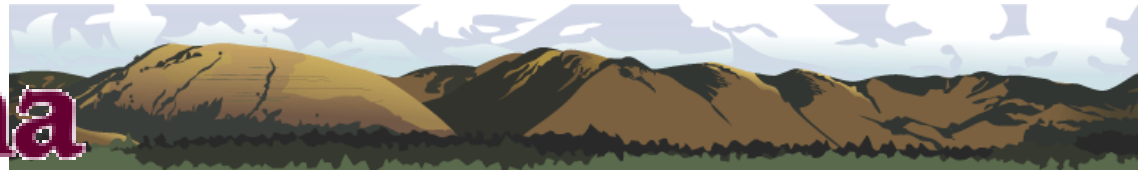
Verification and Validation in Model-Driven Development

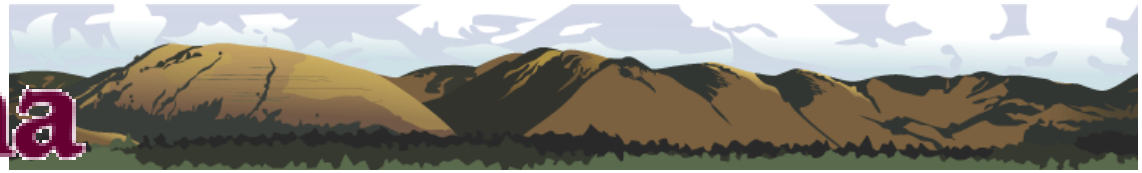
Dr. Joel Henry

May 1, 2008



Overview

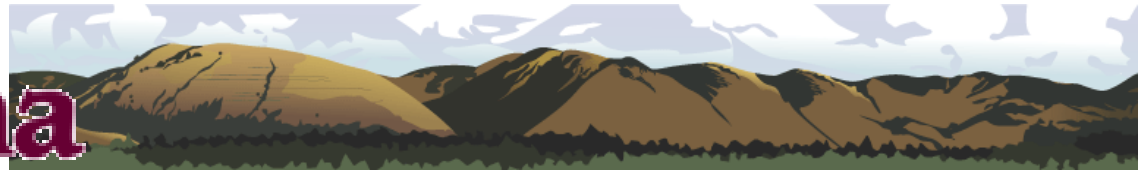
- Background
- Testing approaches
  - Requirements driven
  - Structure driven
- Matlab/Simulink testing
- xUML testing
  - Integration into the process
  - Requirements validation
  - Structure verification



UM and Real-time Systems Lab

- Background
  - Started in Fall of 2000
  - Seven years of funding from NASA, DPRA, and SAIC for a staff of 3-8
  - Research, development, and training
- Areas of expertise
  - Testing methods, tools, and analysis
  - Real-time system simulation
  - NP-Hard problems

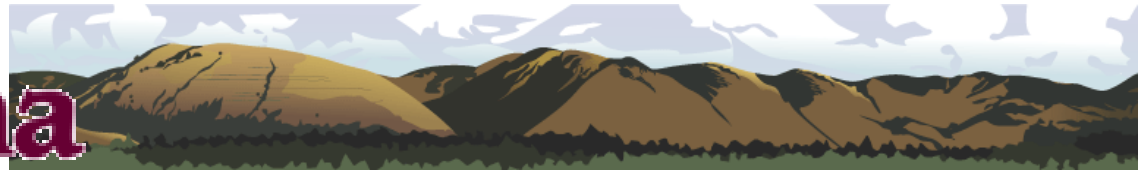
The University of Montana



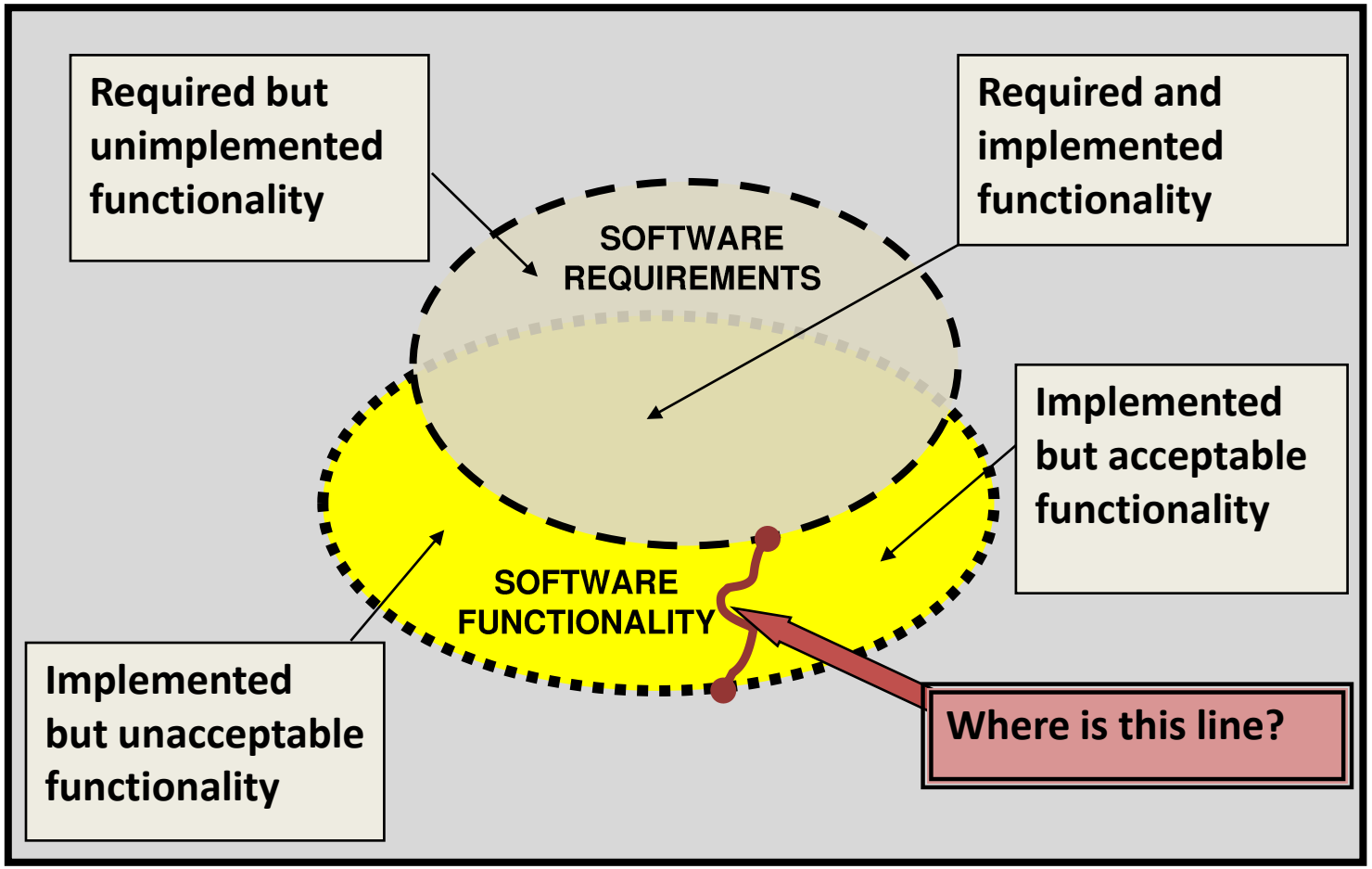
mri technologies www.mricompany.com

Previous Projects

- MatrixX
  - Methods for testing models and autocode
  - Tools and training for NASA and subcontractors
- Matlab
  - Methods for testing models and autocode
  - Constraint determination in models and code
- xUML
  - Testing methods, tools, and analysis
  - Application of methods from other MDA projects

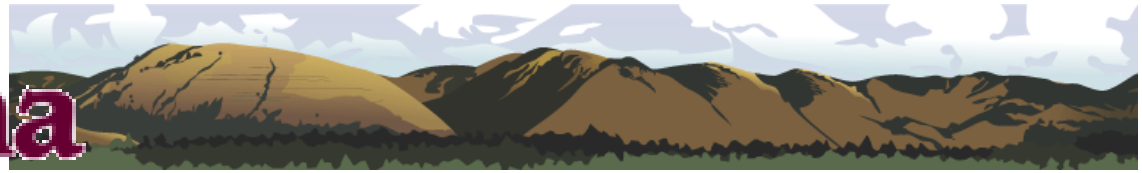


Requirements vs. Functionality



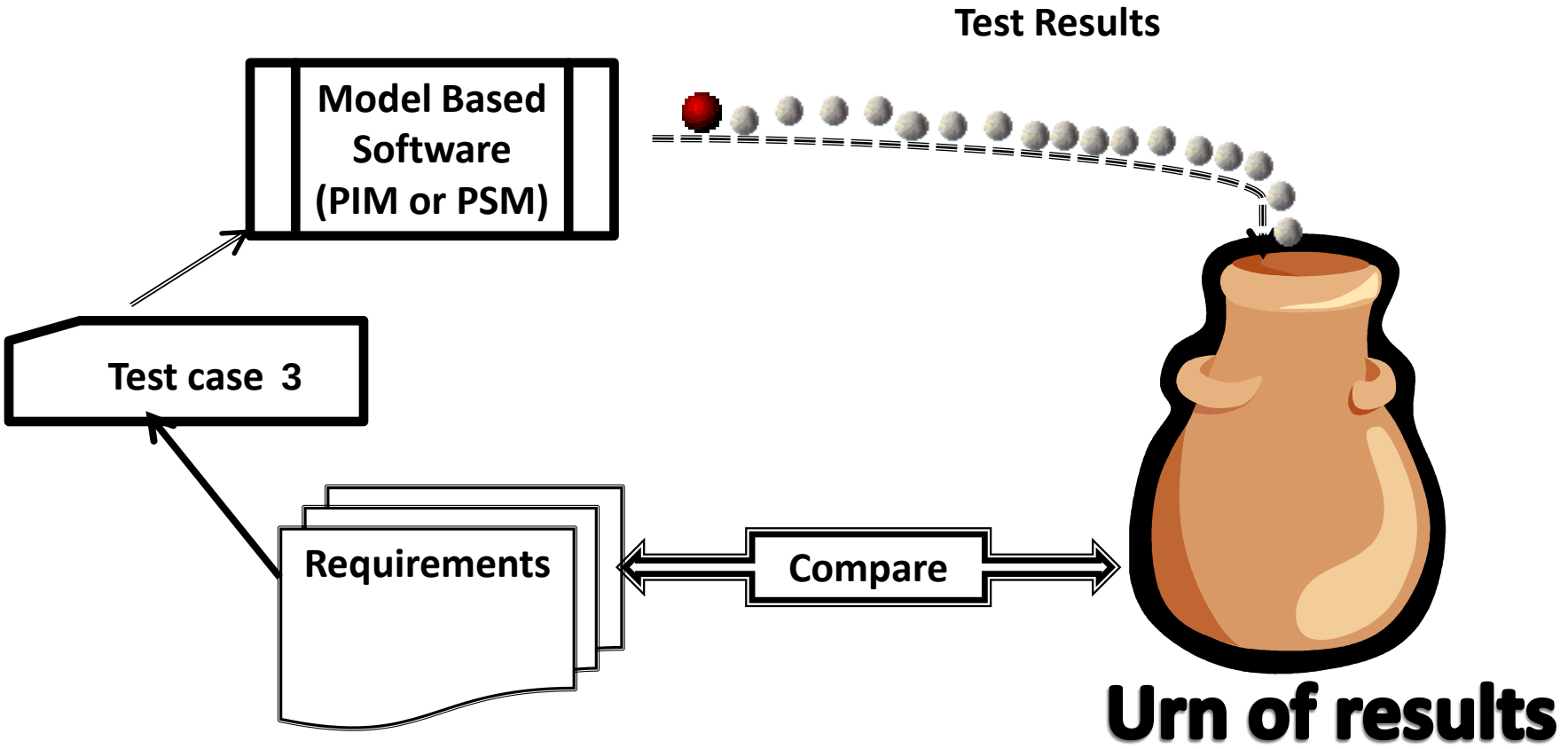
Requirements Driven Testing

- What does this mean?
  - Software and system functionality drives testing
  - Use knowledge of design to generate tests
- Why do this beyond black box testing?
  - Monitor software functionality while testing
  - Detect defects and locations/causes of defects
- What does this require?
  - Knowledge of software and system
  - Tools to generate and configure test data
  - Ability to identify defects in test results



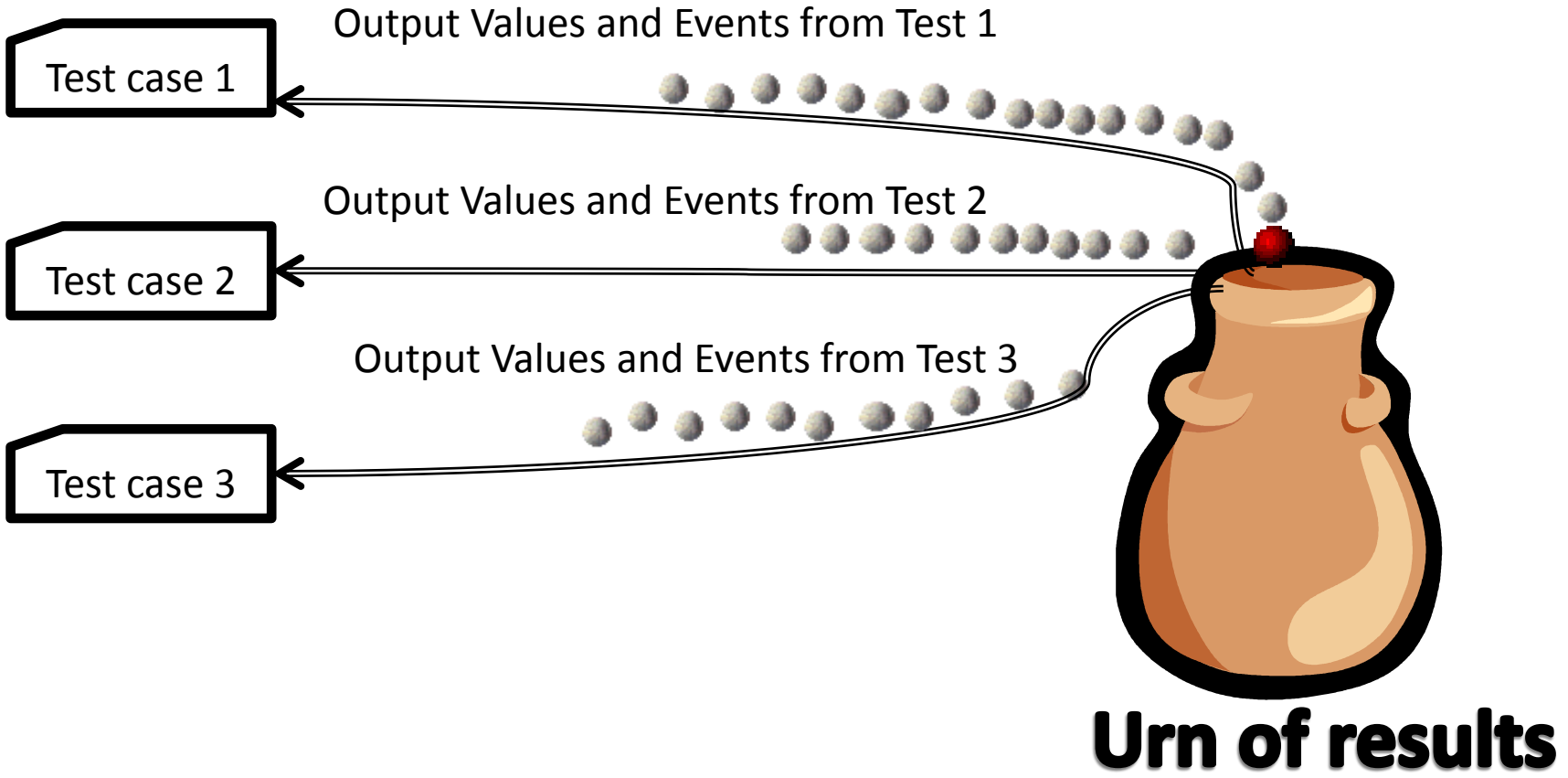
Requirements Driven Testing

Ball and Urn Analogy



Requirements Driven Testing Analysis

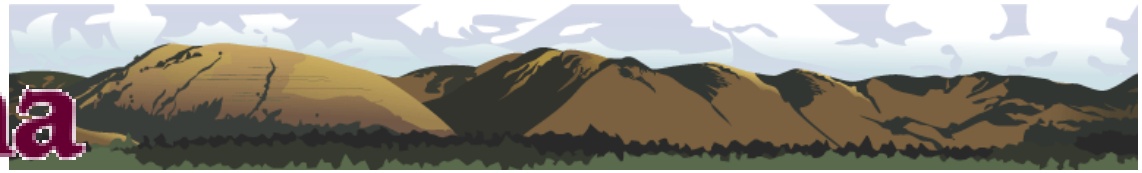
Ball and Urn Analogy



Requirements Driven Testing

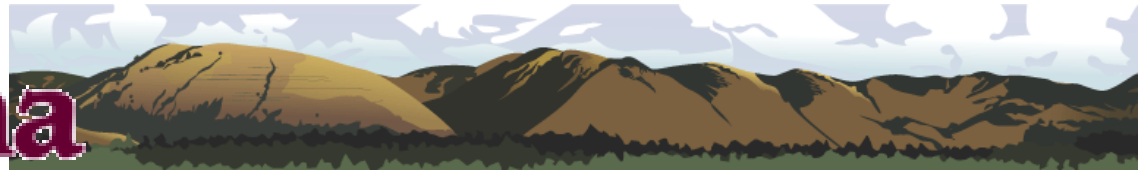
Ball and Urn Analogy

- What does this mean?
  - Test cases for specific software functionality
    - Likely to find defects
    - Critical for safety, reliability, success, etc.
- What to test?
  - Values around critical points
  - Large number of input value combinations
  - Sufficient coverage of each equivalence class
- What are the results?
  - Test coverage for input ranges and combinations
  - Output range coverage, reliability, MTTF, etc.



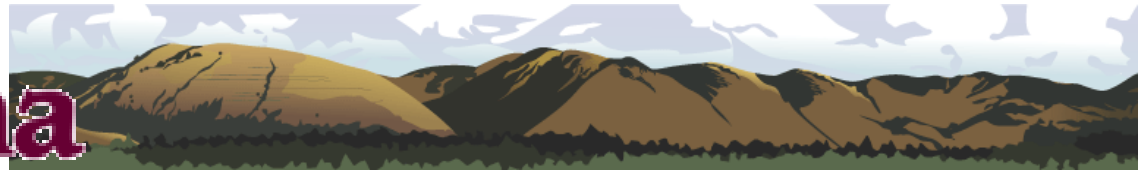
Structure Driven Testing

- What does this mean?
  - Architecture of the model drives test cases
  - White box testing at multiple levels of abstraction
- Why do this testing?
  - Focus on critical design areas
  - Capitalize on tool capability
  - Detect defects and locations/causes of defects
- What does this require?
  - Knowledge software and system
  - Knowledge model architecture
  - Ability to create test data-expected result pairs



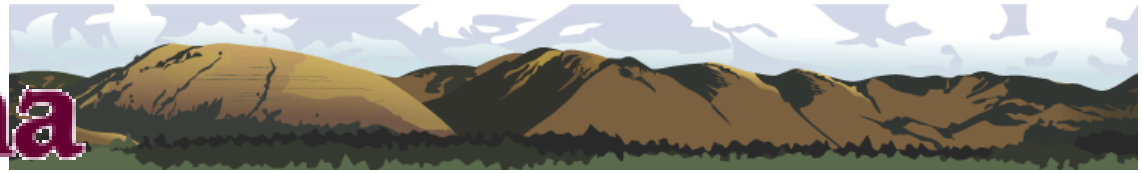
Structure Driven Testing

- T-VEC
  - Model is read, converted, and then analyzed
  - Analysis generates test cases that are executed
- LDRA
  - Tbrun Performs unit testing on C and C++
  - Code analysis, test harness, then tests executed
  - Testbed calculates test coverage metrics
- xUML via Kennedy Carter testing tools
  - Research tools, not readily available and not ready for commercial use
  - Support for testing within Kennedy Carter tools

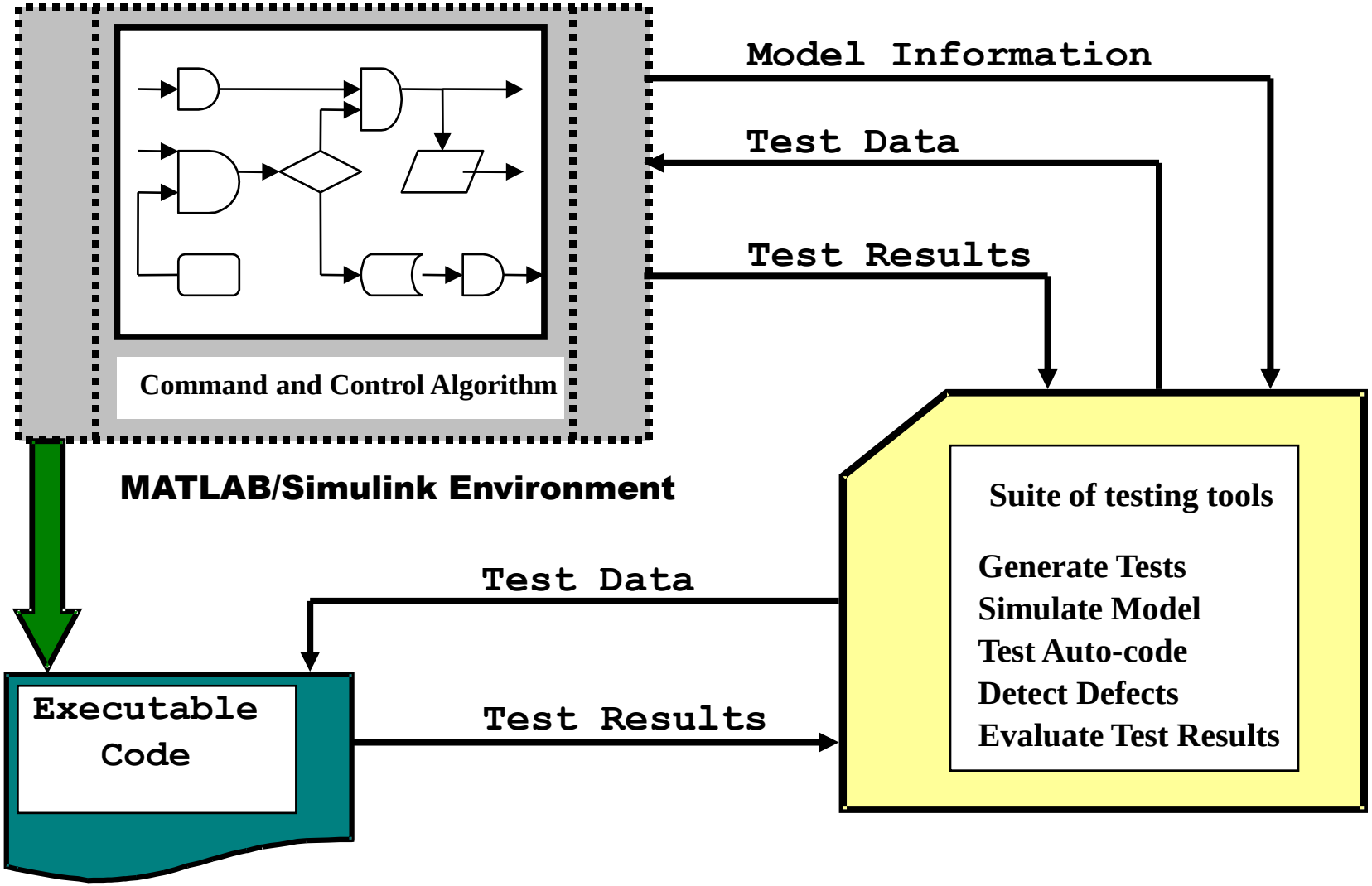


Testing Solution

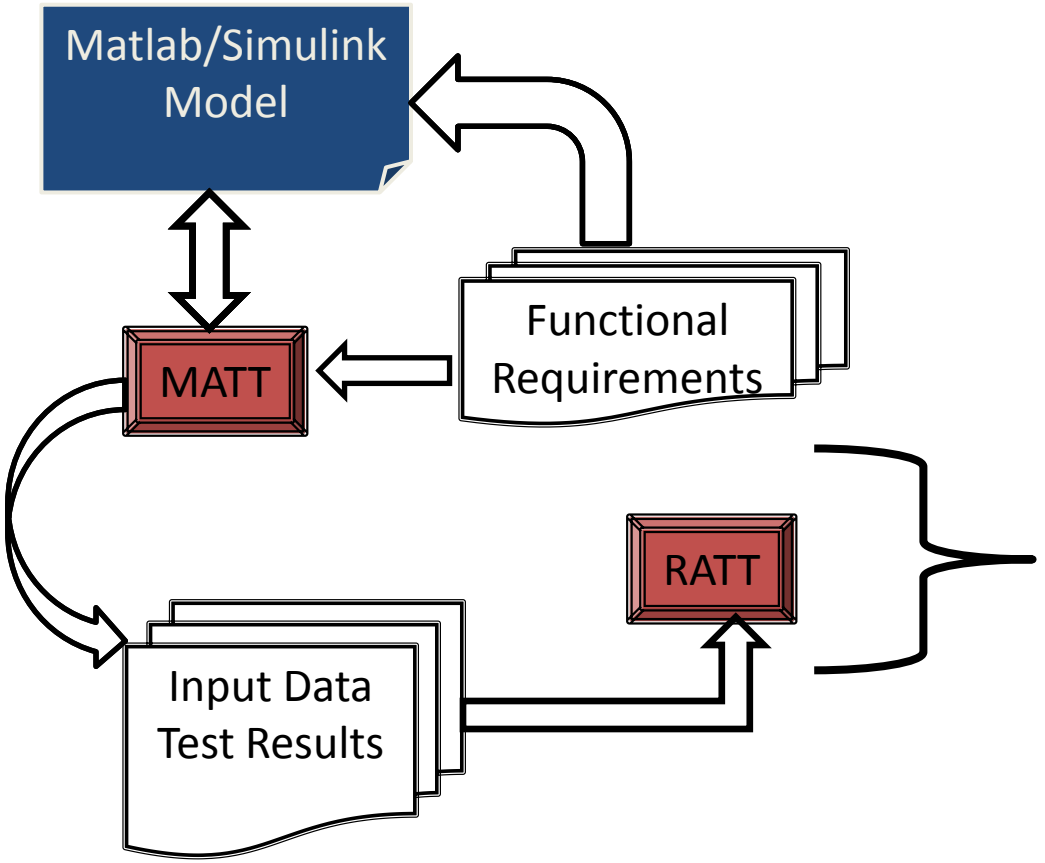
- Innovative, reusable, long-term testing environment
- Requirements and structure driven testing
- Implement without change to models
- Defect detection, test case re-execution, testing measurement
- Test model and translated model with same tests
- Leverage past success with Matlab/Simulink



Matlab/Simulink Testing Solution



Matlab/Simulink Test Tools

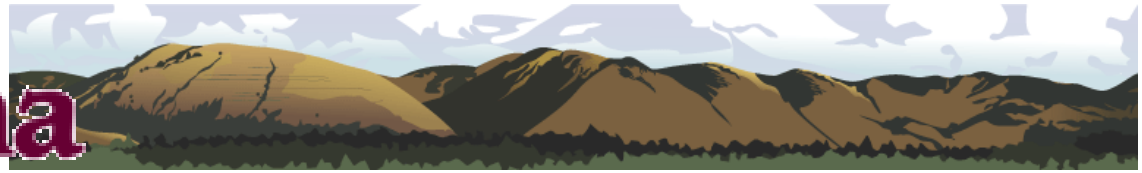


Test results:

Input range coverage
Output range coverage
Probability of exception
Exception prone ranges
MTTF
Equivalence class analysis
Testing recommendations

Testing Problems

- Requirements
  - Input file/matrix
  - Output file/matrix
  - Sample time – variable or set frequency
- Variable Range
  - Input variable – min, max, and accuracy
  - Output variable – min, max, and accuracy
- Exceptions
  - Identification
  - Tracing

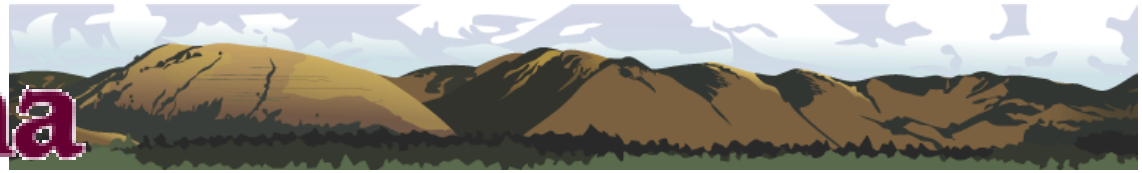


Test Execution

- Create test data
  - Functions, freehand, imported
- Execute tests
  - Model - wrap model, simulate, unwrap
  - Translated model – input data, configure, run
- Capture results
  - Input, states, output
- Detect exceptions
  - Configurable exception detection
- Analyze across multiple tests

Defect Detection

- Simple value range detection
- Percent change
  - Allows exception detection if the output value changes more than a specified percent over a specified number of steps
- Absolute change
  - Allows exception detection if the output value changes more than a specified amount over a specified number of steps



Defect Detection

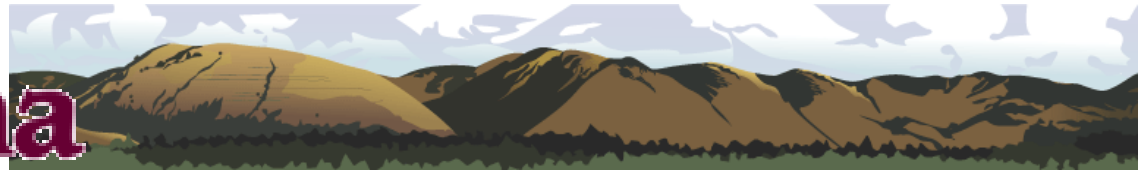
- Advanced Exceptions
  - Combinations of exception definitions
  - Disjoint ranges
    - Create exception definitions by time range
    - Combinatorial definitions based on multiple exception definitions
  - Overall system reliability
    - Scenario based reliability per major function
    - Overall reliability combining scenario reliability

Advanced Methods

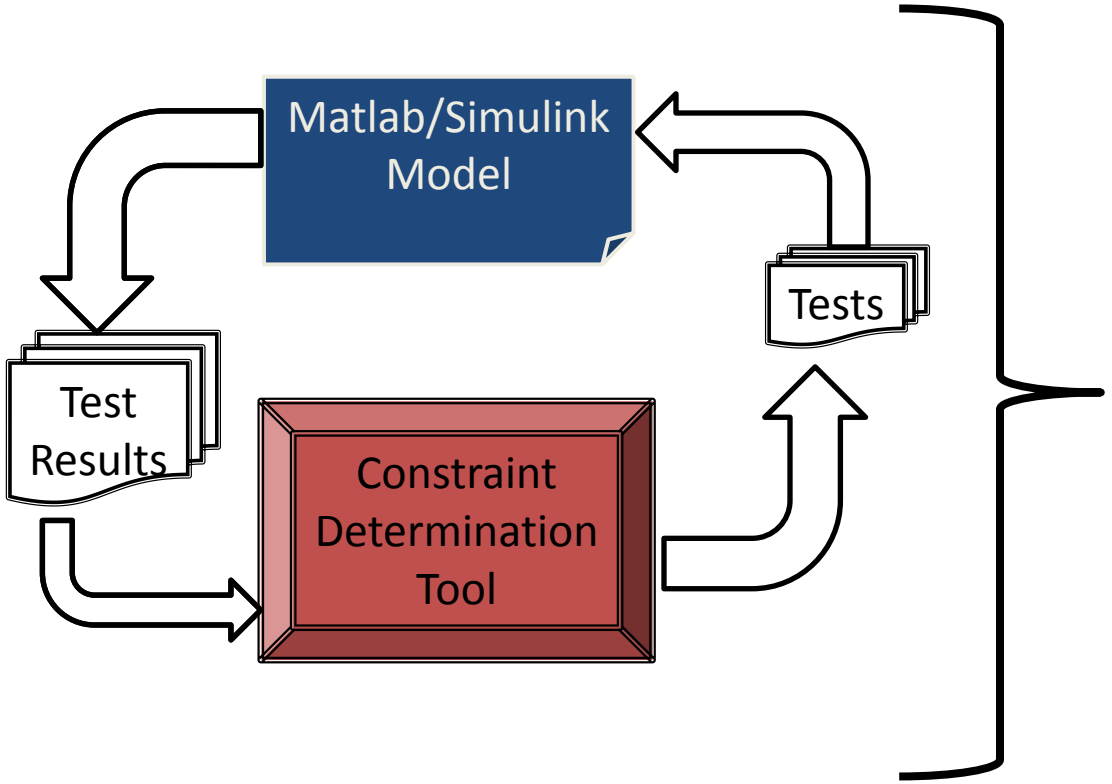
- Apply to other types of MDA
- Tackle critical real-time systems problems
  - Integration of approaches
  - Minimum and Maximum output values for scenarios, units, subsystems

Constraint Determination

- Search based method to find min or max values for a Simulink outport
- Two methods
  - Genetic algorithm
  - Combination of Simplex and Simulated Annealing
- Research tool (needs an interface, help, etc.)
- International acceptance (paper invited to conference in Oxford UK in 2007)



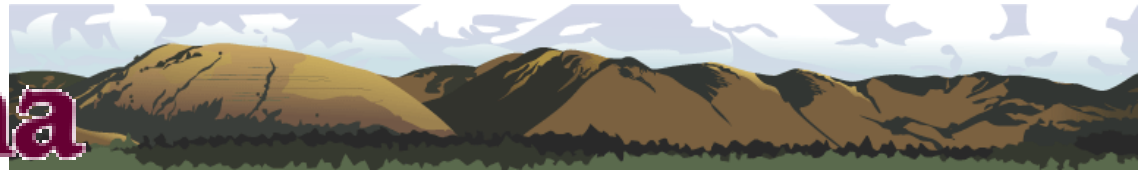
Constraint Determination



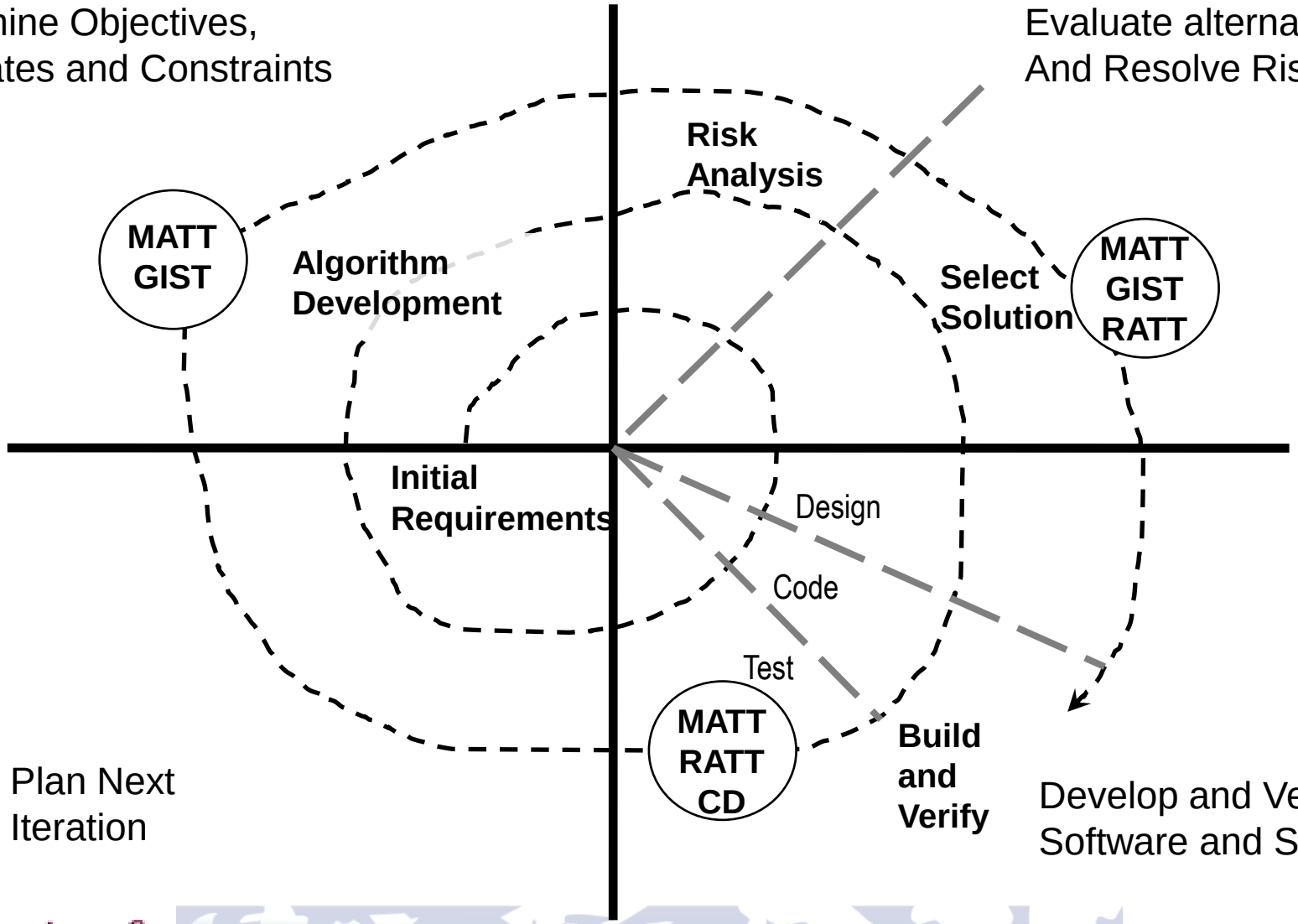
Test results:
Global Minimum
Global Maximum
Input values associated with Min and Max

Integration of Tools and Methods

- Where to place the tools?
- How to use the tools effectively?
- What to do with results?
- How to gain acceptance?



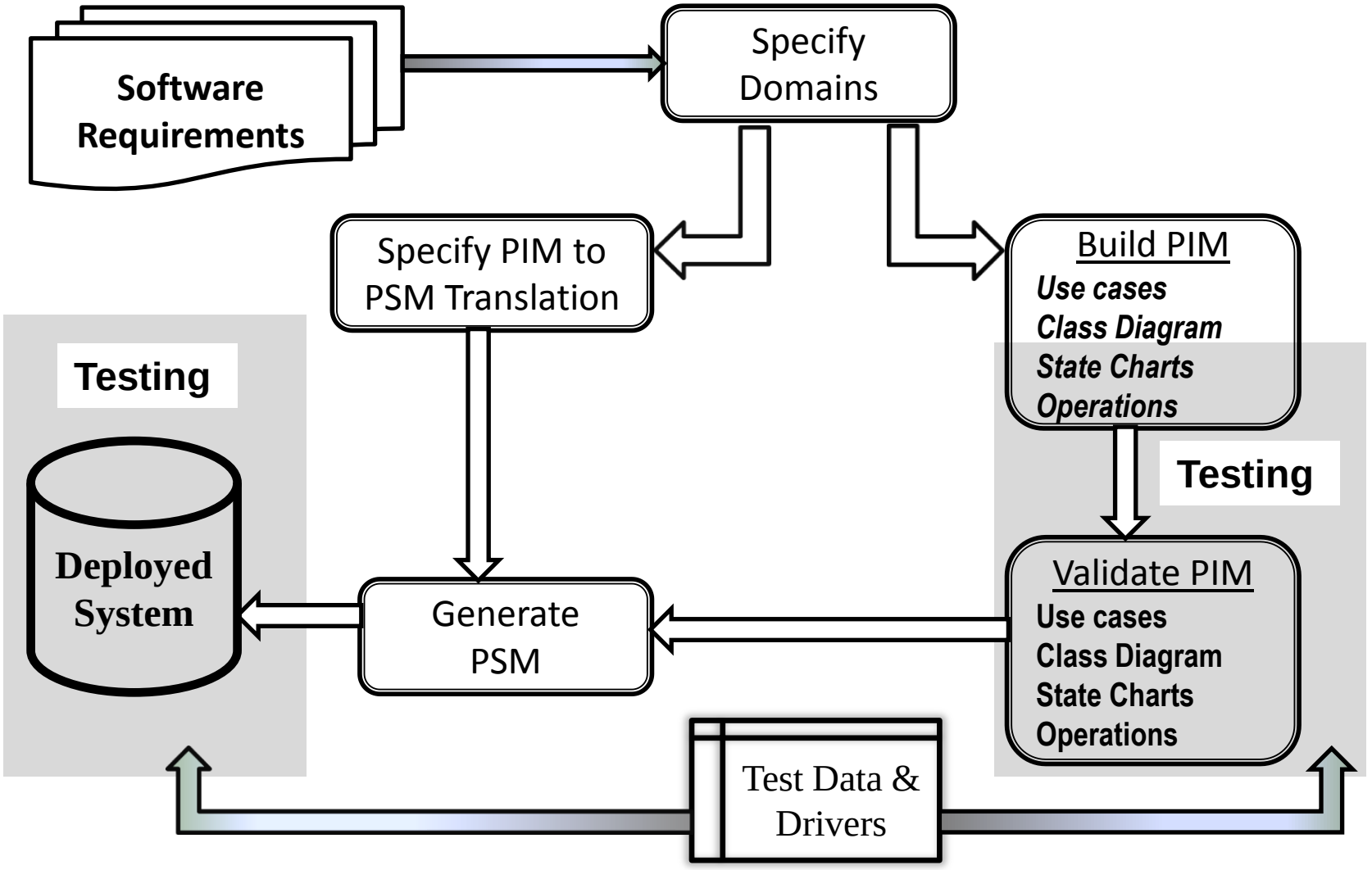
MDA Testing & Tool Usage

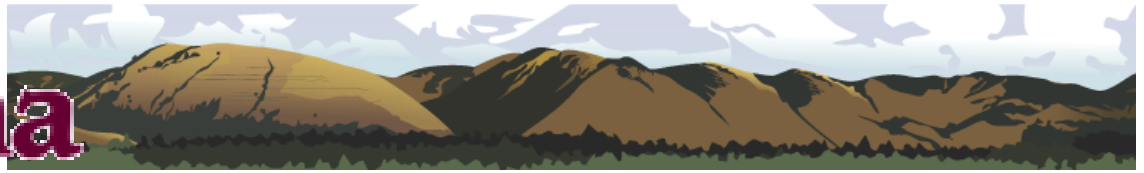


xUML Testing Goals

- Apply Matlab/Simulink techniques to xUML
- Innovative, reusable, long-term testing environment
- Requirements and structure driven testing
- Implemented without change to xUML models
- Defect detection, test case re-execution, testing measurement
- Testing on PIM and PSM with no extra effort

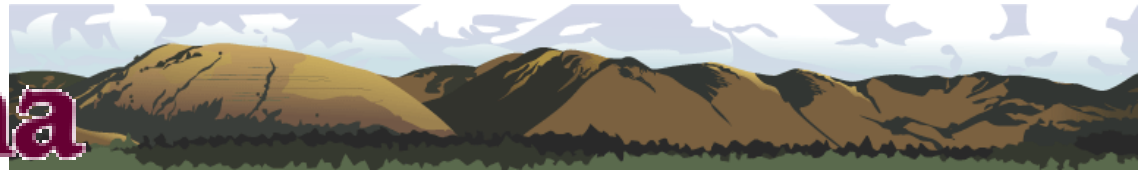
xUML Testing Placement



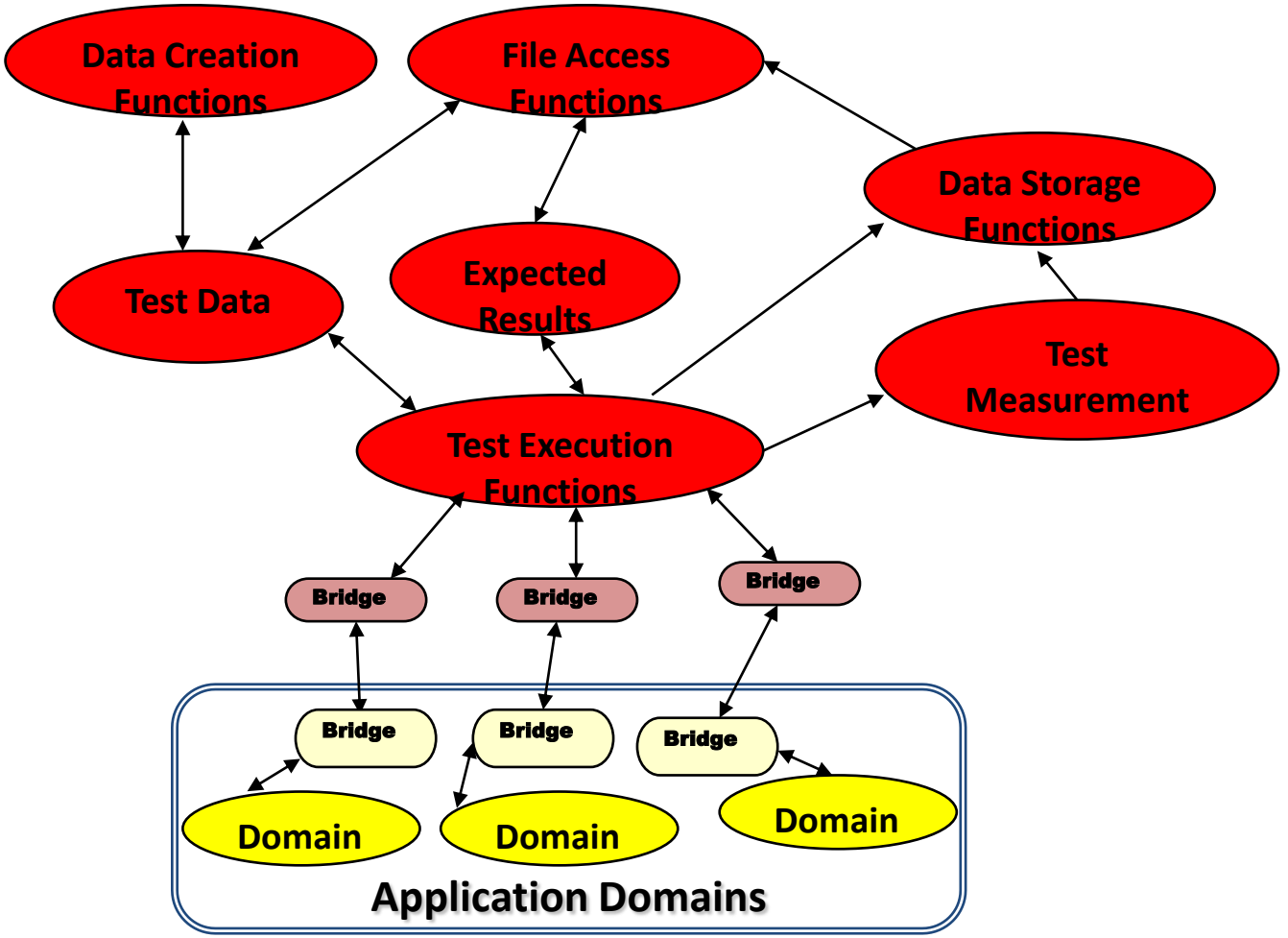


xUML Testing Approach

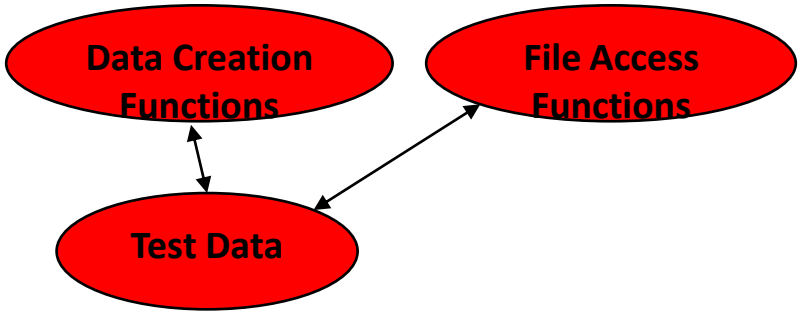
- Build a set of testing domains independent of application domains
- Implement reusable testing methods within testing domains
- Encapsulate application-testing domain coupling in bridges
- Include the ability to automate testing
- New applications require only new bridges
- Suitable for PIM and PSM testing



MRI xUML Testing Approach

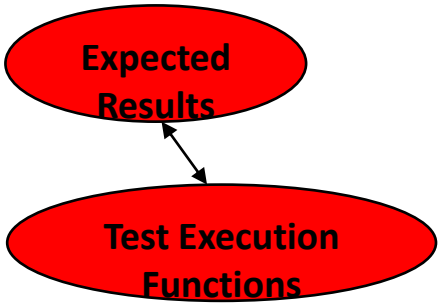


xUML Testing Domains



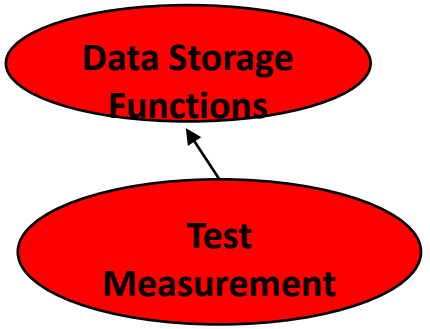
- Data Creation
  - User configurable functions to generate test data
- File Access Functions
  - Functions to retrieve data from files, DB, spreadsheets
- Test Data
  - Data format conversion functions

xUML Testing Domains



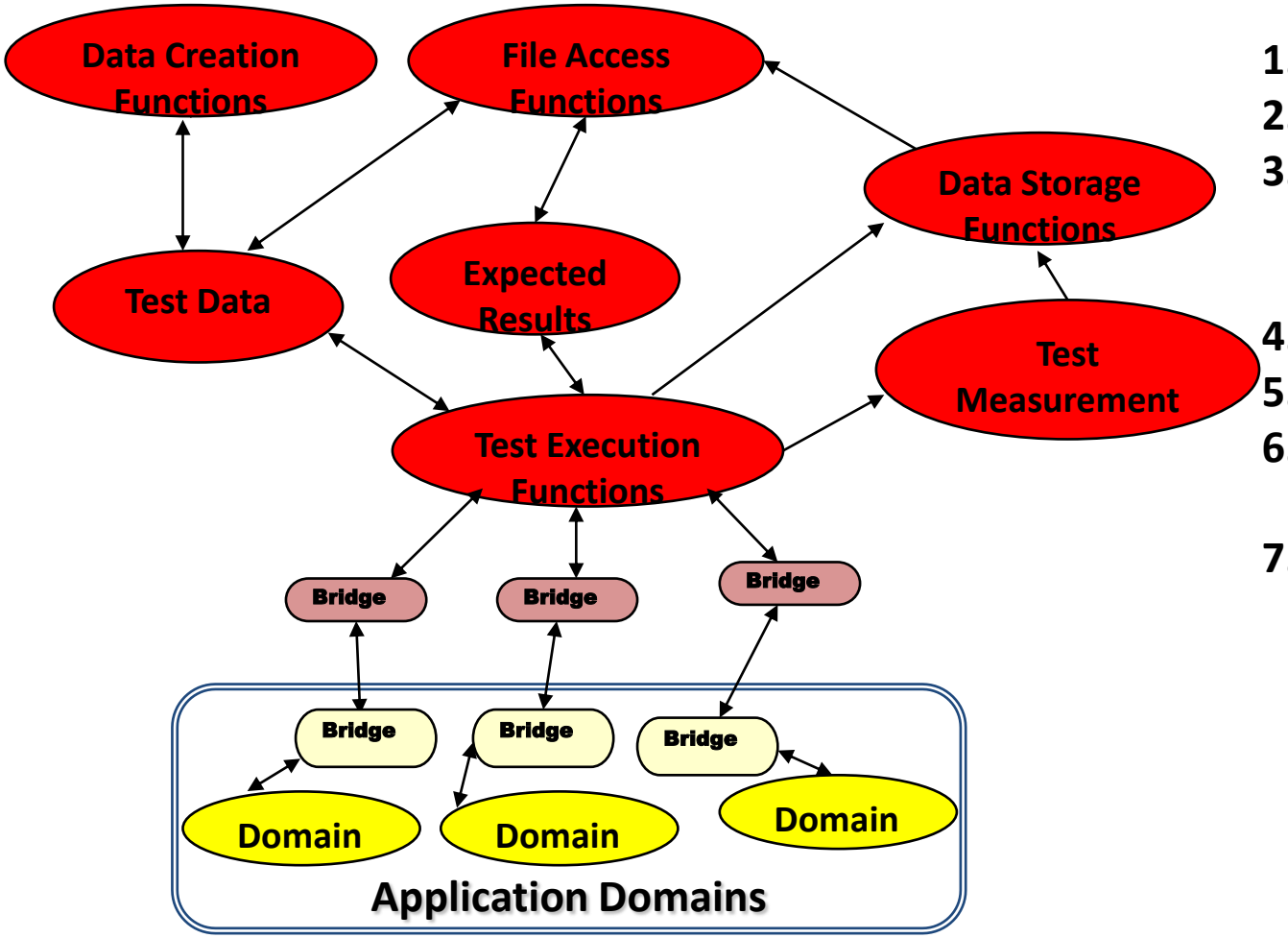
- Expected Results
  - Organize output data into a consistent format
- Test Execution Functions
  - Offer a set of test types that can be executed on any application domain (through Bridges)

xUML Testing Domains



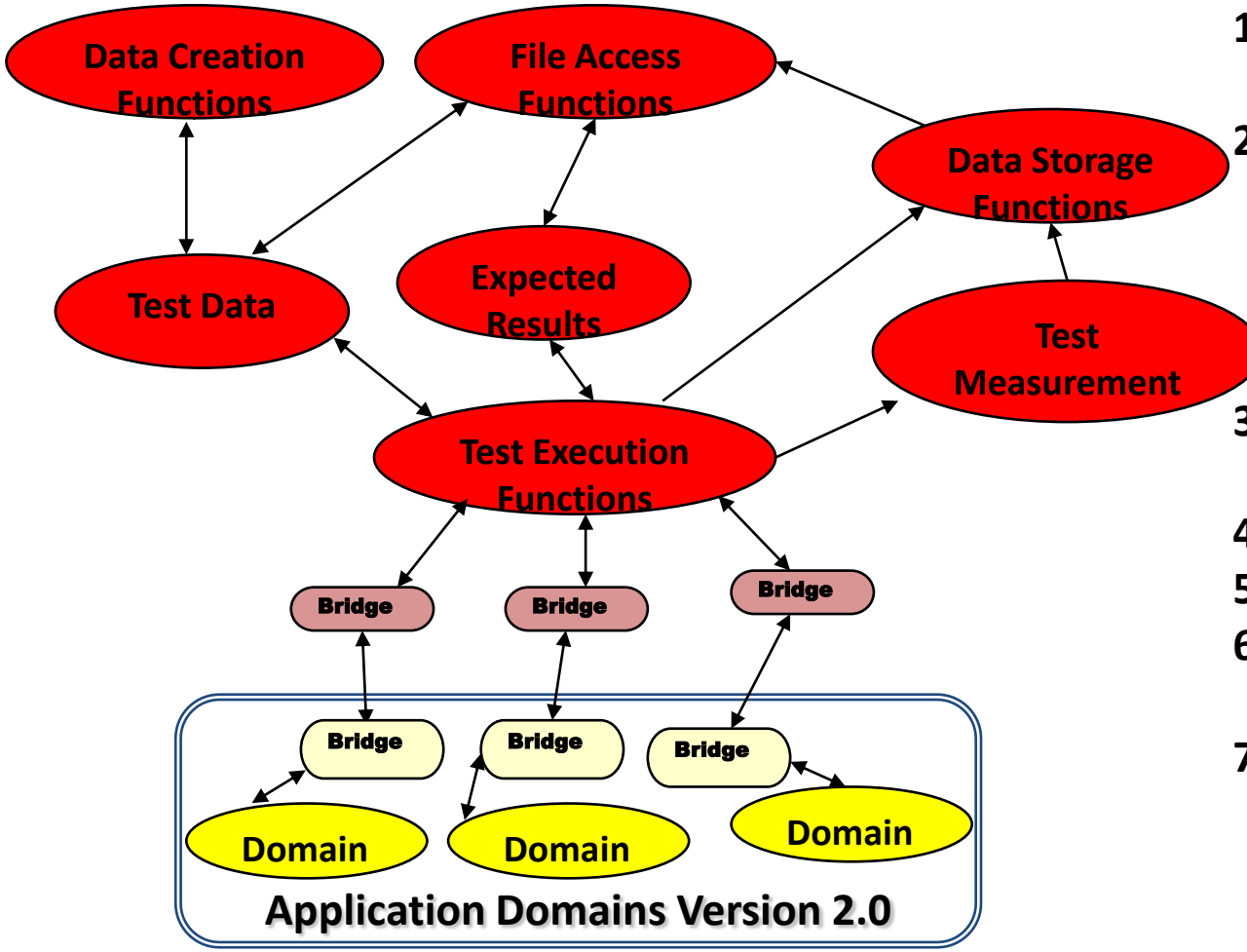
- Data Storage Functions
  - Output the test data, results, expected results, and exceptions in a consistent format to multiple sinks
- Test Measurements
  - Functions that perform a variety of test measurements

xUML Development Testing

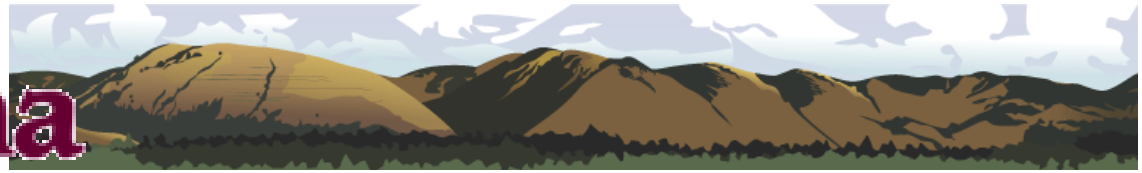


1. Read input data
2. Read expected results
3. Configure the Input Data (initialize, call-data pairs)
4. Execute the tests
5. Capture test results
6. Calculate test measurements
7. Store the test results

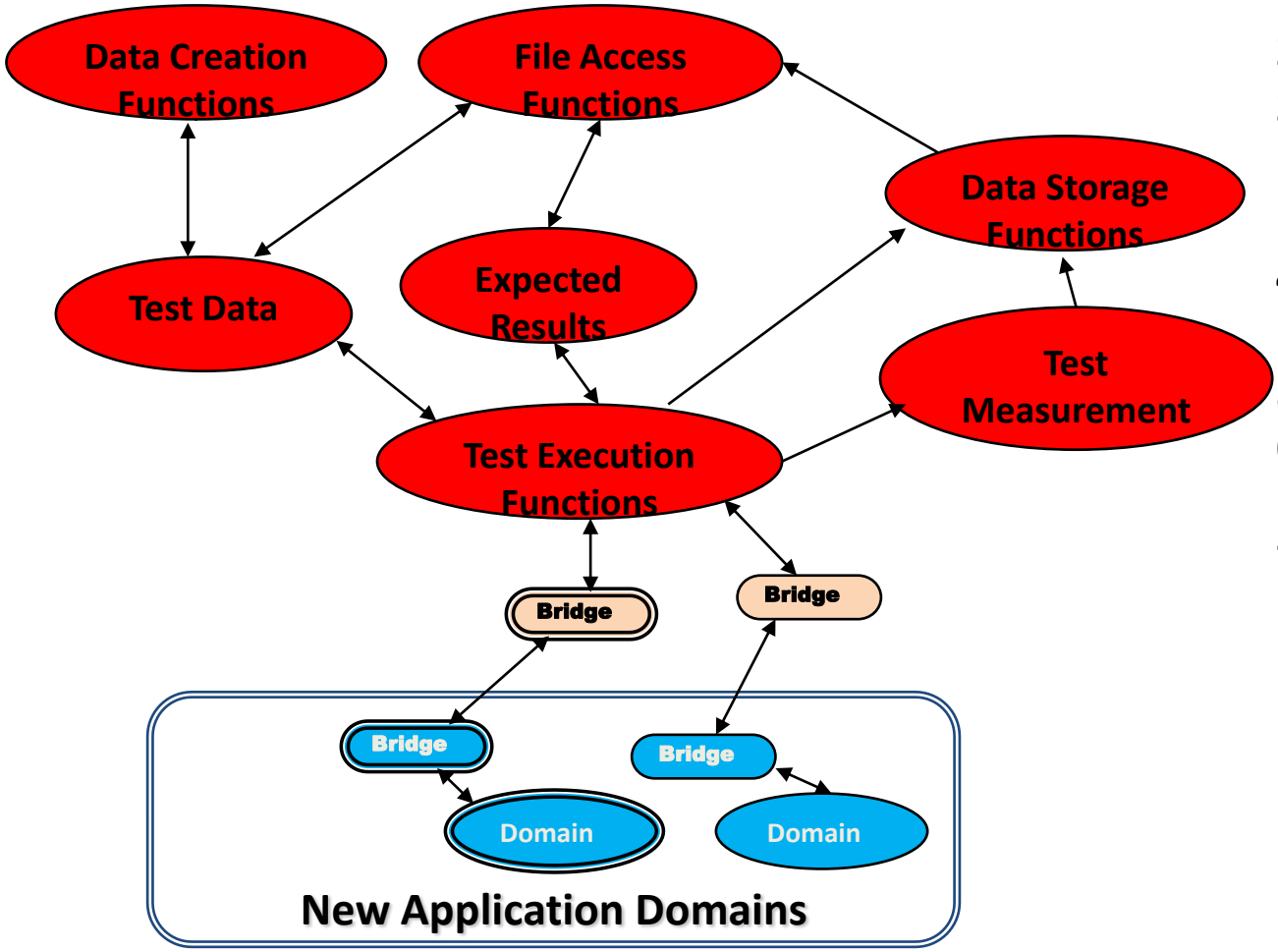
xUML Maintenance Testing



1. Read input data from data gathered during use
2. Read expected results
   a) *Actual results from deployed system* ***OR***
   b) *Expected results for new functionality*
3. Configure the Input Data (initialize, call-data pairs)
4. Execute the tests
5. Capture test results
6. Calculate test measurements
7. Store the test results



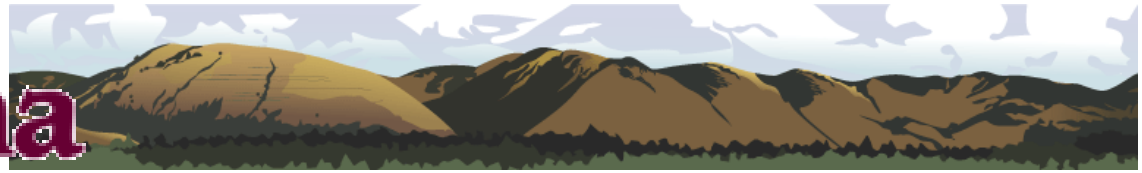
Reuse xUML Testing Domains



1. Read input data
2. Read expected results
3. Configure the Input Data (initialize, call-data pairs)
4. Execute the tests (using new Test Bridges)
5. Capture test results
6. Calculate test measurements
7. Store the test results

xUML Testing Process

- Development organization provides
  - Application bridge specification
  - Requirements documents
- SQA or IV & V build testing domains, bridges
- Test application domains, bridges, subsystems, and finally entire systems
- Tests applicable for PIMs and PSMs
- *Approach has been built and utilized*



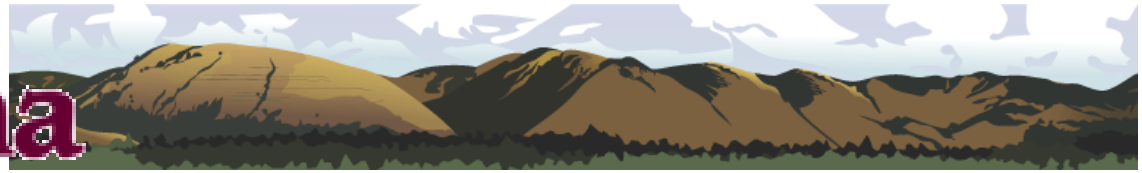
Questions….?

This research done with:

MRI Technologies

Conference support from:

The University of Montana



Acronyms

- MDA – model driven application
- PIM – platform independent model
- PSM – platform specfic model
- xUML – executable unified modeling language
- MTTF – mean time to failure
- MATT – Matlab automated testing tool
- RATT – reliability automated testing tool
- GIST – graphical input specification tool