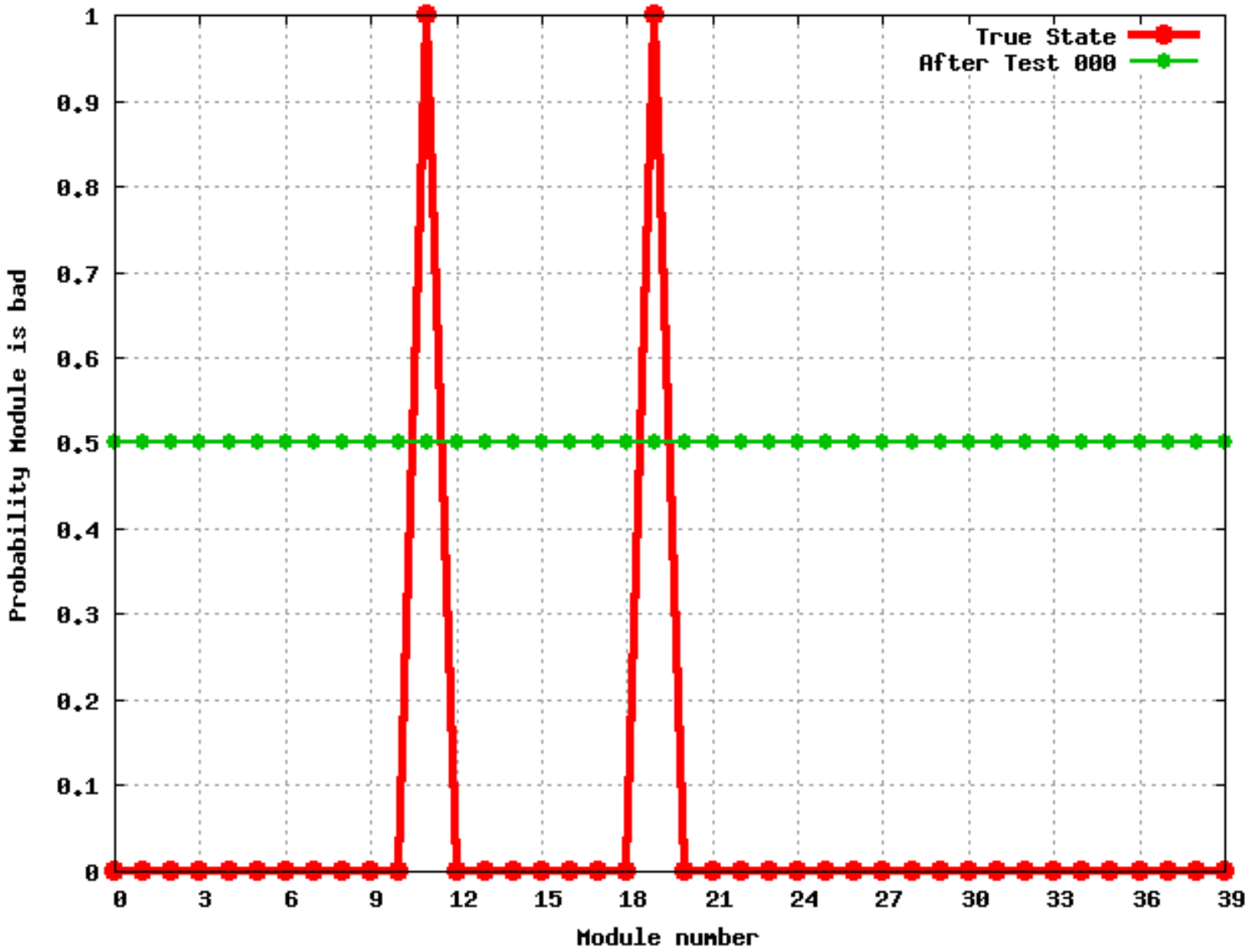
Note that even after all tests are executed we are not 100% certain of our system performance

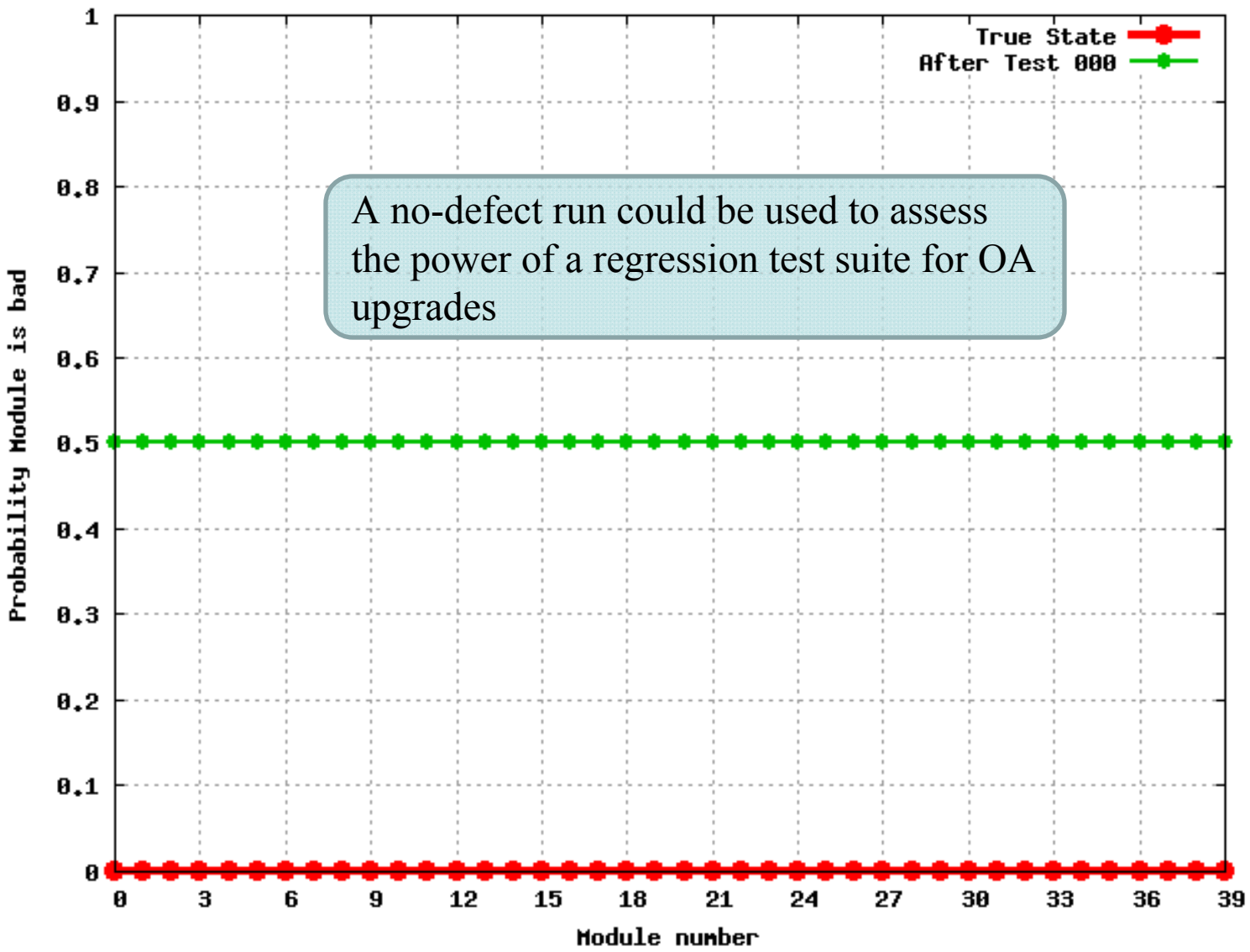


Data from one trial of a large simulation, showing the evolution of $\{b_i\}$

Module probabilities (green) are adjusted by testing, converging to the true state (red) of the system as tests are applied

Defects planted in both Module 11 and Module 19





A no-defect run could be used to assess the power of a regression test suite for OA upgrades

But, what does it all mean?

- *Effective, cost-efficient testing is critical to the long-term success of Open Architecture*

- This model and prototype decision aid provide a rigorous yet tractable way ahead to improve system testing
  - And, to better understand and document the system and component interdependencies across the enterprise

- 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

Future work

- To further refine the current prototype into an operational capability will require time and effort, notionally:
  - Three months to work with subject matter experts in simulating real-world cases from the OA community
  - Six months to improve the user interface and tune the system specification software to meet operational requirements
  - Three months for user training and documentation updates
  - ➢ *This schedule only works if we have the OA test cases*