Key UMass Amherst Resources for SERC Collaboration

Leon J. Osterweil (ljo@cs.umass.edu)
Lori A. Clarke (clarke@cs.umass.edu)

Lab. For Adv. SW Engineering Research
(LASER http://laser.cs.umass.edu)

Presentation to SERC Research Review
Malvern, PA
October 16, 2009

Microprocess:
A “Horizonal Technology Cut”

Process is a central issue in system engineering

- Goal: systems that are fast, agile, safe, effective,…
- Approach: processes for
  - Building, analyzing, using, evolving, training, …
- Processes specify how systems are
  - Developed, used, evolved, …
  - As collaborations of people, software, devices
- (Development) processes are used to build systems
  - Better systems come from better processes
- (Usage) processes guide how systems are wielded
  - Better processes exploit systems better
- System improvement from Process Improvement

Example: Agile System Evolution

- How to quickly, surely enhance deployed systems?
- Improve their:
  - Speed, functionality, usability, robustness
  - Quickly, correctly, reliably
- Requires processes for:
  - Coordinating development
    - People, tools, management
  - Assuring product qualities
  - Deploying product
  - Training users
- Requires being sure of these processes

A case in point--***Coming up later in this presentation***

- Agile development (e.g. Scrum) can
  - Speed systems to deployment
  - Close system improvement loops fast
- But can it also
  - Allow defects to creep in unnoticed?
  - Render development vulnerable to poor developer performance?
- Process Analysis:
  - Can be used to identify single points of failure leading to development hazards
  - The basis for removing such defects
  - Process Improvement => System improvement

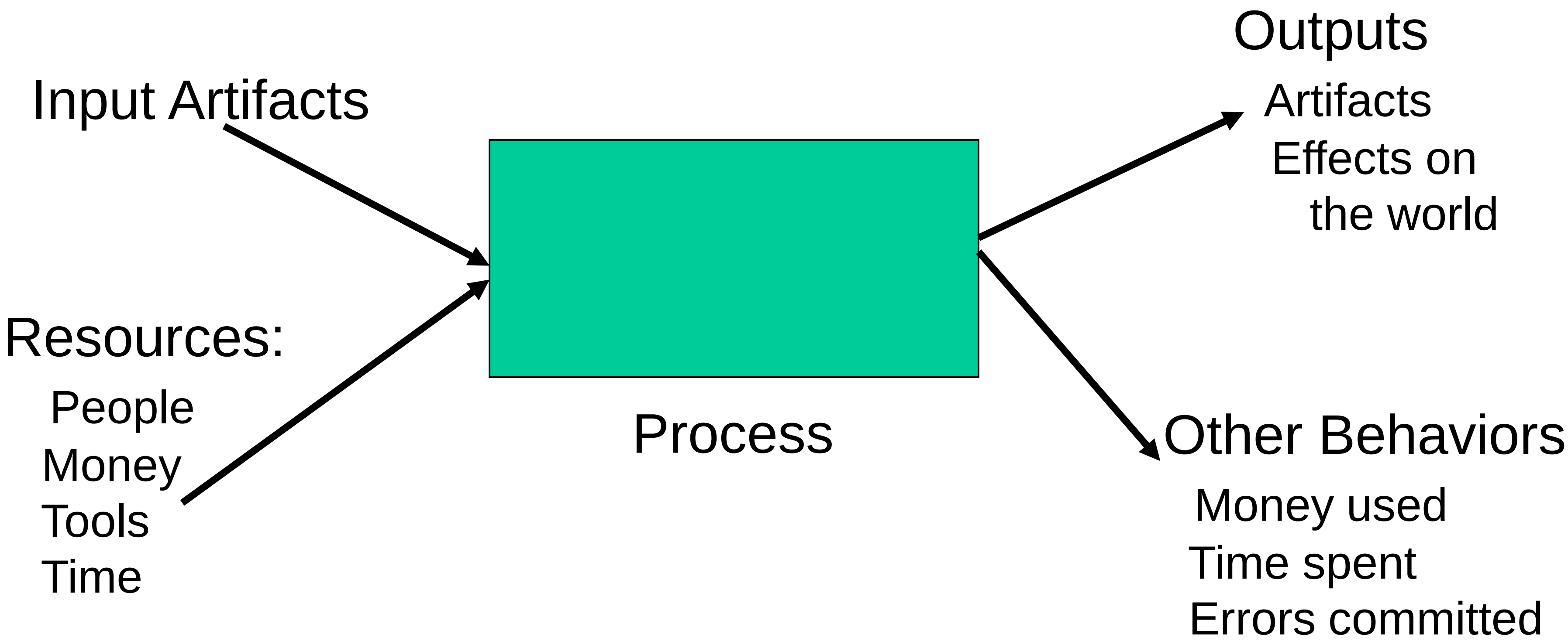
Key UMass Capability:
Technology-Based Continuous Process Improvement

- Process is a central issue in system engineering
  - Collaboration of people, software, devices, etc.
- Process Improvement is a central goal
- UMass concepts, tools, and technologies support process:
  - Definition
  - Analysis/evaluation
  - Education
  - Performance/execution/simulation
  - Evolution

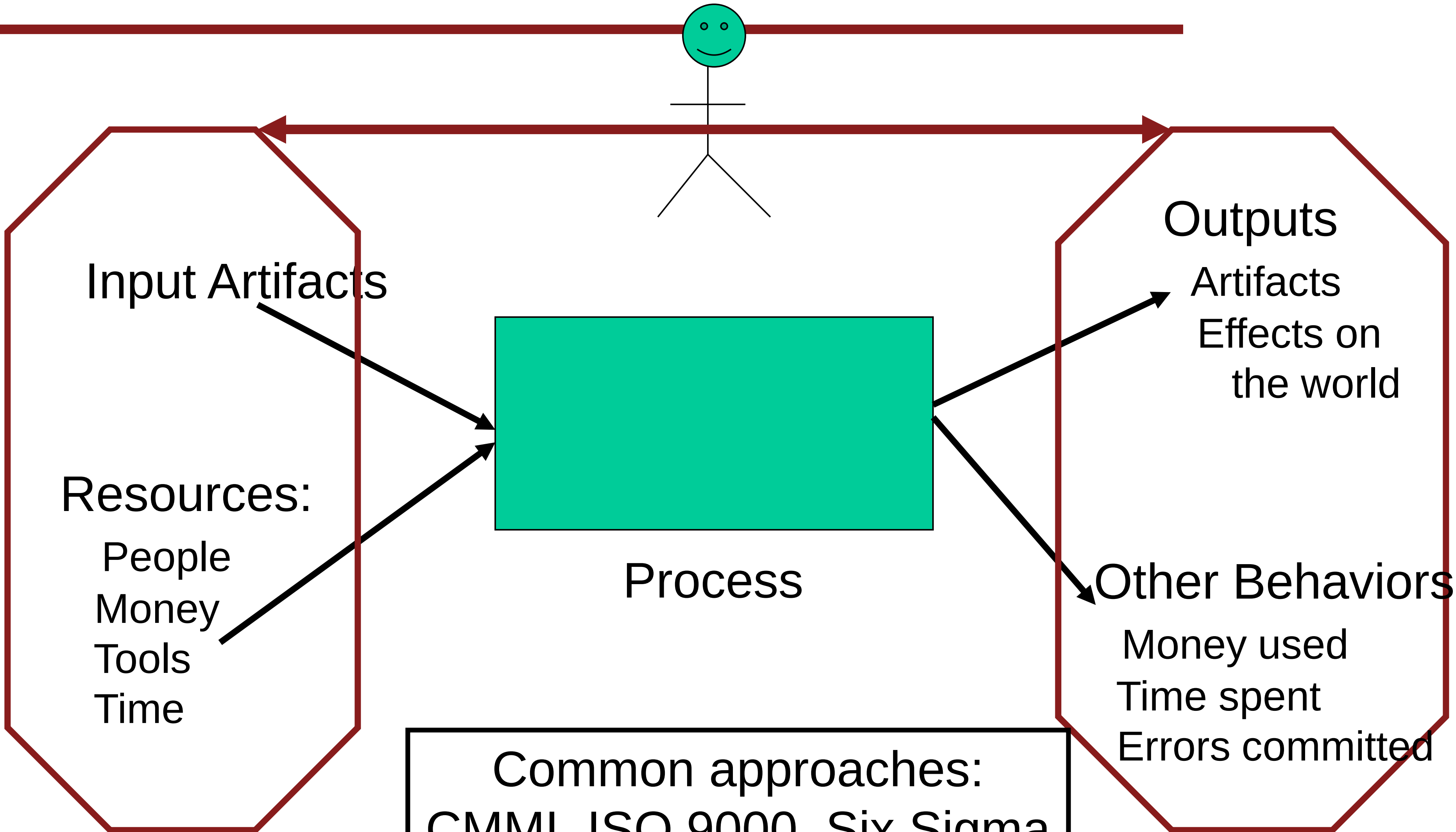
Our approach is based upon

MICROPROCESS
research

Process as Object



Macro-Process Focus

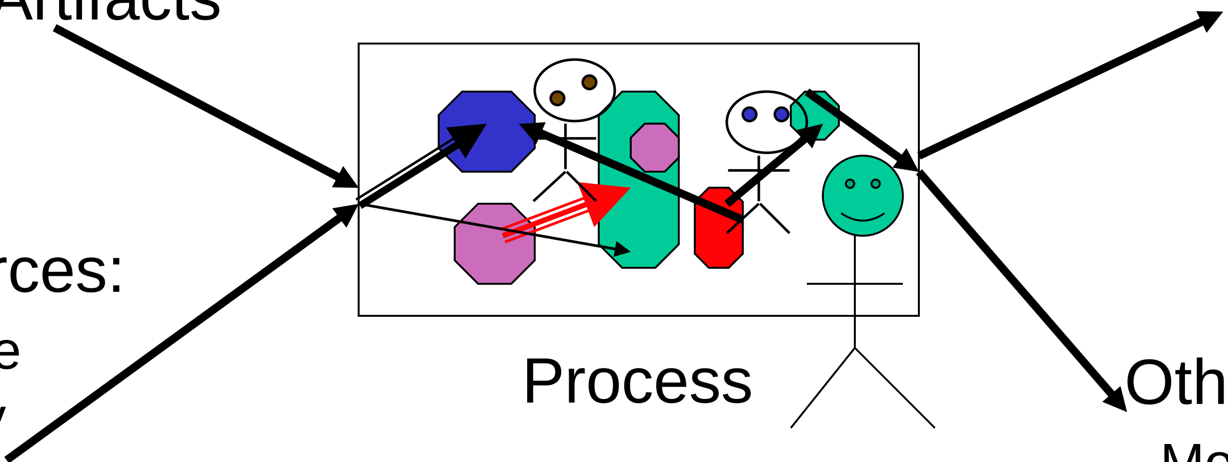


Common approaches:
CMMI, ISO 9000, Six Sigma

Micro-Process Focus

Input Artifacts

Resources:
- People
- Money
- Tools
- Time



Process

Outputs
- Artifacts
- Effects on the world

Other Behaviors
- Money used
- Time spent
- Errors committed

Needed approach: Define, analyze
Automate, precise process definitions

Bridging Micro- and Macro-

- Use details of process model to predict how system attributes and behaviors are produced
- Suggest changes, predict their effects
- Validate changes before they are made

Each has interests in all of these
Each knows it needs the other's approach

What we learn from analogies to other disciplines (e.g. medicine)

- Macro- approach comes first
- Limited success in engineering
- Micro- approach/theory follows
- Facilitates more effective engineering
  - Improved predictability
  - Reduced uncertainty
  - Greater cost effectiveness
  - Better understanding of limitations
  - Fewer surprises

We are here (?)

Time for SERC to take the lead in showing how:

Microprocess technology can transform System Engineering

The Microprocess Vision

- Define processes with a precisely defined executable language
- Analyze processes for defects
  - And fix them to improve them
- Execute, simulate the defined processes
  - To provide user Guidance
- Use them as the basis for education and workforce development

The Microprocess Vision

- Define processes with a precisely defined executable language
- Analyze processes for defects
  - And fix them to improve them
- Execute, simulate the defined processes
  - To provide user Guidance
- Use them as the basis for education and workforce development

Apply this to the many processes implied in the SERC Research Strategy

Little-JIL process language features

- Blends proactive and reactive control
- Coordinates human and automated agents
  - Without favoring either
- Emphasizes exception specification, management
- Facilities for abstraction, scoping, hierarchy
- Artifact flow, resource utilization integrated
- Concurrency, synchronization with message-passing
- Articulate specification of resources
- Semantics for aborting activities
- Pre/post condition constructs
- Facilities for human choice

There are many more

Little-JIL: A Real Language with Precise Semantics

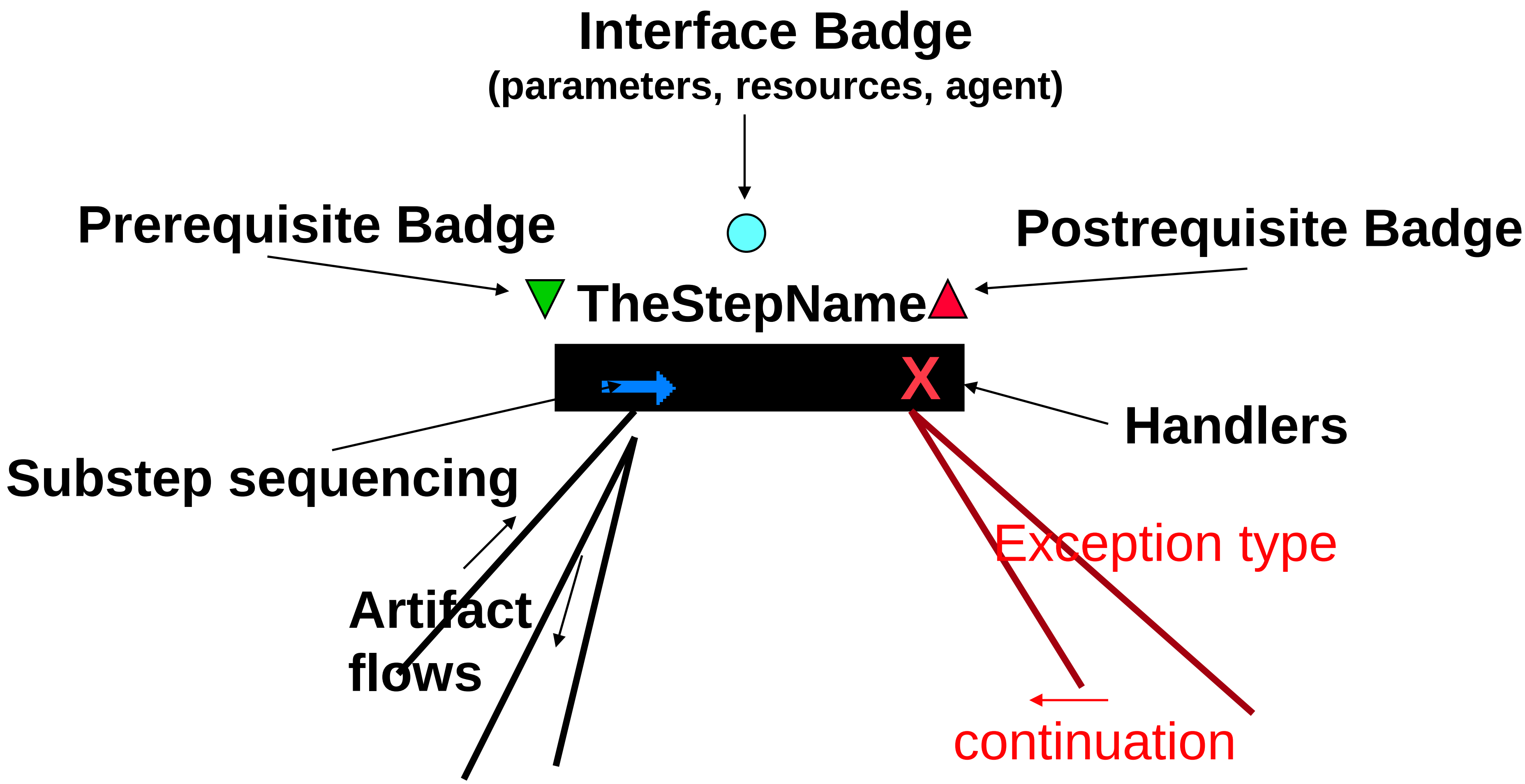
- Process definition is a hierarchical decomposition
- Think of steps as procedure invocations
  - They define scopes
  - Copy and restore argument semantics
- Encourages use of abstraction
  - Eg. subprocess reuse

Little-JIL: A Real Language with Precise Semantics

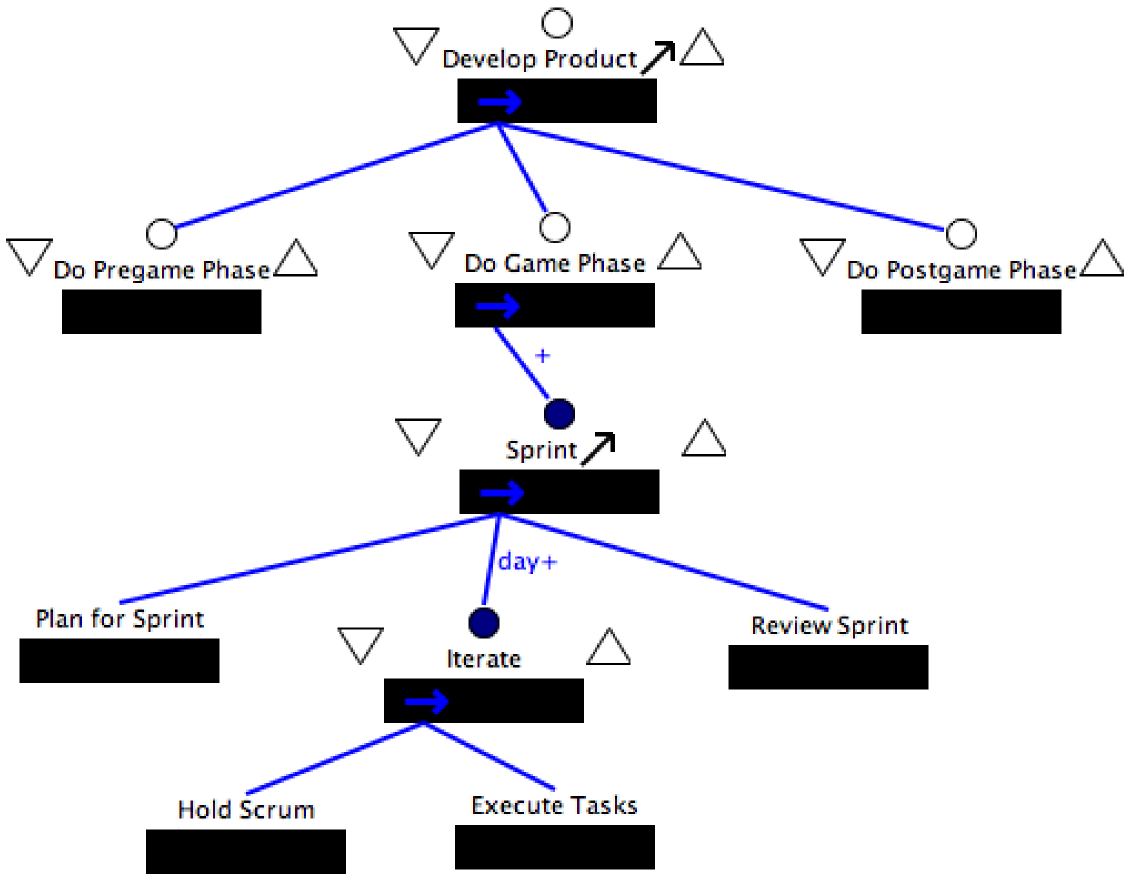
- Process definition is a hierarchical decomposition
- Think of steps as procedure invocations
  - They define scopes
  - Copy and restore argument semantics
- Encourages use of abstraction
  - Eg. subprocess reuse

A key feature in distinguishing this from less formal languages (e.g. workflow)

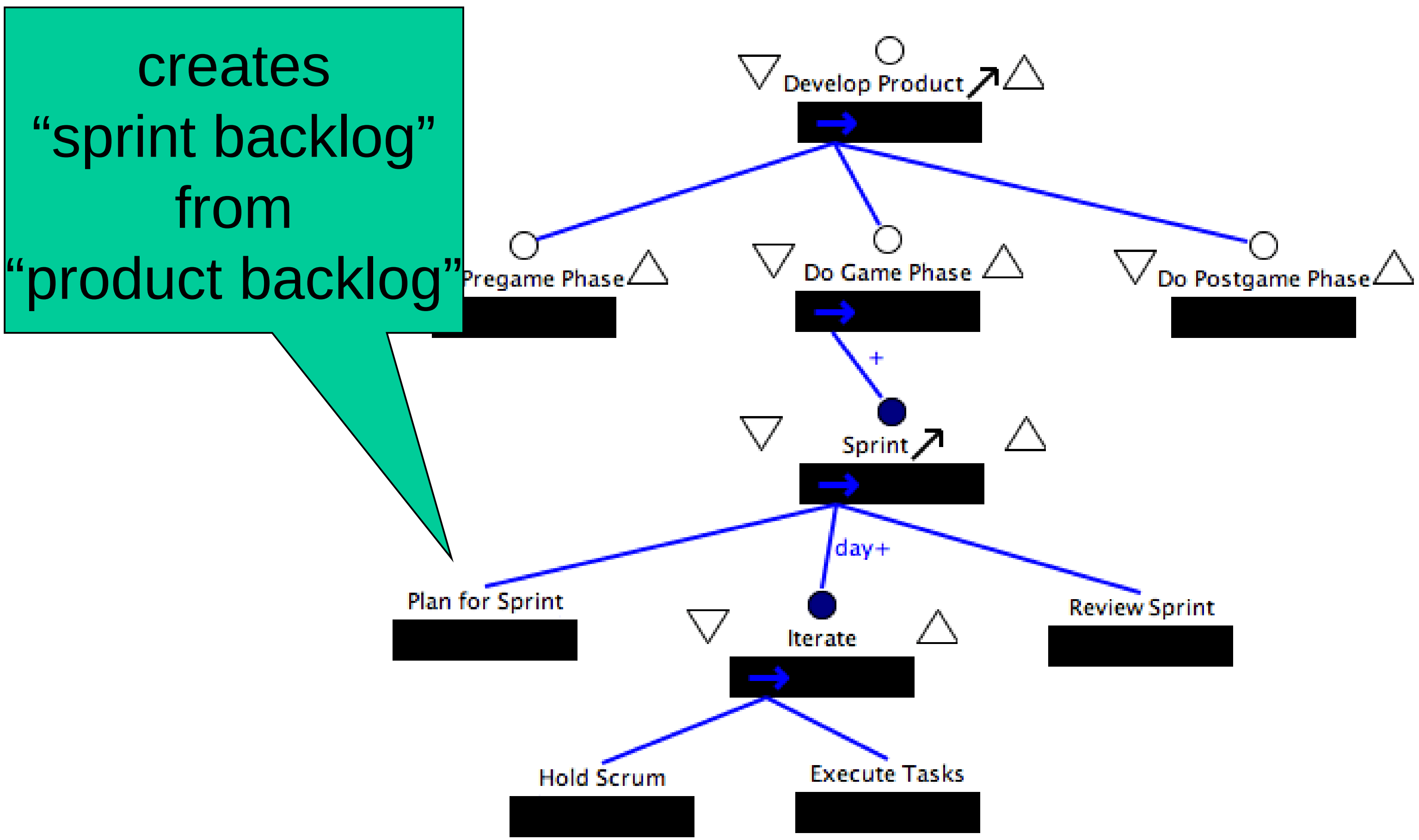
“Step” is the central Little-JIL abstraction



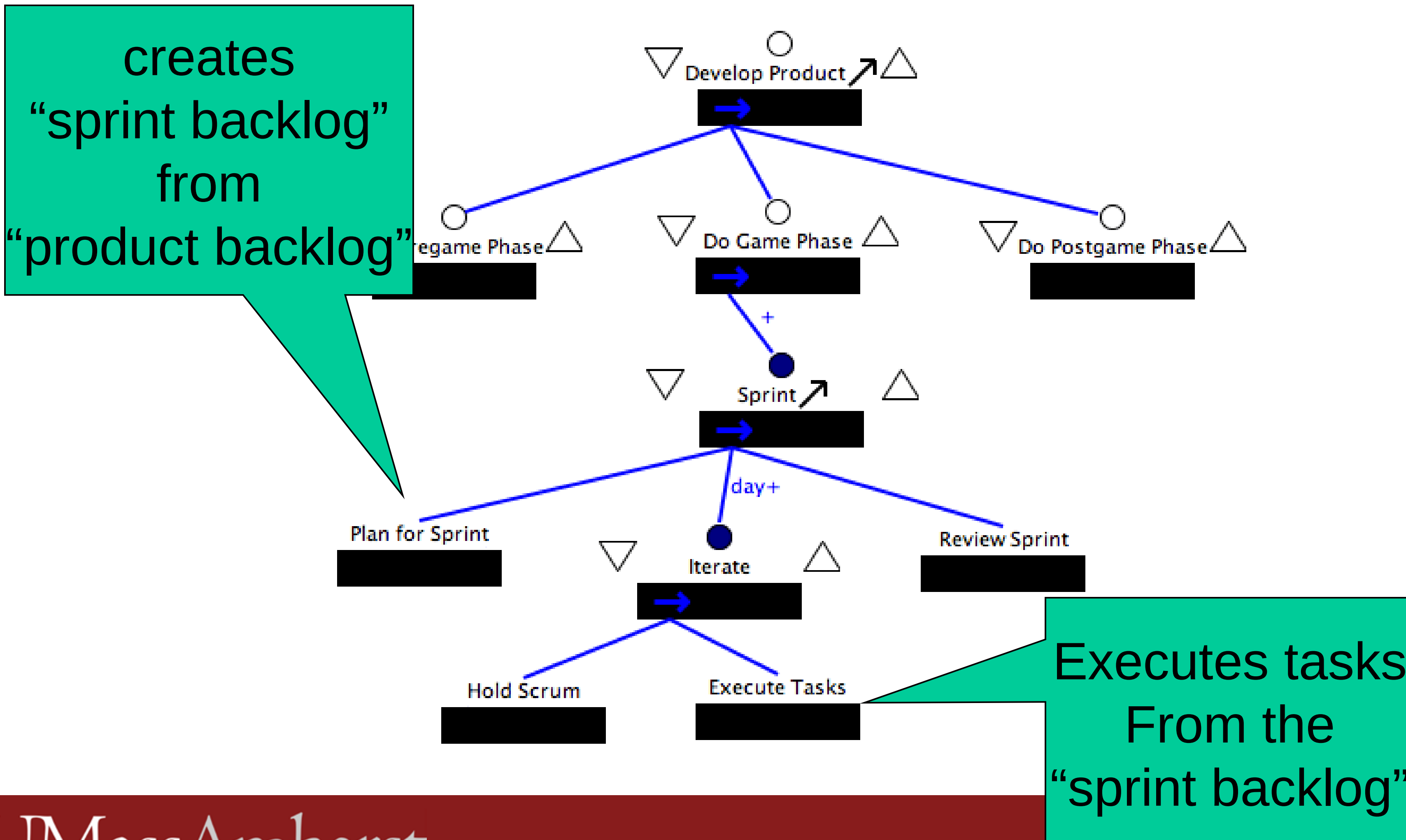
Top level of Little-JIL Scrum process definition



Top level of Little-JIL Scrum process definition



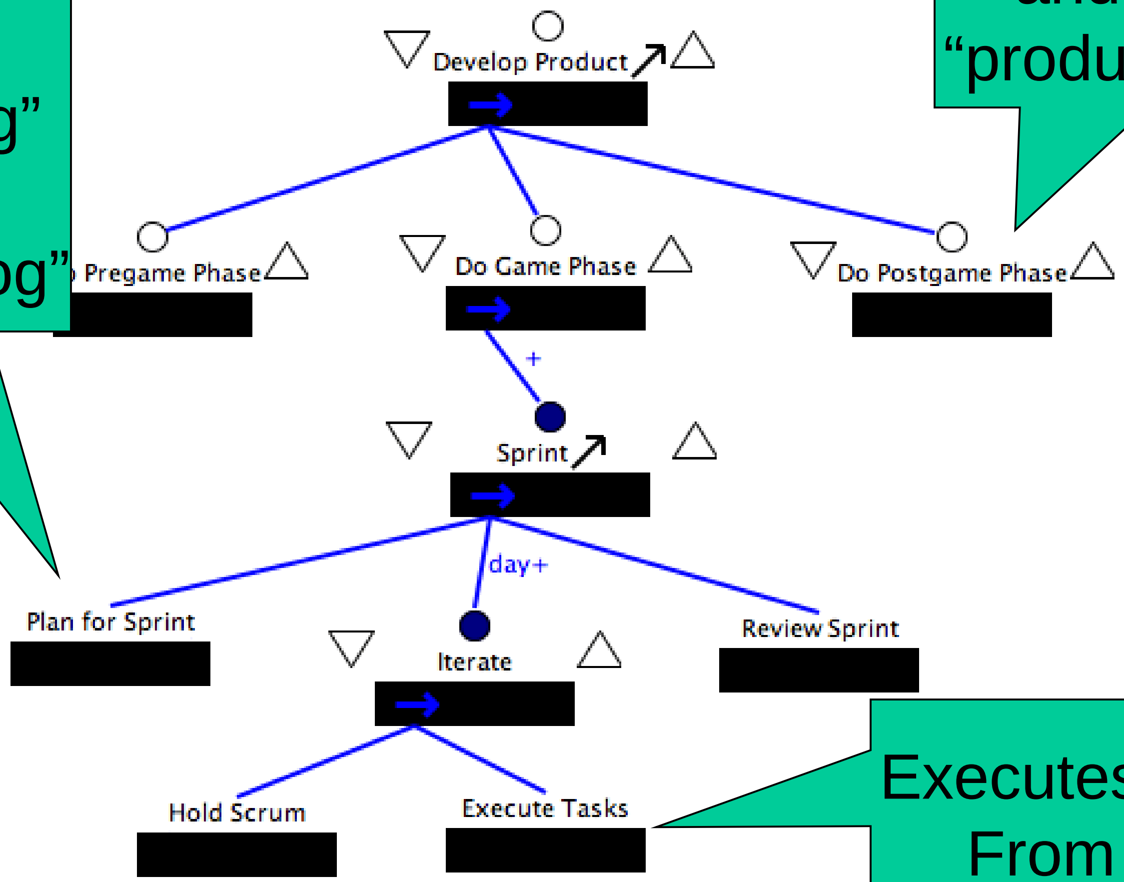
Top level of Little-JIL Scrum process definition



Top level of Little-JIL Scrum process definition

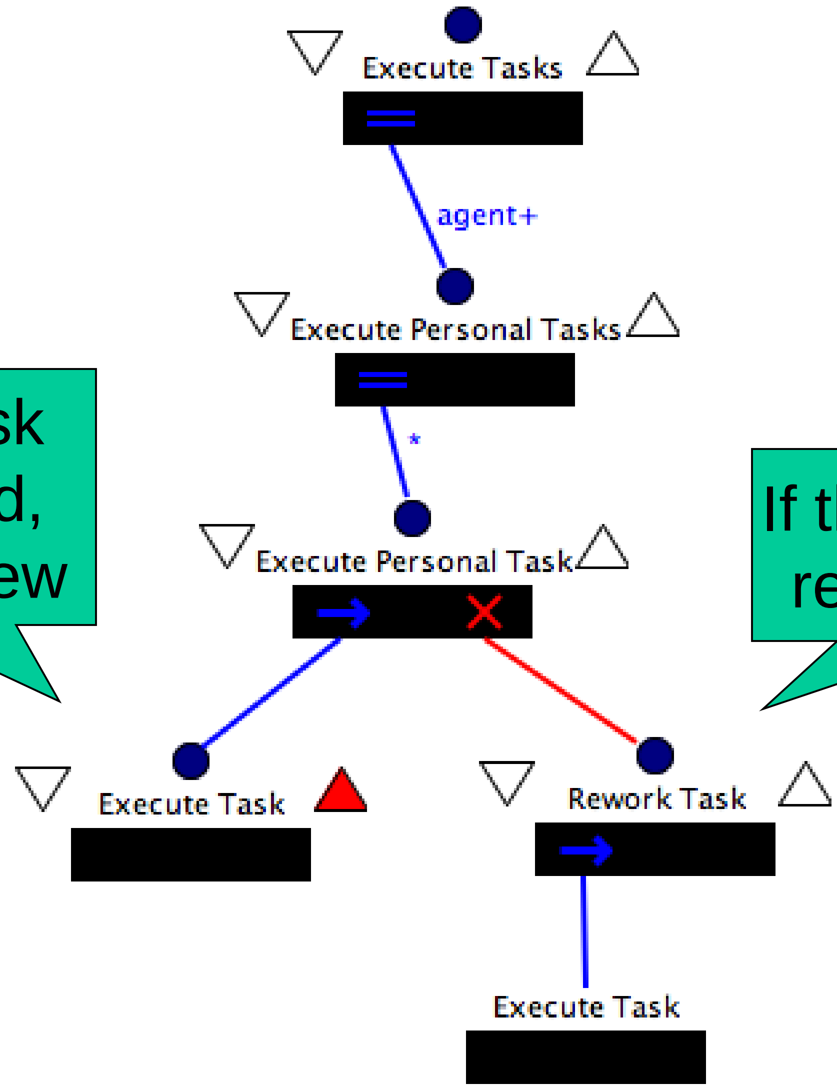
creates "sprint backlog" from "product backlog"

Reviews sprint and updates "product backlog"



Executes tasks From the "sprint backlog"

Elaboration of “Execute Tasks” step



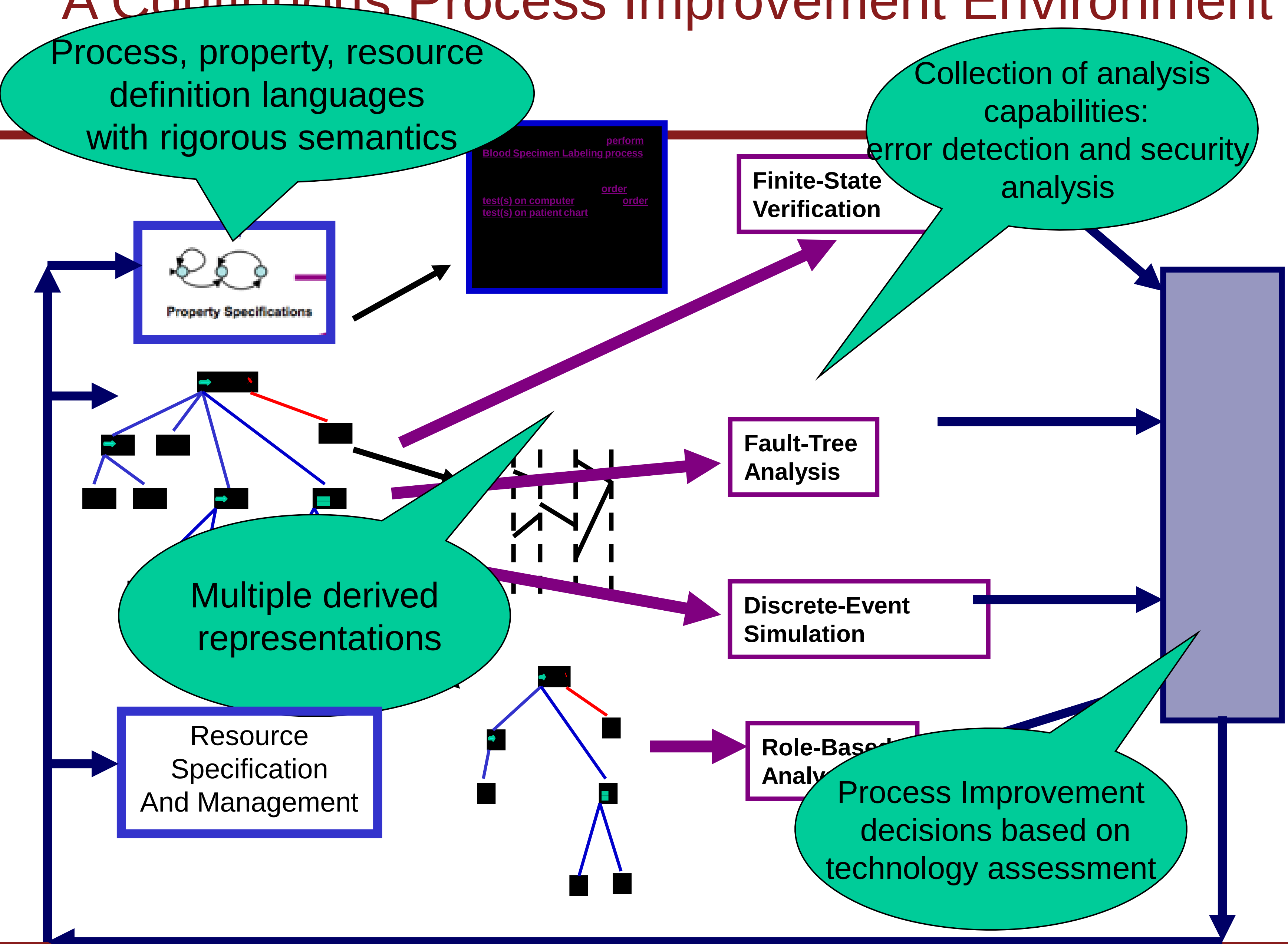
After the task is completed, perform review

If the review fails, rework the task

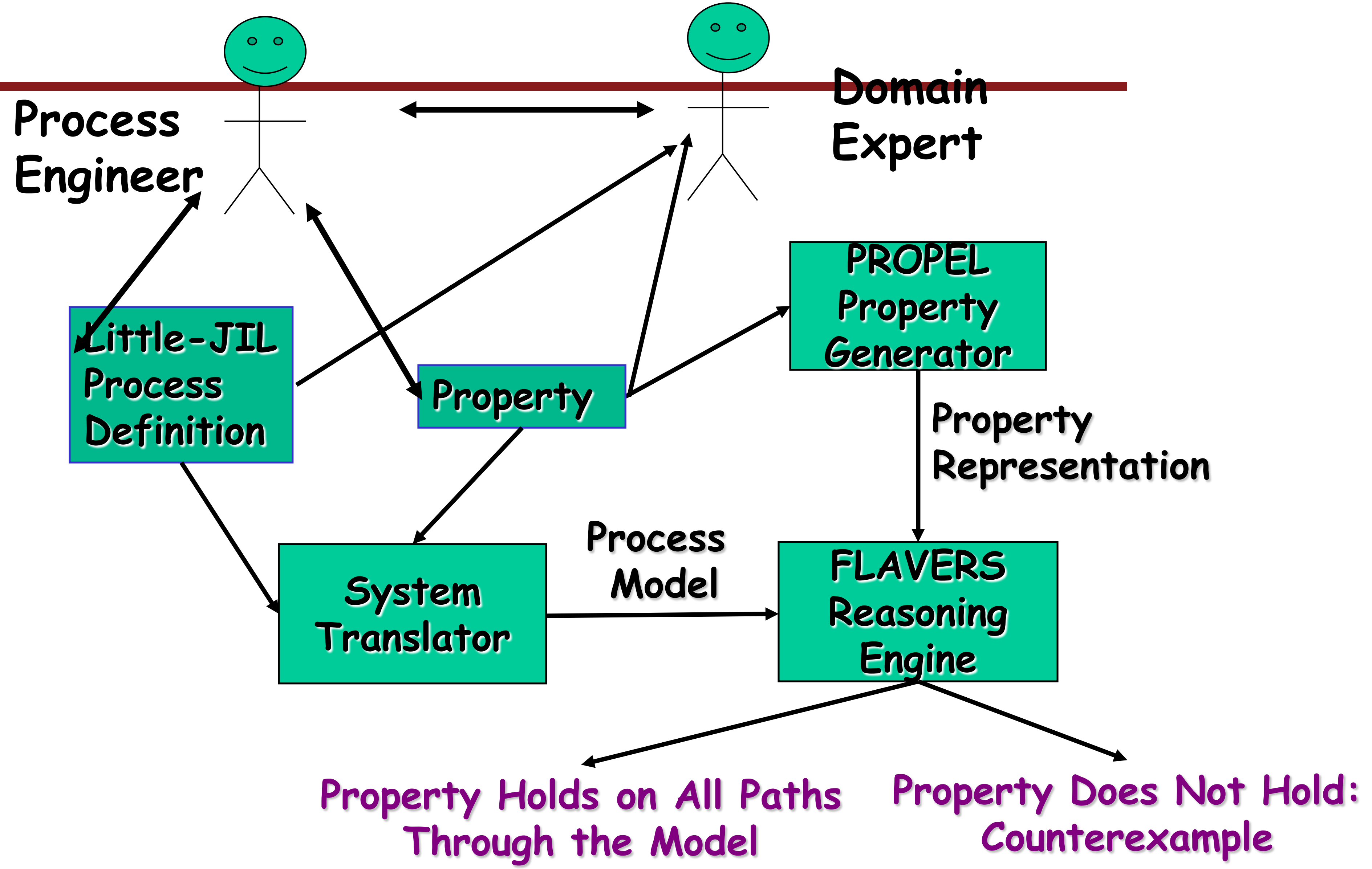
The Basis for Engineering

- Such definitions can then be the subjects for sound analyses
- They can be executed
  - To provide user guidance
- They can be support education and training
- They form the basis for disciplined improvement

A Continuous Process Improvement Environment



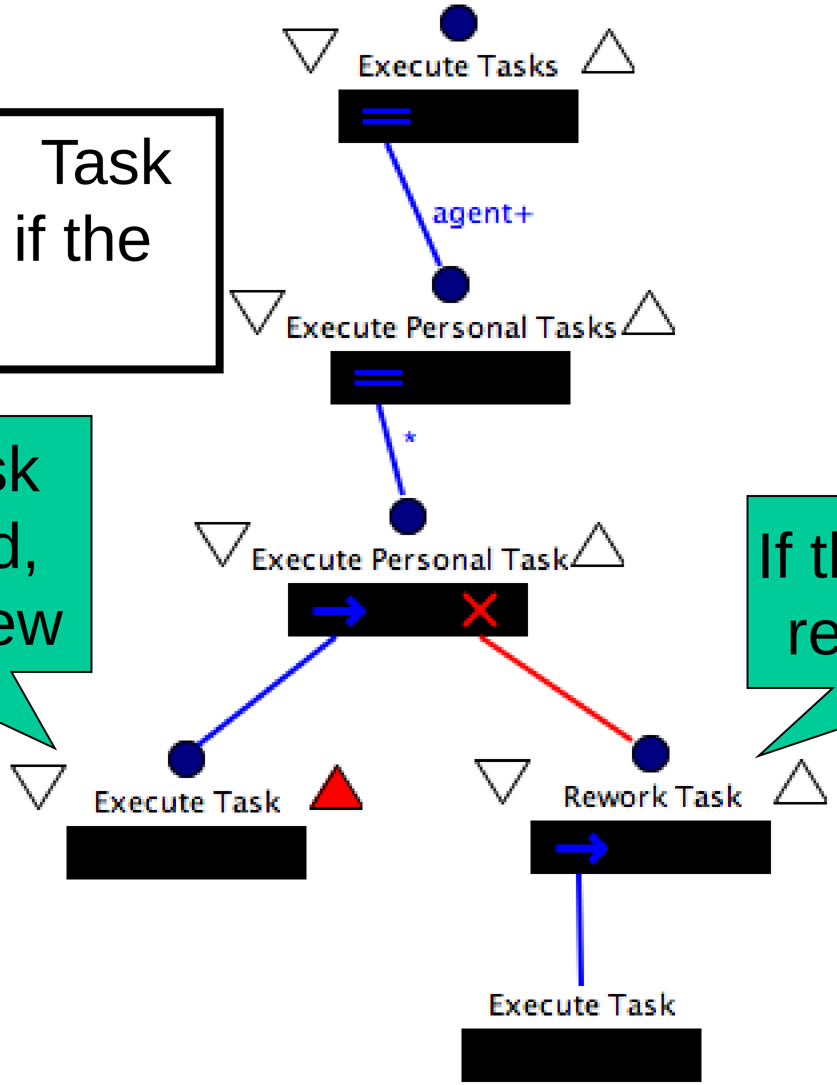
Finite-State Verification



Finite State Verification of Properties

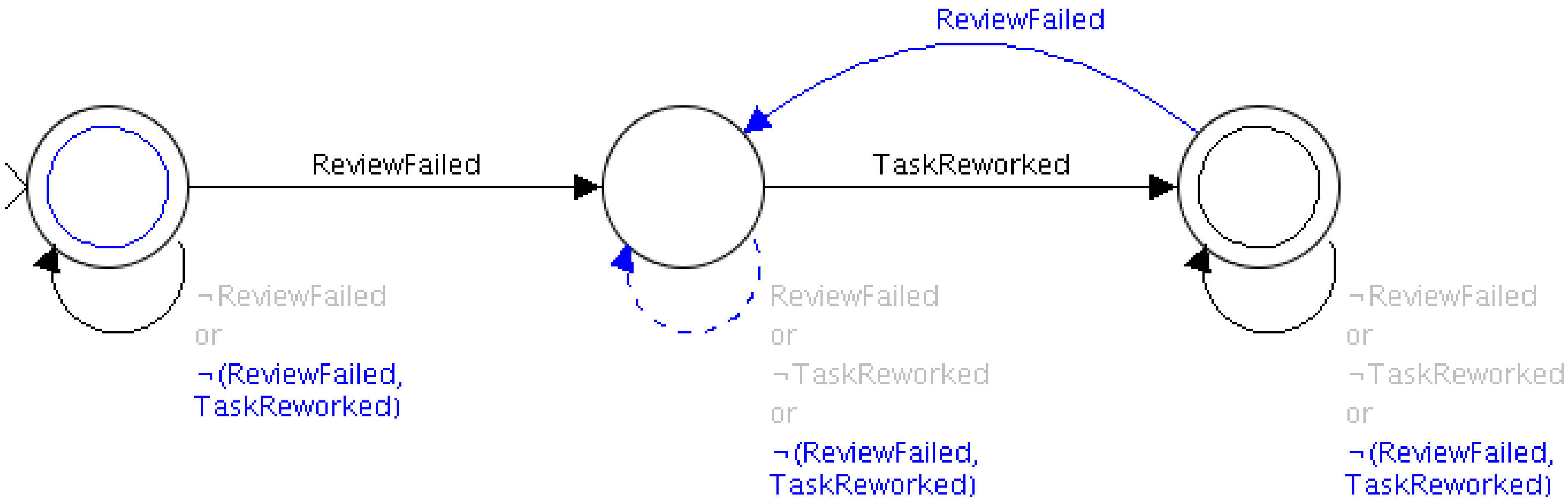
Process property: Task must be reworked if the review fails

After the task is completed, perform review



If the review fails, rework the task

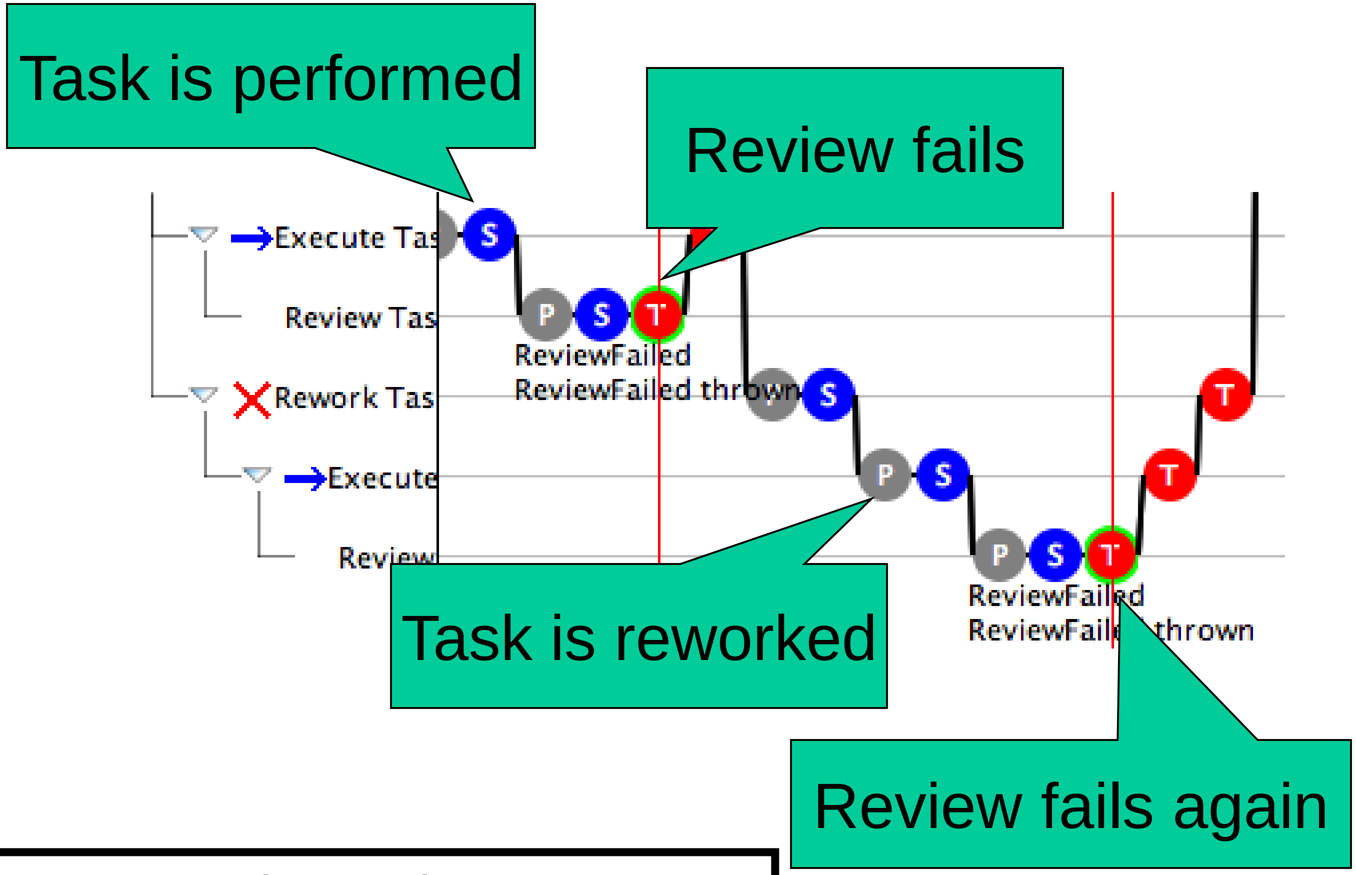
Using Propel to define property "Task must be reworked if the review fails"



Corresponding (Disciplined) English Description of Property

QuickTime™ and a
decompressor
are needed to see this picture.

FLAVERS-generated trace showing how the property can be violated



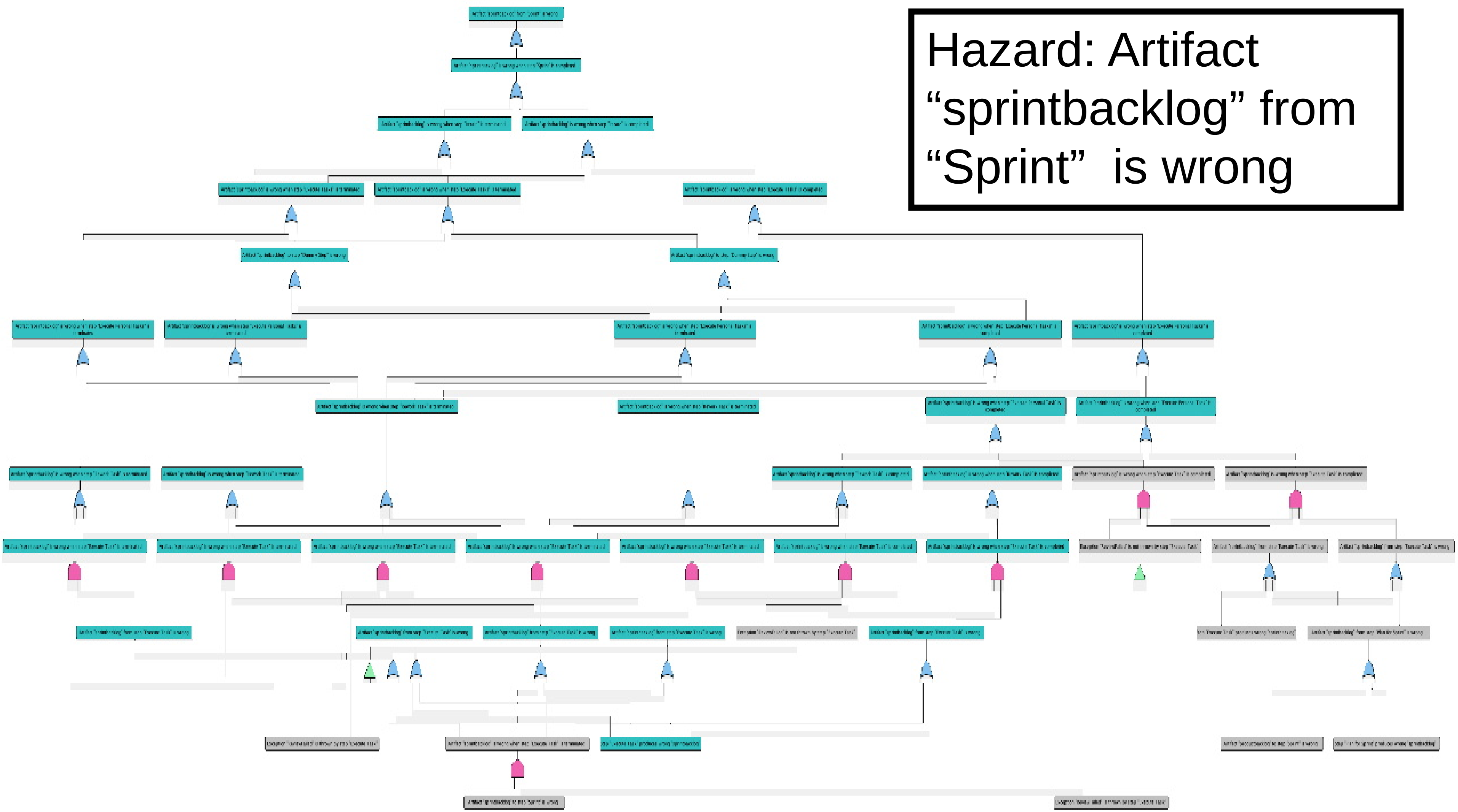
Suggesting a correction to the process-technology-driven process Improvement

Fault Tree Analysis (FTA)

- A well accepted and widely practiced hazard analysis technique
- Systematically identifies and reasons about all possible events that could lead to a given hazard
  - Create fault tree for a hazard
  - Analyze each fault tree
- Analysis results can be used to improve the process => process improvement

Fault Tree Automatically generated from Little-JIL

Hazard: Artifact “sprintbacklog” from “Sprint” is wrong

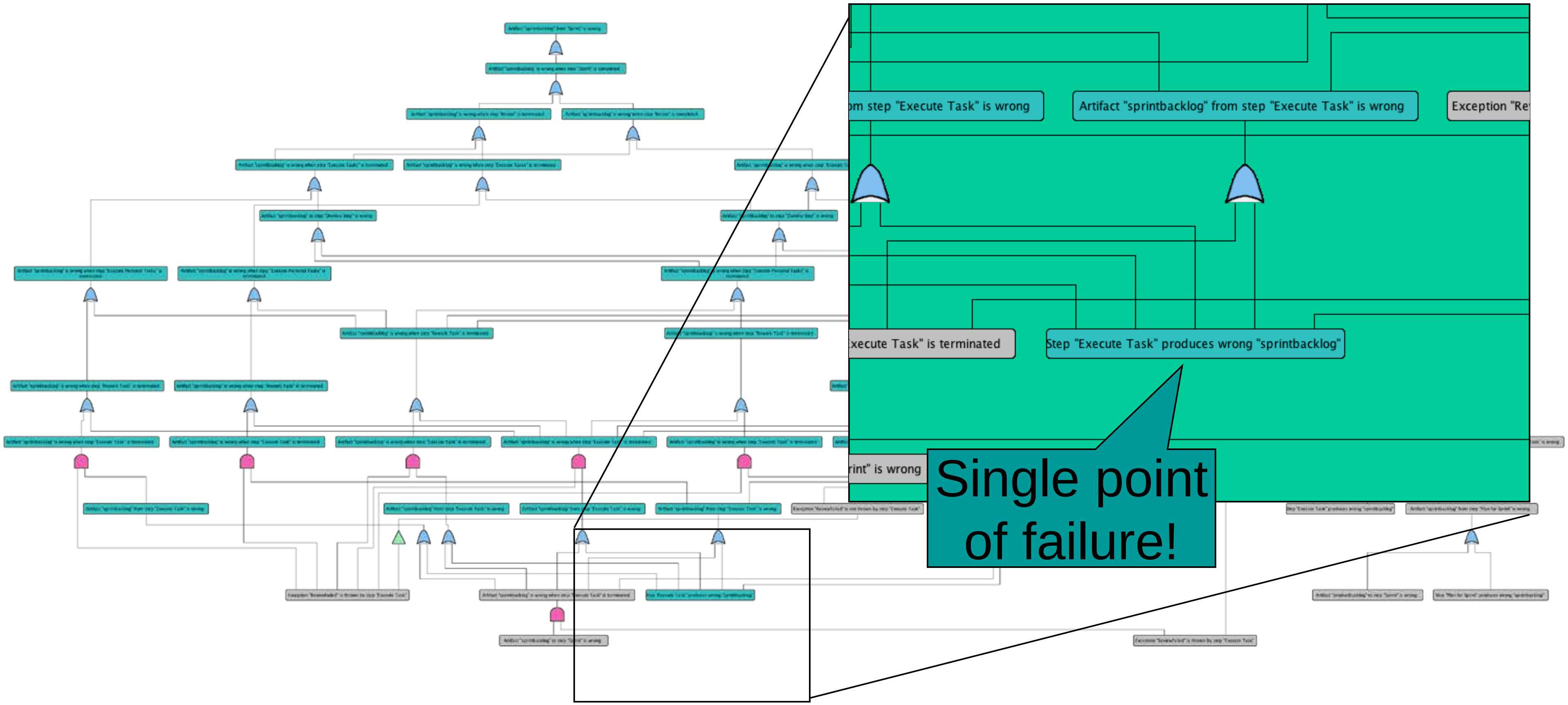


Minimal Cut Sets Can Be Generated Automatically

Minimal Cut Set	Size
!(ReviewFailed thrown by "Execute Task") "Execute Task" produces wrong sprintbacklog	2
!(ReviewFailed thrown by "Execute Task") productbacklog to "Sprint" is WRONG	2
"Plan for Sprint" produces wrong sprintbacklog !(ReviewFailed thrown by "Execute Task")	2
sprintbacklog to "Sprint" is WRONG ReviewFailed thrown by "Execute Task"	2
"Execute Task" produces wrong sprintbacklog	1

Single Point of Failure

Location of Single Point of Failure

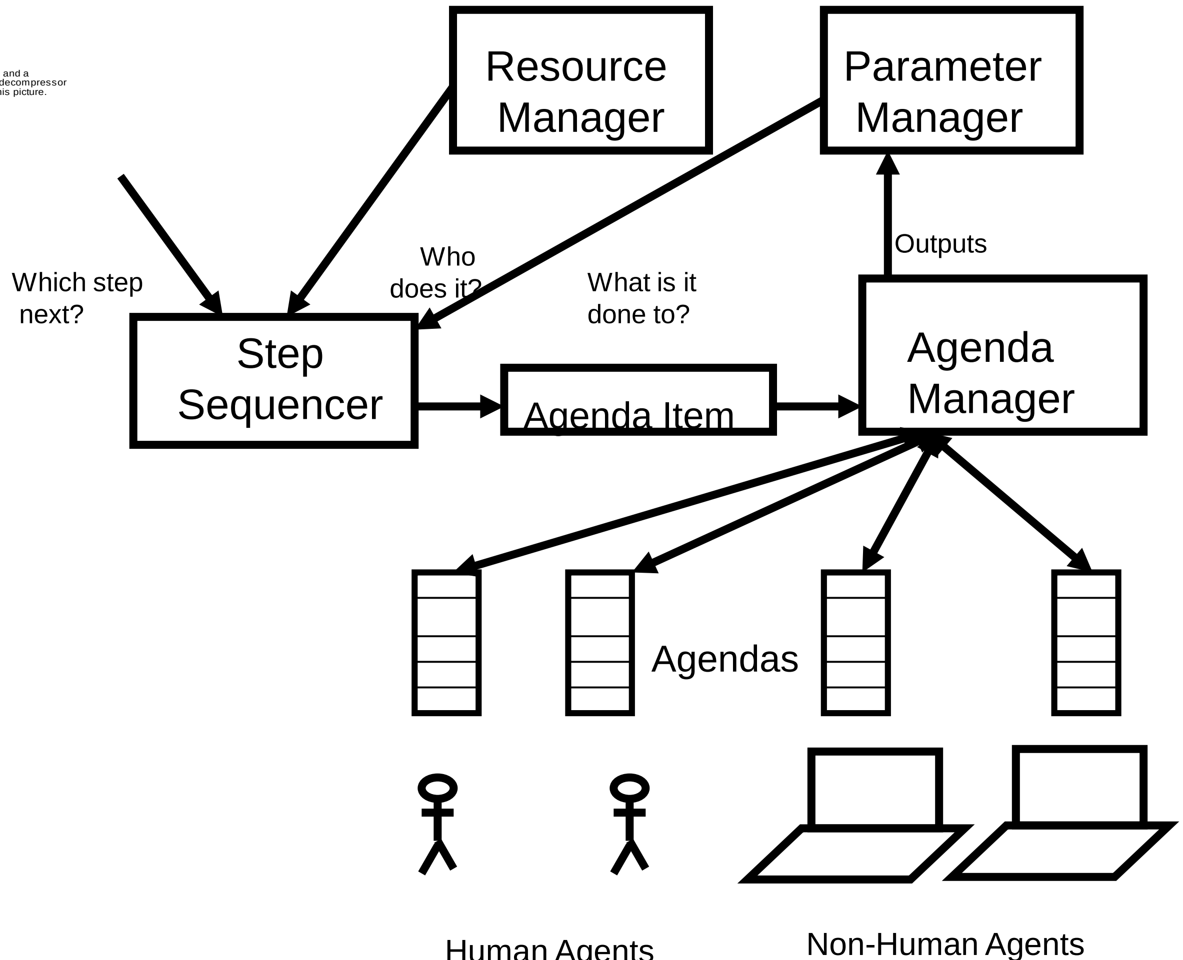


Discrete-Event Simulation

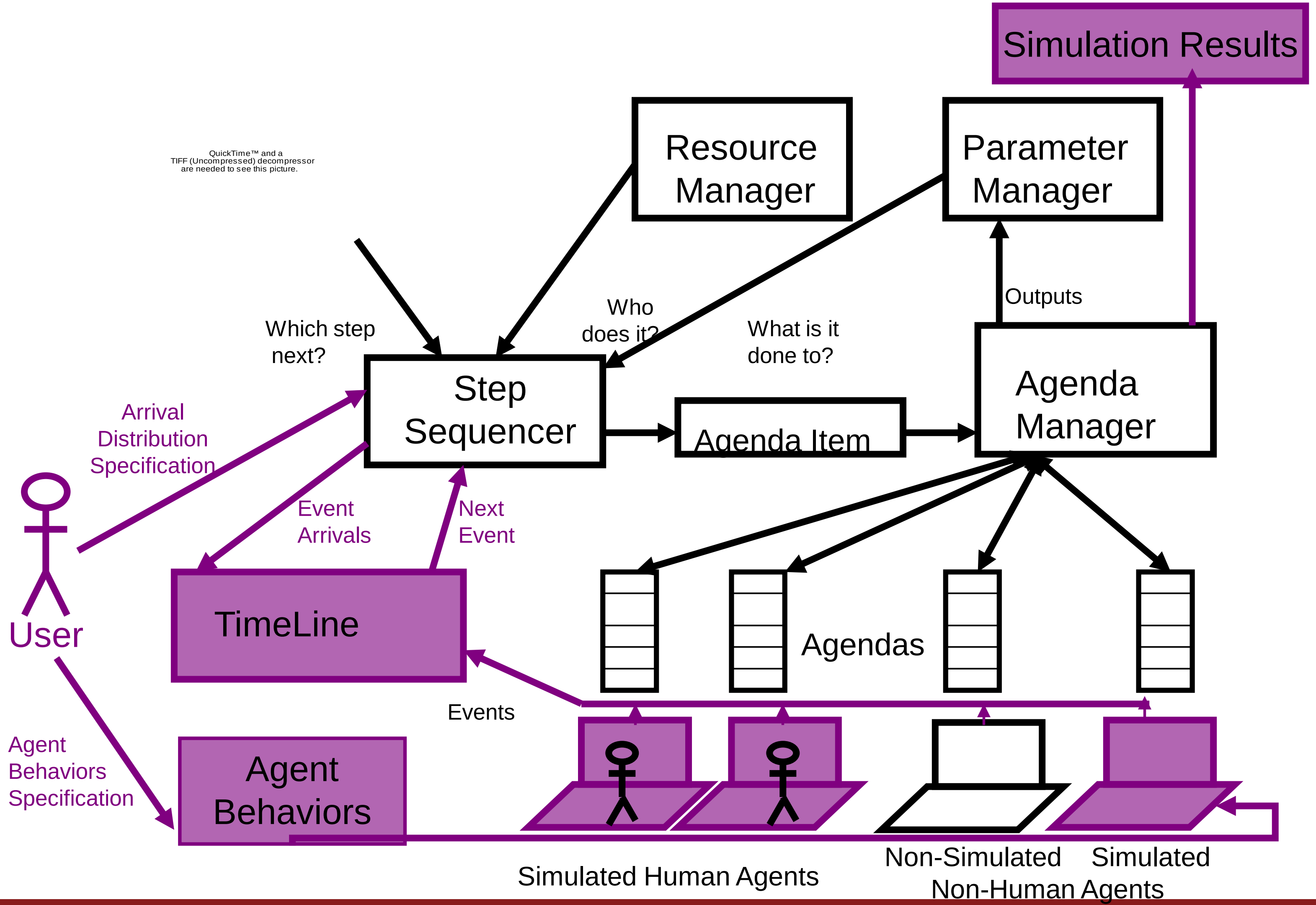
- Use the Little-JIL process models, combined with a resource manager to drive discrete-event simulation
  - Evaluate alternative resource allocations
    - More architects, more programmers, or more testers

Little JIL Interpreter Architecture

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.



Little JIL Simulator Architecture



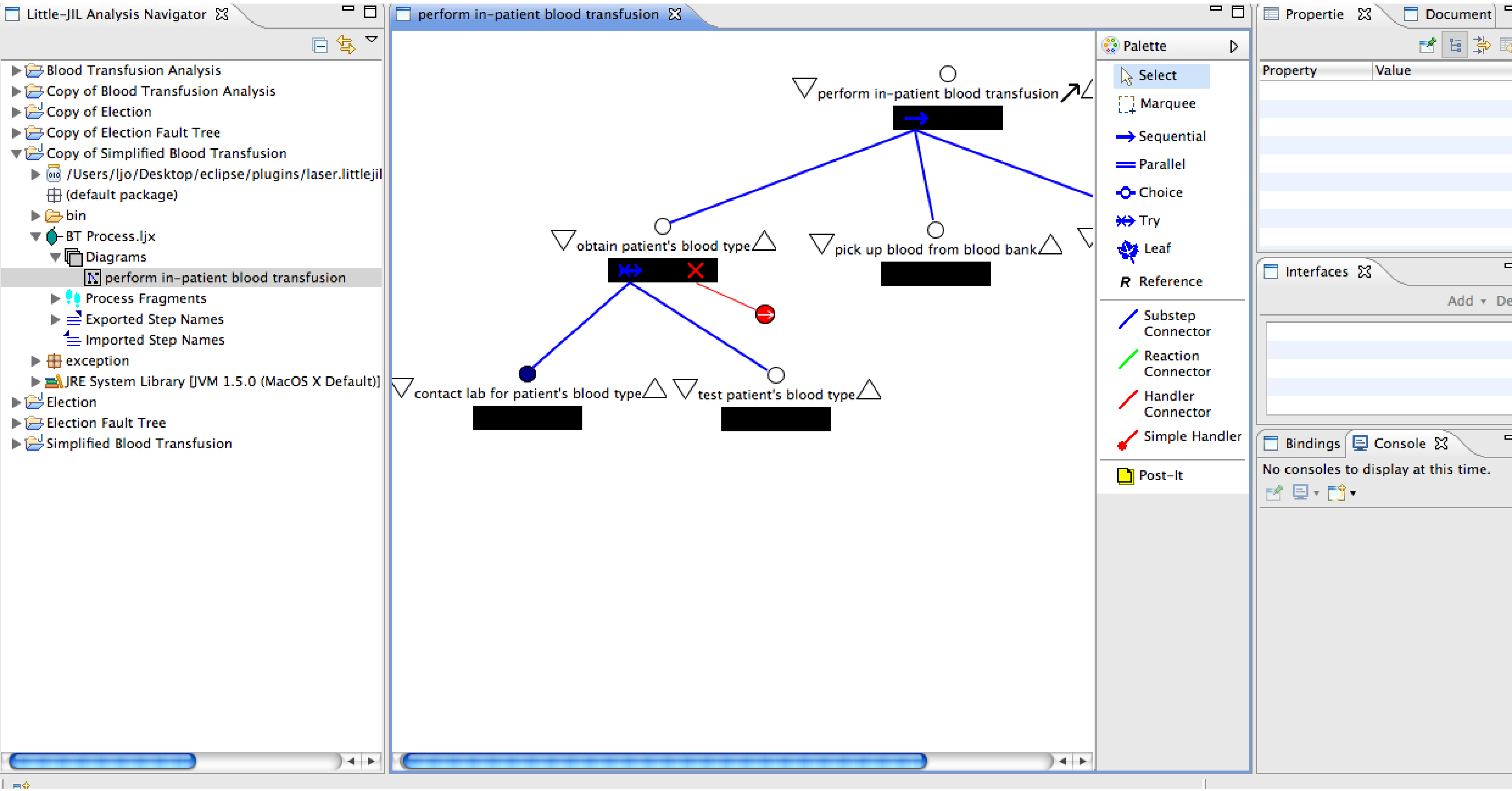
Life Cycle Process Engineering : An engineering discipline applied to the domain of processes

- Integrated approach to process
  - Definition
  - Analysis
  - Simulation
  - Execution
  - Education
  - Improvement

Toolset Status

- Little-JIL language 1.5 is defined
- LASER currently distributes
  - Visual JIL graphical editor
  - Propel property specification system
  - FLAVERS finite state verification system
  - Fault tree generator and analyzers
- Working, but not distributed yet
  - Juliette runtime execution system
  - ROMEO resource manager
  - JSim finite state simulation system

Toolset Integrated through Eclipse



A SERC-relevant Application: Agile/Adaptive Software Development

- Applying this approach to processes for agile/adaptive system development
- Some examples can be drawn from Agile Methods, Extreme Programming
- Case in point: Scrum-oriented development
  - Define Scrum
  - Analyze Scrum
    - Identify and fix weaknesses
  - Train and educate
  - Provide automated guidance in doing Scrum

Some Research Areas

- What semantic features should a microprocess definition language have?
- How to specify its semantics?
- What analysis approaches should be explored?
  - What can be learned from each
- What is the architecture of a microprocess execution system?
  - What components?
  - How integrated?
- Software artifact provenance
  - What is needed?
  - How to provide it?

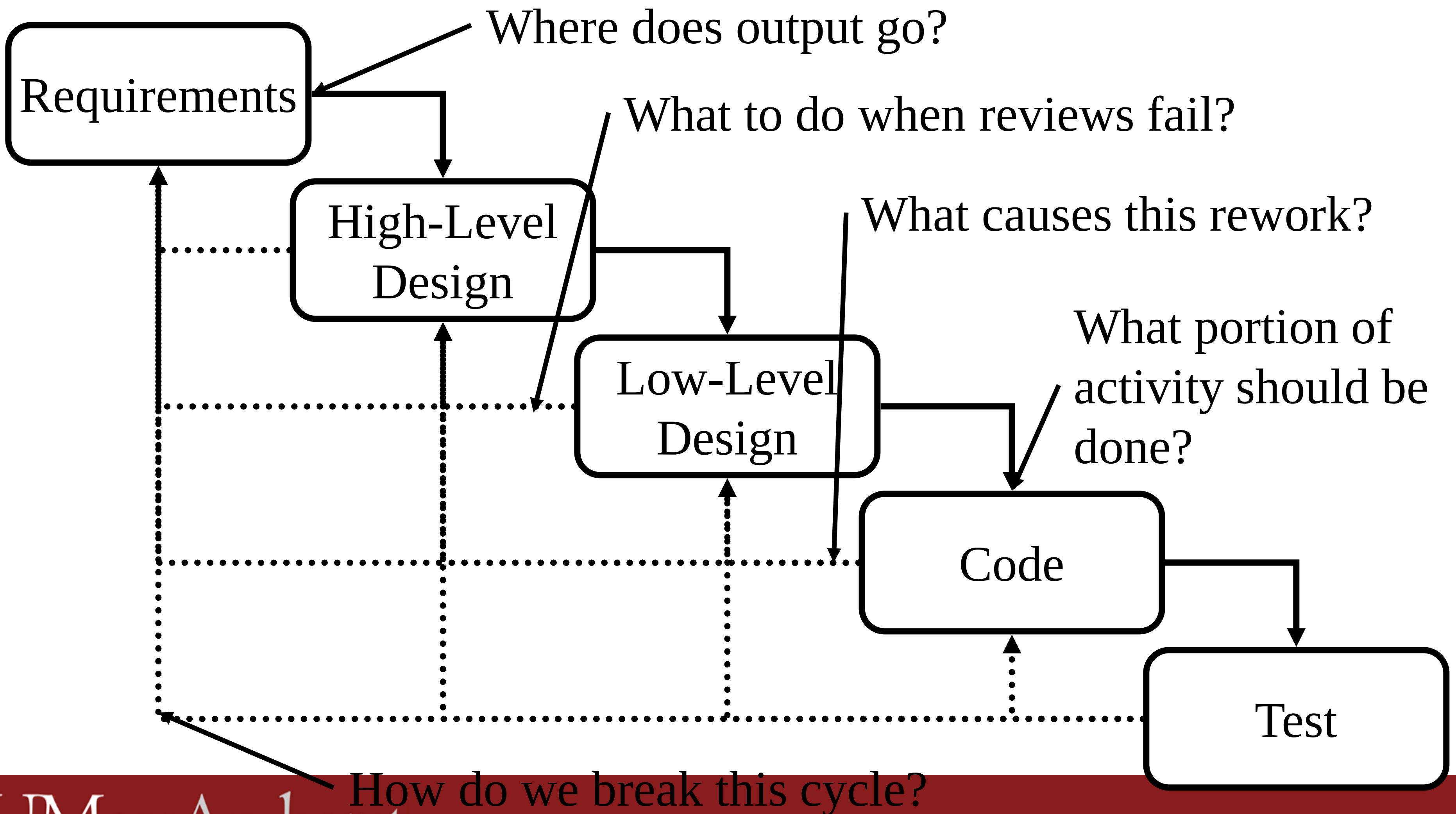
Questions and Discussion

Backup Slides

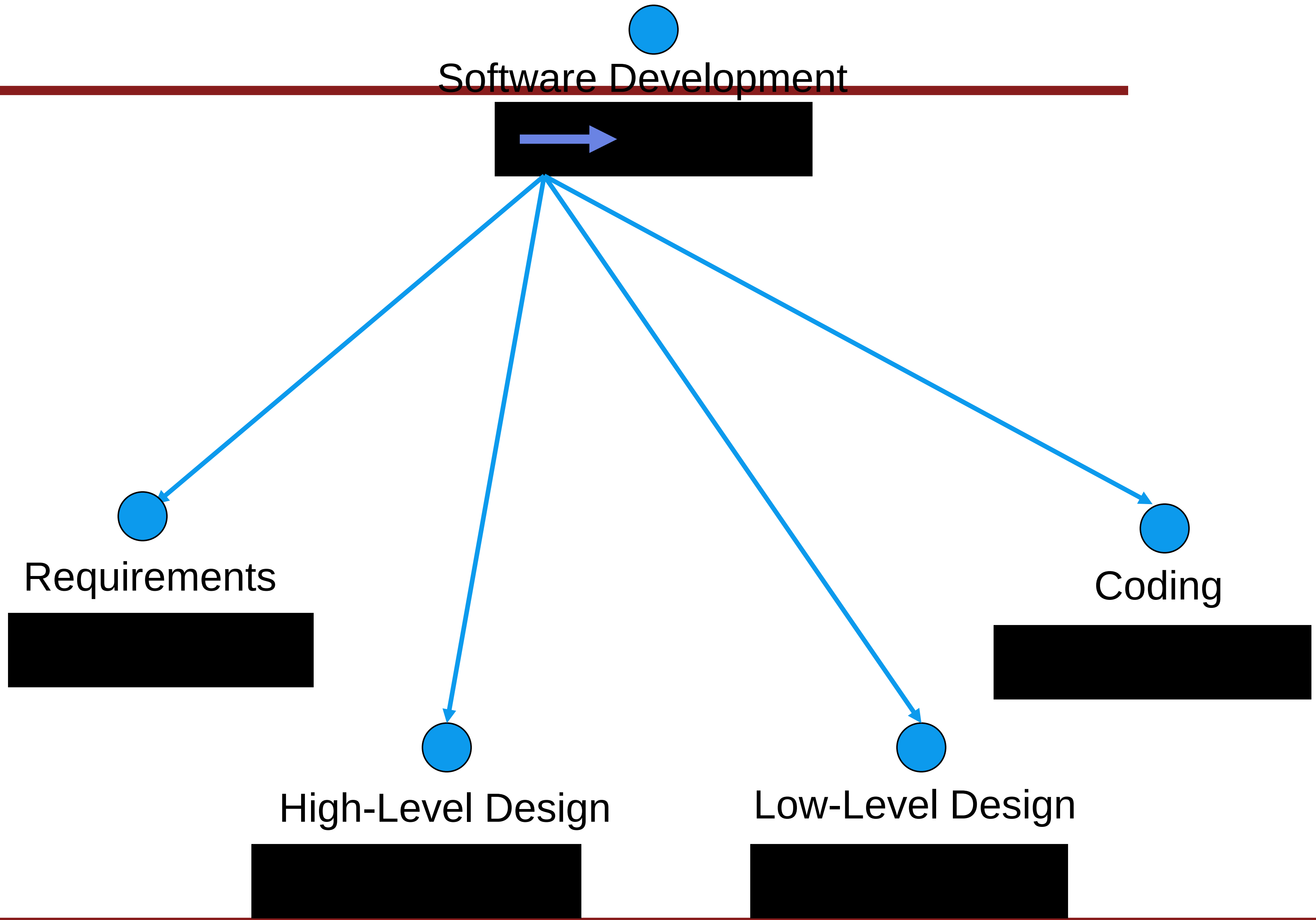
Four parts to a Little-JIL Process

- Coordination diagram
- Artifact space
- Resource repository
- Agents

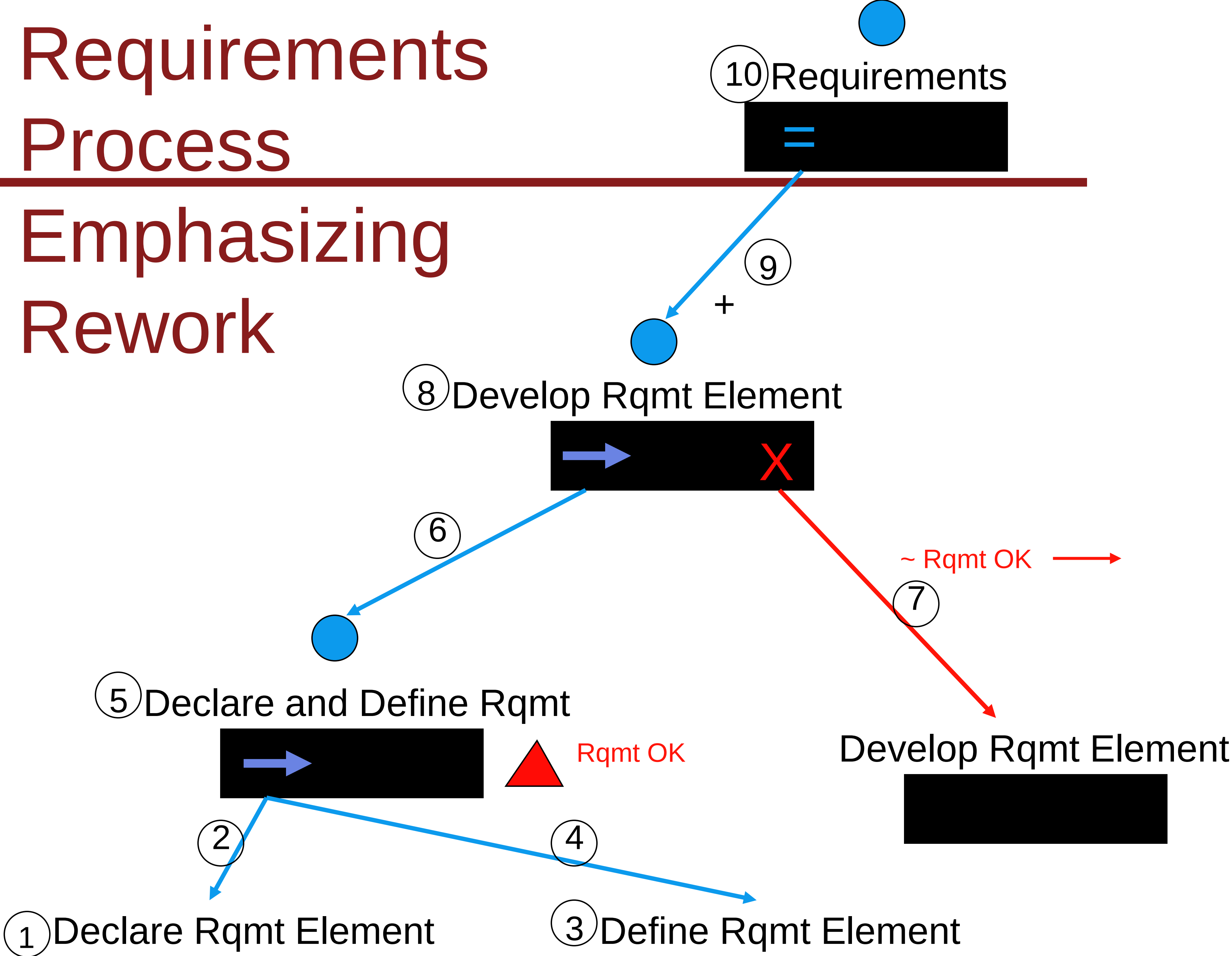
An Articulate Process Can Help Answer Questions Like These

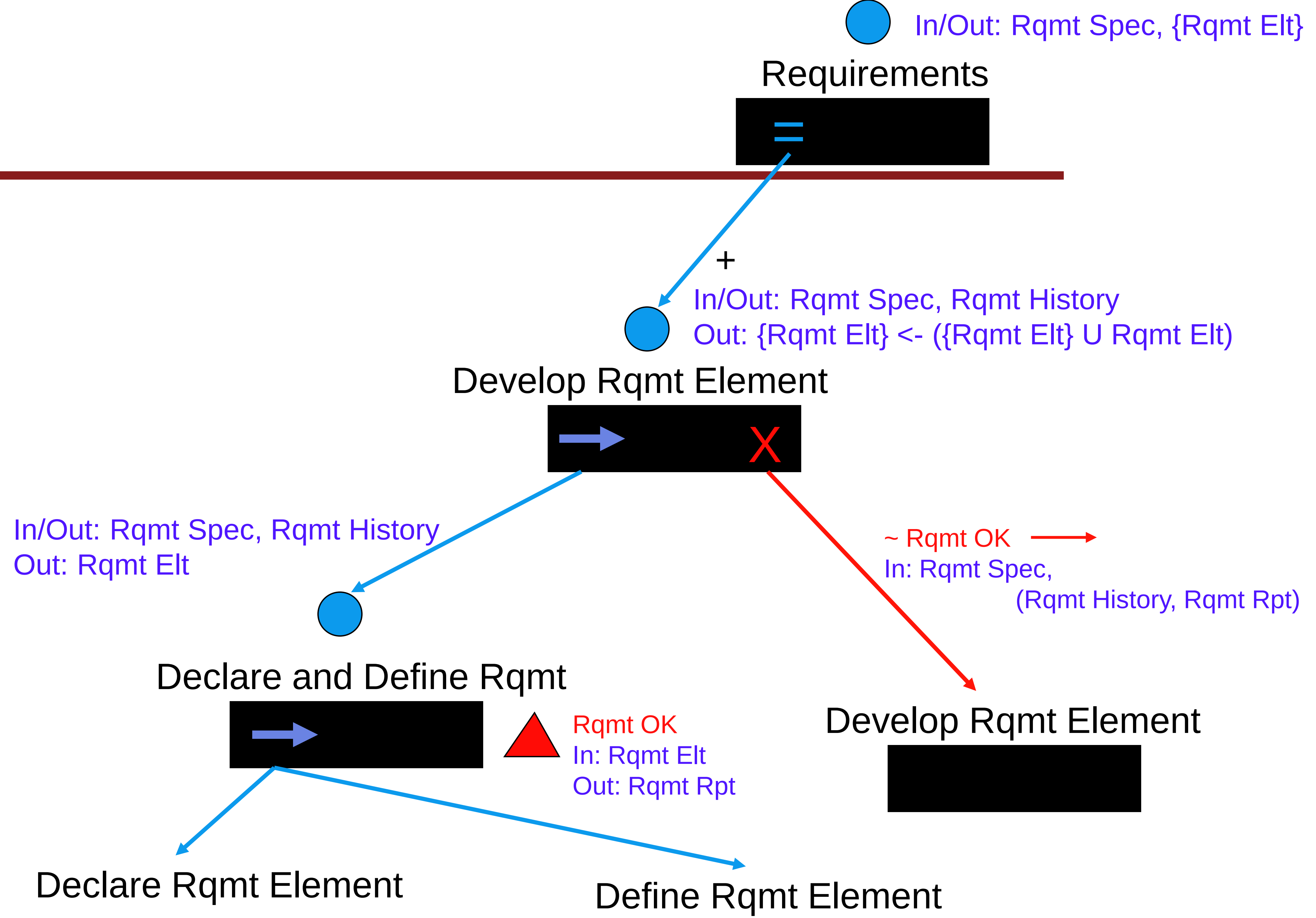


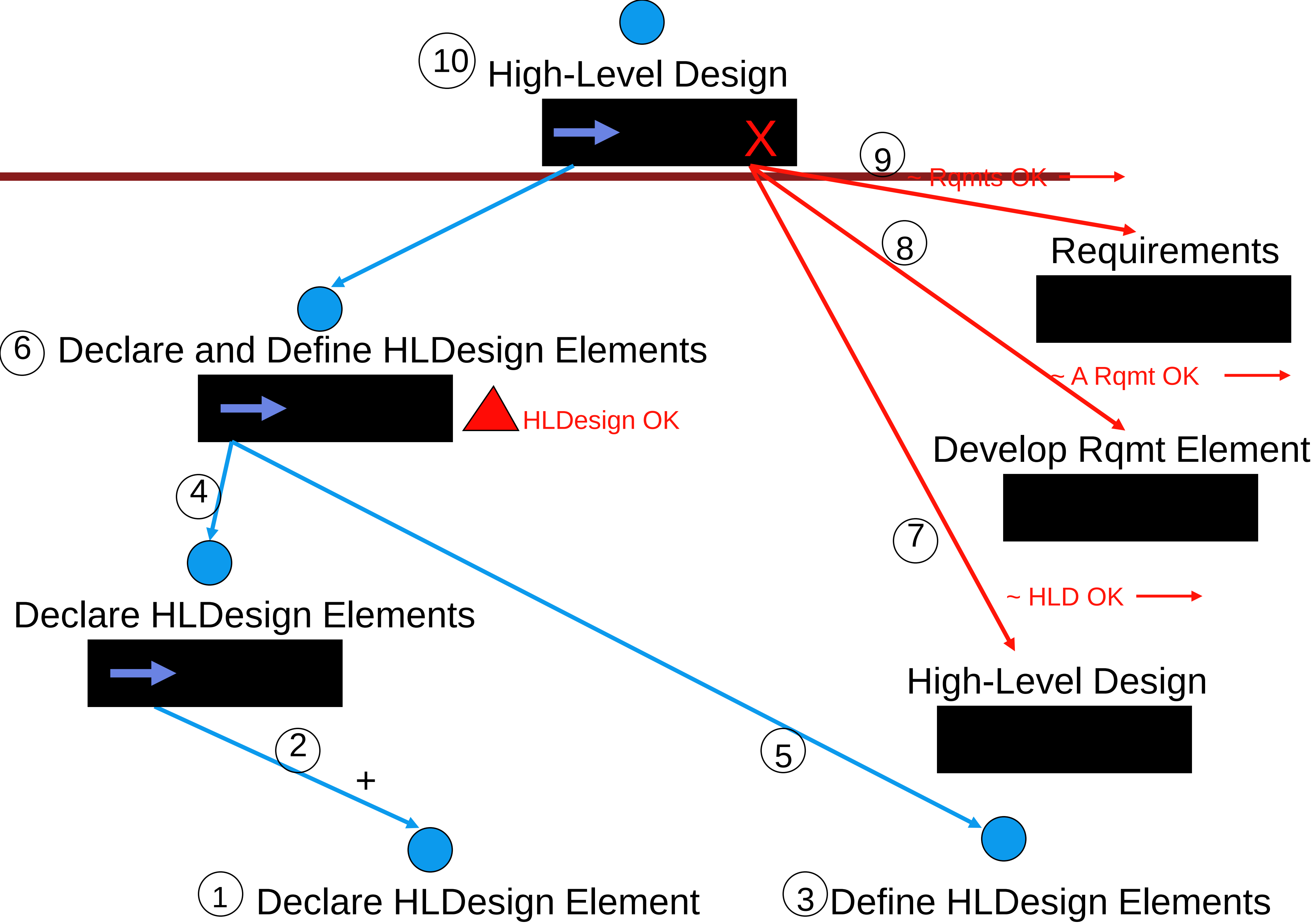
High-Level Process



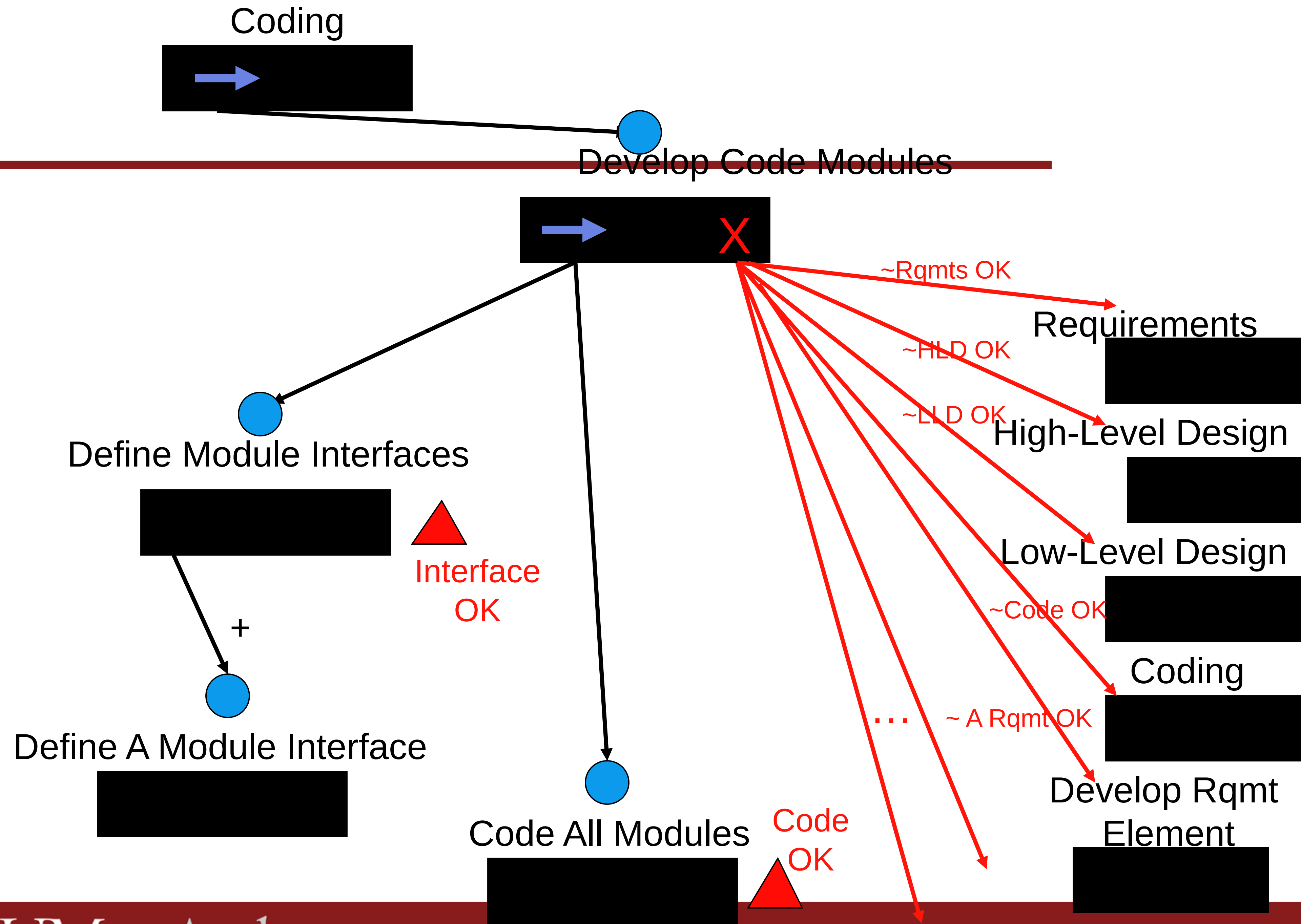
Requirements Process Emphasizing Rework




In/Out: Rqmt Spec, {Rqmt Elt}
Requirements
[redacted]
+
In/Out: Rqmt Spec, Rqmt History
Out: {Rqmt Elt} <- ({Rqmt Elt} U Rqmt Elt)
Develop Rqmt Element
[redacted]
X
In/Out: Rqmt Spec, Rqmt History
Out: Rqmt Elt
~ Rqmt OK
In: Rqmt Spec,
(Rqmt History, Rqmt Rpt)
Declare and Define Rqmt
[redacted]
Rqmt OK
In: Rqmt Elt
Out: Rqmt Rpt
Develop Rqmt Element
[redacted]
Declare Rqmt Element
Define Rqmt Element



10
High-Level Design
[redacted]
X
9
~ Rqmts OK
Requirements
[redacted]
8
~ A Rqmt OK
Develop Rqmt Element
[redacted]
6
Declare and Define HLDesign Elements
[redacted]
HLDesign OK
4
7
~ HLD OK
Declare HLDesign Elements
[redacted]
High-Level Design
[redacted]
2
+
5
1
Declare HLDesign Element
3
Define HLDesign Elements


Coding
Develop Code Modules
~Rqmts OK
Requirements
~HLD OK
~LLD OK
High-Level Design
Define Module Interfaces
Interface
OK
+
Low-Level Design
~Code OK
Coding
Define A Module Interface
. . .
~ A Rqmt OK
Develop Rqmt
Element
Code All Modules
Code
OK
. . .

FSV Using Propel and FLAVERS