A developer-oriented approach to

# Software Assurance and Evolution

William L. Scherlis

CMU School of Computer Science

**scherlis@cmu.edu**
**412-268-8741**

**The Fluid Project**
www.fluid.cs.cmu.edu

March 2005

## Distrust in software T&E criteria

### Ariane 5 — mission critical software

- Starting point: "Heritage" Ariane 4 code
- The code contained a known unhandled exception in the software
- Decision: Reuse for Ariane 5: **don't repair**
  - **Trust** in the legacy
    - It worked for Ariane 4
  - **Distrust** in defined criteria
    - Too risky to modify "working" software even when it is known to be broken
- Ariane 5 veered off course and exploded 40 seconds into its maiden flight

## Direct measures

**We treat our software as if it were a phenomenon of nature**

— Sir Tony Hoare, HCSS 2004

**Indirect Measures**
- Process
- People
- Bug counts
- KLOC counts

➡

**Direct Measures**
- Model coverage
  - By attribute kind
  - By code coverage
- Code/model consistency

## Verification for software engineering — goals

- Scale-up
- Composition
- Adoptability by working developers
- Respect for engineering process
- Relevance and utility for specific needs
- ROI model for investment
- Attribute comprehensiveness
- Predictive and evaluative measures

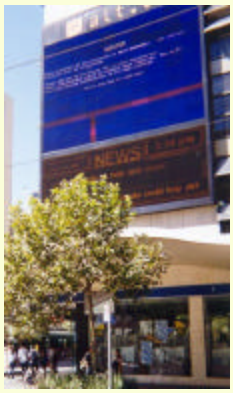
Small theorems about large programs

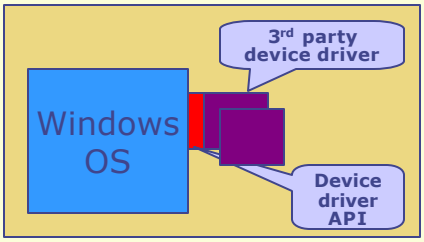
## Verification for software engineering — our approach

- Constructive engagement
  - **Integrate with the engineering process and its participants.**
  - **Leverage and drive business and economics forces.**

- What's new – why this is feasible now
  - **Concrete evidence of the possibility**
    - Attribute-specific tools with specific useful capabilities
    - ROI case enabled by measurement and feedback
  - **The evidence:**
    - Adoption arguments for individual engineers
    - Market arguments for organizations
    - Technical and empirical evidence
    - Infrastructure to enable the approach

## Evidence of the possibility: Microsoft SLAM for Windows XP






**Direct analysis of WinXP device driver code for API protocol compliance**

1. Compelling **business case** for direct assurance of software artifacts – reduced frequency of blue screens. Explicit feedback loops.
2. Based on the **basic science** of model checking and BDDs: a deeply technical approach to assurance, originated in university labs – with DARPA and NSF sponsorship.

## Who is in your supply chain?

- Participants
  - Internal dev't groups
  - Prime outsource
    - Integrator
  - Secondary outsource
    - Vendor
    - Innovator
  - Offshore
  - Off-the-shelf
  - Open Source
  - etc
- Conventional means for achieving assurance
  - Test, inspect, etc.
  - Trusted parties
    - Organizations
    - Individuals
  - Process constraint
    - Actions
    - Documents

## Quality stakeholders

At each supply chain interface:

- Developers
  - Immediate coding guidance
  - Basis for dependability claims
  - Incremental progress
- Managers
  - Direct evidence / measurement: modeling and assurance
  - Asset capture: Design intent
- CIO organization
  - Standards (e.g., framework enforcement)
  - Organizational memory
- Acceptance evaluators
  - Direct artifact evaluation
  - Proxy elimination

## "Software organizations" and IT SCM

Interface barriers exist between producers and consumers
at all stages of IT supply chains

| **Barriers** | **Mitigation** (today's best) |
|---|---|
| Contractor qualification | CMM / CMMI |
| Requirements definition | Close relationships |
| "Second" sourcing | API conventionalization |
| Risk allocation | Asymmetry |
| Engineering acceptance | Testing, inspection, design analysis |

***Our problem***:
Inadequate to assure dependability

***Source of complacency*** :
Scale-up and diminishing defect density

***Source of concern***:
Plateau of overall assurance achievable?

***Producers:***
Internal teams
Primes
Subcontractors
Outsource suppliers
Off-the-shelf vendors
Open source projects
*etc*
*Nearly all major firms*

## "Software organizations" and IT SCM

Interface barriers exist between producers and consumers
at all stages of IT supply chains

| **Barriers** | **Mitigation** (today's best) |
|---|---|
| Contractor qualification | CMM / CMMI |
| Requirements definition | Close relationships |
| "Second" sourcing | API conventionalization |
| Risk allocation | Asymmetry |
| Engineering acceptance, 1st 90% | Testing, inspection, design analysis |
| Engineering acceptance, last 10% | what goes here? |

***Acceptance evaluation***
- Understanding the code – semantics-based queries?
- Evaluation of the code
  Test, inspect, verification, modeling, simulation, runtime observation, red-team, static analysis, etc.

***Complicating factors***
- Frameworks, libraries
- Intertwined processes
- Invisible producers: M&A code, brownfield

***Producers:***
Internal teams
Primes
Subcontractors
Outsource suppliers
Off-the-shelf vendors
Open source projects
*etc*
*Nearly all major firms*

## Why now?

- Recent emergence of advanced tools
  - Analysis tools
  - Developer and team support: Server-side DB
    - Development traceability
- Measurement approaches
  - Instrumentation
    - Coarse (Watson) and fine (autonomic probes)
  - Attribute-specific measures
- Evidence of a business case
- Need
  - What is pervasive is becoming critical
  - CC++
  - IT SCM needs

## Evaluation best practice

What's also needed:

**Direct assurance** (focused tools and ongoing research)
(Analysis tools, Analytic models)
Assure the software itself — Quality, dependability, security
(Objective evidence)

**Accepted best practices for evaluation:**

- **CMM/CMMI** (cf. ISO 9001x)
  (timeless; comprehensive)
  - Evaluate the **team**
  - Evaluate the **process**

  **Cost** and **schedule** predictability
  (correlates with bug reduction)

- **NIAP/CC** (cf. ISO 15408)
  (timeless; comprehensive)
  - Evaluate the **process**
  - Evaluate the **design**
  - Sample the **product** *

  **Security policy definition**
  **Design compliance**
  (*sampled – no direct assurance)

## Process evaluation

- Premise:

  **The quality of a software system is governed by the quality of the process used to develop and maintain it.**

  – Watts Humphrey, SEI

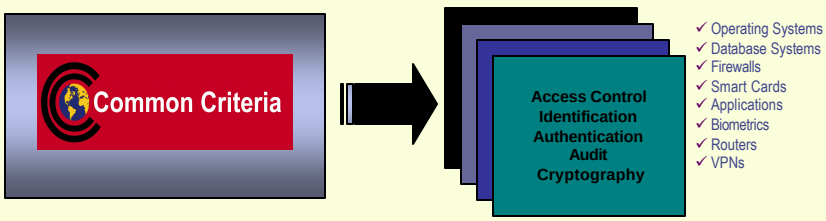
- CMM
  - Developed in 1991, building on work of Deming & others
  - Rationale
    - Qualification of producers to deliver reliably (cost, schedule, quality)
  - Role
    - Describe essential management tasks
      - Document best practices
    - Provide roadmap to improved predictability and quality
      - Prioritize improvements
    - Enable measurement of capability of the organization's process
      - Criterion for source selection
  - Cf. ISO 9001

- Does this approach extend to the "last 10%" ??

## Defining Requirements

ISO/IEC Standard 15408

**Protection Profiles**

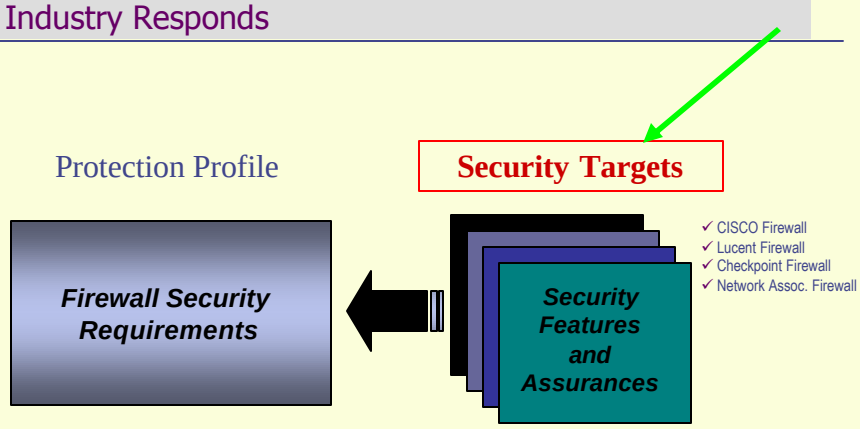



*A flexible, robust catalogue of standardized IT security requirements (features and assurances)*

*Consumer-driven security requirements in specific information technology areas*

From NIST

## Industry Responds




*Consumer statement of IT security requirements to industry in a specific information technology area*

*Vendor statements of security claims for their IT products*

From NIST

## Evaluation: Process and Product

**A Common Criteria certificate:**

- **Does not imply** that the **functional requirements** of the product are approved as "good enough" to provide an adequate level of security in its intended environment of use
- **Does not imply** with absolute certainty that the **product conforms to the security claims** stated by the vendor in the security target
- **Does not imply** or guarantee that the product is free from **malicious or erroneous code**

From NIST

## Looking Forward …

**What's also needed:**

- **Direct assurance** (focused tools and ongoing research)
  (technology-dependent; attribute focused)
  - Assure the **software itself**

**Quality, dependability, security**
(objective analysis)

**Accepted best practices for evaluation:**

- **CMM/CMMI** (ISO 9001x)
  (timeless; comprehensive)
  - Evaluate the **team**
  - Evaluate the **process**

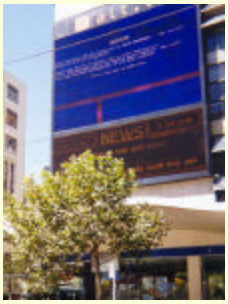
**Cost** and **schedule** predictability
(correlates with bug reduction)

- **NIAP/CC** (ISO 15408)
  (timeless; comprehensive)
  - Evaluate the **process**
  - Evaluate the **design**
  - Sample the **product***

**Security policy definition**
**Design compliance**
(*sampled – no direct assurance)

## The challenge

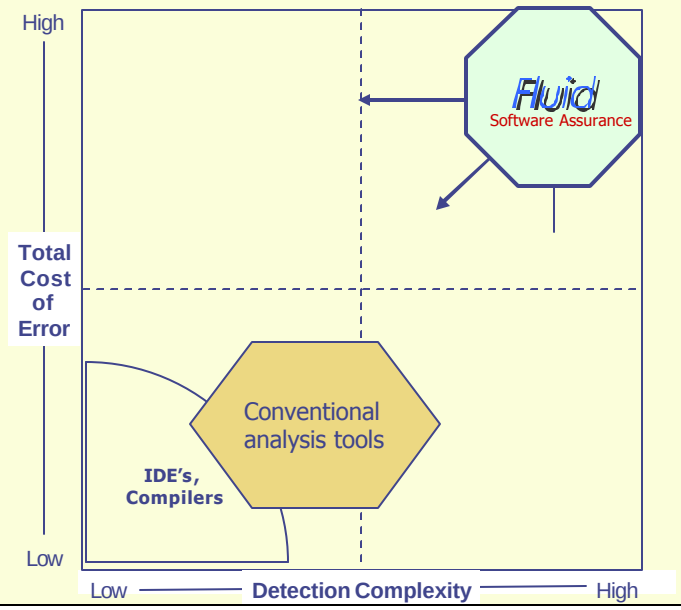
- What are strategies to stimulate **adoption** in the full supply chain for critical systems
- What is the stimulus for a **business case?**
  What economic value do we place on these attributes?
  What causes this to change?
  - Assurance and evaluation
  - The actuarial challenge
  - Means and variances
- We are entering a period of rapidly increasing valuation on dependability, safety, and security attributes
  - Impact on incentive structure within supply chain
  - Not gradual



**Fluid** –
a case study of attribute-specific high assurance

**The Fluid Project**
www.fluid.cs.cmu.edu

## The Testing Dilemma – what attributes do we consider?



## Examples of dependability attributes that are difficult to assure

- **Safe concurrency**
  - Race conditions
  - Lock management
  - Single thread concurrency control
  - Deadlocks
- **Policy compliance**
  - API policy compliance
  - Framework compliance
  - Object references and aliasing
  - Patterns, uses, structure
- **Real time**
  - Real-time thread/memory policies

Focus of this talk

- **Information flows**
  - Security attributes
  - Aliases, references, effects
  - Encapsulation, overlay abstractions
- **Code safety**
  - Appropriate typing

- Hard to **test**
  - Nondeterminism
- Hard to **inspect**
  - Non-local
  - Model-based

Logger.java

```
415     public void log(LogRecord record) {
416         if (record.getLevel().intValue() < levelValue
417             return;
418         }
419         synchronized (this) {
420             if (filter != null && !filter.isLoggable(record)) {
421                 return;
422             }
423         }
```

Hazard vs. Failure vs. **Error** vs. **Fault**

```
386     * @param   newFilter  a filter object (may be null)
387     * @exception  SecurityException  if a security manager exists and
388     *             the caller does not have LoggingPermission("control
389     */
390     public void setFilter(Filter newFilter) throws SecurityException {
391         if (!anonymous) {
392             manager.checkAccess();
393         }
394         filter = newFilter;
395     }
```

**Example race condition**

```
128  * All methods on Logger are multi-thread safe.
```

## The Fluid Project

- Create and maintain safe, dependable, secure code
  - Directly assure critical **dependability** attributes
    - Attributes tend to defy testing and inspection
      - {Dependability, safety, security}
    - Direct static assurance
  - Express dependability-related **models**
    - Incrementally capture design intent
  - Provide **direct positive assurance**
    - Do not allow false negatives
  - Support **measurement of progress**
    1. Inventory of fault-relevant sites
    2. Modeling progress
    3. Analysis progress: assurance, potential faults
  - **Adoptability** and **scalability** are paramount
    - Ease of use by practicing developers
    - Management value – metrics and process support
    - Composability and components
    - Incrementality and early rewards
    - Partiality and contingency

## Models are missing

- **Programmer design intent** is missing
  - Not explicit in Java, C, C++, etc
    - What lock protects this object?
      - **This lock** protects **that state**
    - What is the actual extent of shared state of this object?
      - **This object** is "part of" that object
- Adoptability
  - Programmers: "Too difficult to express this stuff."
  - Fluid: Minimal **effort** — concise expression
    - Capture what programmers are **already thinking about**
    - No full specification
- The Incremental Reward Model
  - Programmers: "I'm too busy; maybe after the deadline."
  - Fluid: **Payoffs** early and often
    - Direct programmer utility – negative marginal cost
    - Increments of payoff for increments of effort

## FLUID Distinguishing Characteristics

1. The "hardest" errors
   - Errors that defy conventional testing and inspection
   - Attribute-specific focus
2. Positive assurance – no false negatives
   - Documenting the absence of important categories of errors
3. Design intent
   - **Incremental Reward Model**
     - Readily adoptable by working developers and easily integrated into ongoing development processes
   - **Analysis-Based Verification**
     - Scalable and composable, enabling assured components to be assembled into assured systems
4. No false positives in practice
   - Tool relies on explicit design intent
   - Tool can assist developer in inferring potential design intent

## Reporting Assurance Results

**Assurance results**

Model – programmer provided design intent / cutpoint

Assured – design intent is consistent with code

Not Assured – design intent is inconsistent with code
Relative to design intent

**Inferred results**

Next steps, reasonable defaults

Warnings and possible problems

**Metric results**

How much have I done?
- Model building
- Assurance development

**Assurance locator**
- Identifies where models and assurance exist within the system's structure
- Incrementality allows assurance of focused "islands" within a large software system
  - Cut points allow programmer selected modularization of assurance efforts

## Formative empirical studies

- Analysis of concurrency in production Java code
  - Abundant race conditions, including in published code exemplars
- Automatic "quality" analysis of 2 MLOC of production Java code
  - E.g., 20% of caught exceptions are ignored (most without comment)
- Analysis of 45,000 Java open source bug reports
  - "Hard" problems –
    - API confusion, design comm, code safety, ripples, portability
  - Concurrency bugs often don't get resolved:
    - "Works for me"
    - "Its intermittent – the garbage characters will usually go away if refresh your browser."
    - "After several more months looking at this bug I might have some more insight…"
- CERT security exploits database
  - More than 90% enabled by engineering errors
  - More than half result from coding and low-level design errors

## Race conditions

- Races can occur when:
  - Multiple threads of control access shared data
  - Data gets corrupted when internal integrity assumptions are violated.
- **How we protect against races**
  - Use "lock" objects that enable access by one thread at a time
    - E.g., event dispatch
    - A language feature in Java, Ada95, etc.

    … **or** …

  - Follow a thread discipline in which only one thread can access critical data
    - E.g., graphical toolkit redraw – common in GUI APIs.
    - Also used in Java critical realtime software – noHeapRealtimeThread.
- Issue: How to provide verification regarding race conditions?
  - Understanding the limits of testing and inspection
  - Fluid approach: **Analysis-Based Verification**

## Demonstration

Scenario 3: responsibility, aggregation

**Apache log4j**
BoundedFIFO

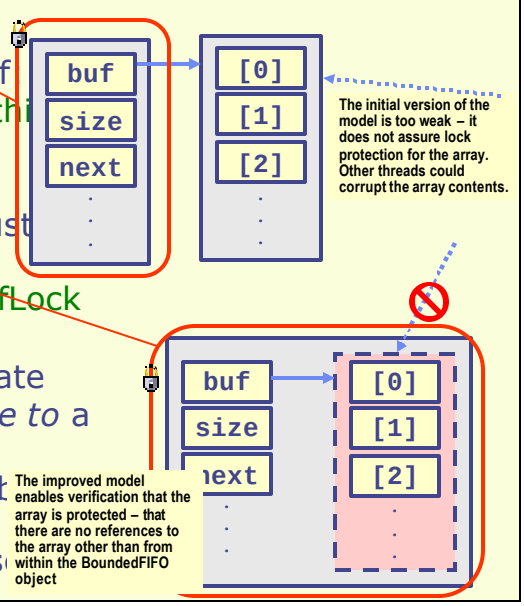
## Demonstration

Apache Log4j BoundedFIFO: Model semantics

Express lock policy
- Object protects itself
  @lock BufLock is thi
  protects Instance

- Caller of method must
  acquire lock:
  @requiresLock BufLock

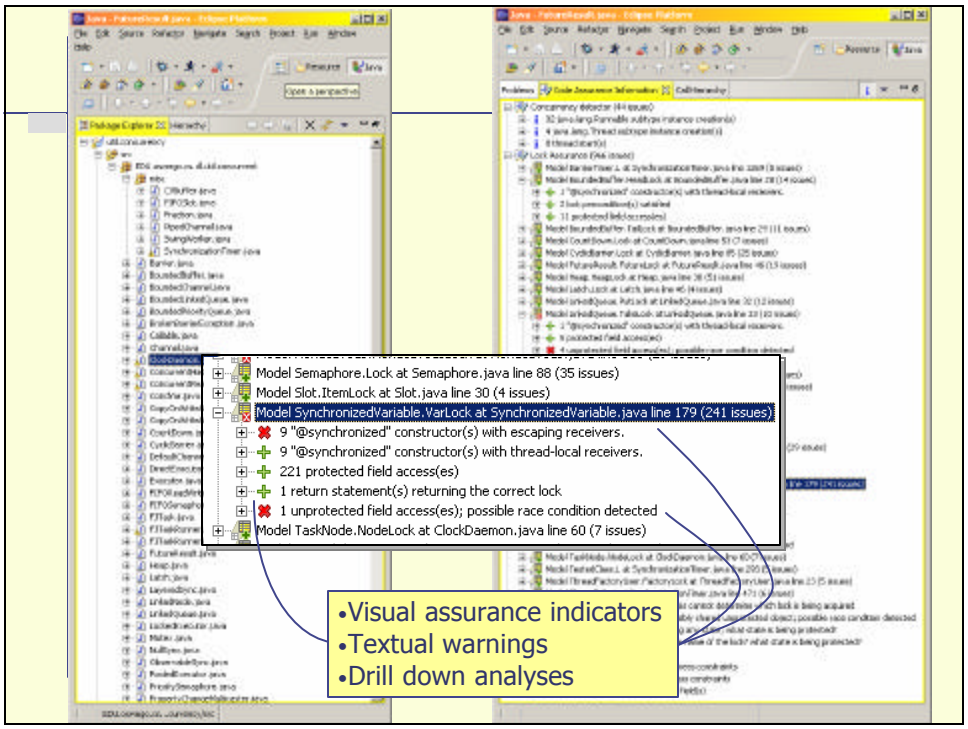
Aggregate subsidiary state
- *Warning*: A *reference to* a delegate
  object is protected, b
  not the
  referenced object its



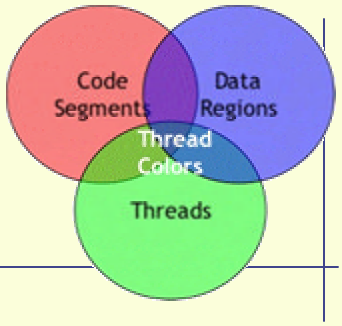
## Demonstration

Scenario 4: mid-scale, JMM

util.concurrent



## Scenario 6: Non-lock concurrency




AWT and GraphLayout
(Dean Sutherland)

## AWT Threading Model

Annotations associated with **`java.awt`**

```
/*@ color group AWT_Thread_Usage
*       color AWT
*       color Compute
*       thread count AWT <= 1
*       incompatibleColors AWT, Compute
* end AWT_Thread_Usage */
```

*Don't steal the redraw thread.*
*Don't try to call paint yourself.*

## Annotation required in AWT

- What library methods require the AWT color?

```
//@requiresColor AWT
    java.applet.Applet
    java.awt.event.MouseListener
    java.awt.event.MouseMotionListener
    java.awt.Graphics
    java.awt.Button
    java.awt.Checkbox
```

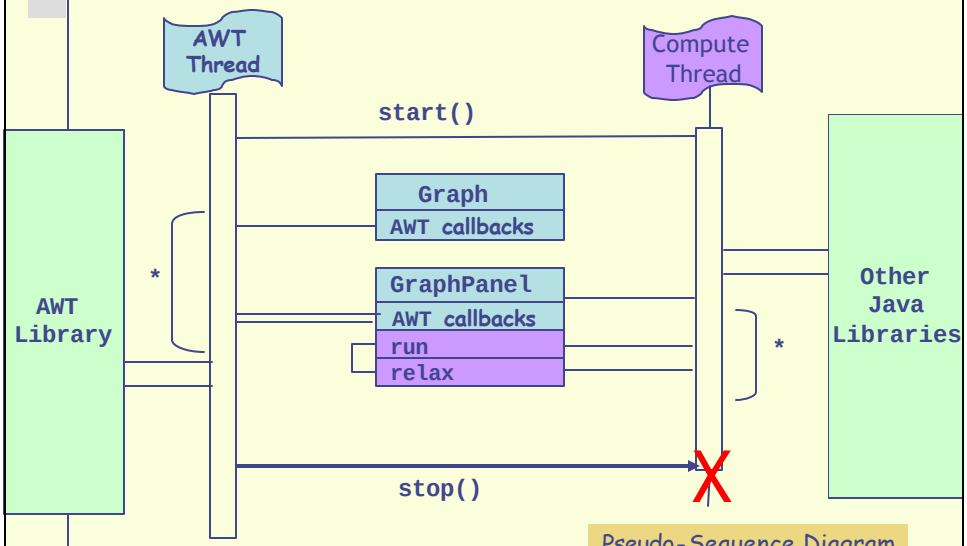
```
//@requiresColor AWT
    java.awt.Component
     on methods
        update, mouseDown,
        mouseClicked,
        mousePressed,
        mouseEntered, mouseExited,
        mouseDragged, mouseMoved,
        paint
```

- What library classes are "Don't Care?"

```
//@transparent
    java.util.StringTokenizer
    java.lang.Number
    java.lang.String
    math.*
    thread.*
```

Note: Only 24 AWT annotations required for GraphLayout.

## GraphLayout Thread LifeCycle




Pseudo-Sequence Diagram
(no temporal ordering implied)

## Result of initial analysis – **no annotation** in GraphLayout

**GraphPanel**

| |
|---|
| GraphPanel |
| findNode |
| addNode |
| addEdge |
| Run# |
| relax |
| paintNode* |
| Update* |
| mouseClicked* |
| mousePressed* |
| mouseReleased* |
| mouseEntered* |
| mouseExited* |
| mouseDragged* |
| mouseMoved* |

**Graph**

| |
|---|
| Graph |
| Init+ |
| Destroy+ |
| Start+ |
| Stop+ |
| getAppletInfo+ |
| getParameterinfo+ |
| actionPerformed* |
| itemStateChanged* |

AWT

Compute

[ Transparent ]

Can't Tell

Elsewhere…

| |
|---|
| Java.lang.string |
| Java.lang.Number |
| … |

- **22 of 24** methods colored successfully
- **AWT_Thread_Usage** not verified
  Specifically -- unable to verify annotation
  **Compatibility AWT != Compute**
  because some methods are not colored

*Only to be used as ****AWT** *or +****Applet*** *or #****Thread*** *callbacks*

## Result of analysis – with model in GraphLayout

**GraphPanel**

| |
|---|
| GraphPanel |
| findNode |
| addNode |
| addEdge |
| Run# |
| relax |
| paintNode* |
| Update* |
| mouseClicked* |
| mousePressed* |
| mouseReleased* |
| mouseEntered* |
| mouseExited* |
| mouseDragged* |
| mouseMoved* |

**Graph**

| |
|---|
| Graph |
| Init+ |
| Destroy+ |
| Start+ |
| Stop+ |
| getAppletInfo+ |
| getParameterinfo+ |
| actionPerformed* |
| itemStateChanged* |

AWT

Compute

[ Transparent ]

Can't Tell

Elsewhere…

| |
|---|
| Java.lang.string |
| Java.lang.Number |
| … |

- **All** methods colored successfully
- **AWT_Thread_Usage** verified

*Only to be used as ****AWT** *or +****Applet*** *or #****Thread*** *callbacks*

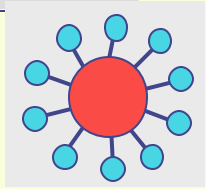
## Fluid: published results

**Annotation, analysis, and tool publications**

- SOCP '05 (to appear)
- POPL '05
- CSJP '04
- OOPSLA '03 Eclipse eXchange
- PASTE '02
- ICSE '02
- Software—Practice and Experience '01
- ECOOP '99
- ICSE '98

http://**www.fluid.cs.cmu.edu/**

## Assured Development: Hub and spokes



- Hub – Fluid core infrastructure
  - Representations, core analyses, etc.
  - Interactive online, build-based offline
  - Verification support
    - Proof management and assertion propagation
    - Effects, aliasing, regions, permissions, etc.
- Spokes – attribute-specific analyses (examples: present, future ):
  - Assurance:
    - Races (lock)
    - Concurrency (non-lock)
    - Deadlock
    - Modular non-lock
    - Real time
    - ? Data boundaries
    - ? Null pointers
    - ? Framework compliance
    - ? Custom API compliance
    - ? Performance
  - Indicators
    - Appropriate typing
    - Exceptions ignored
    - Concurrency finder
    - Thread effects
    - ? Management
    - ? Modeling metrics
    - ? Assurance metrics
    - ? Query support
    - ? Query-based modeling

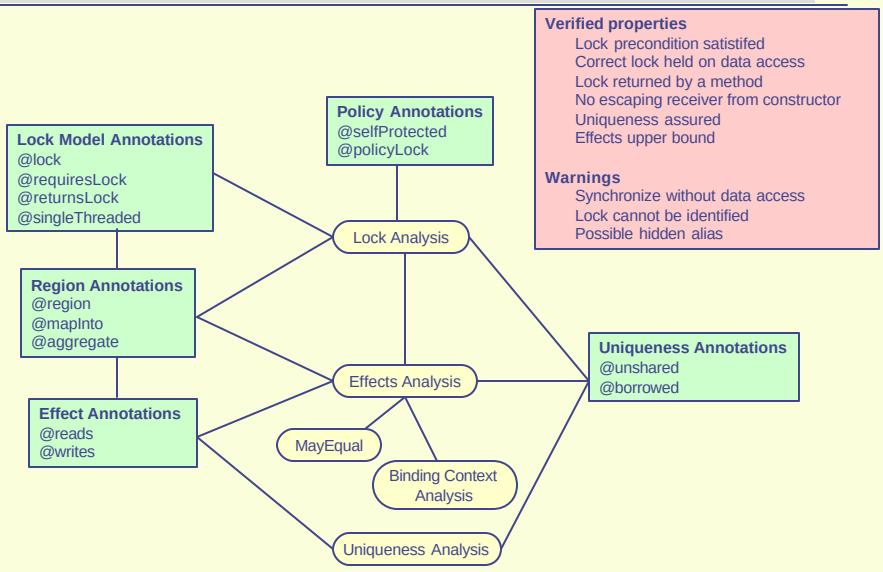
## Other expressions of design intent

Familiar informal expressions of design intent:

- "Only these two subclasses" (even though it is public)
- "Not intended to be aliased" (but a reference can be "borrowed")
- "The object is immutable" (but lazy calculation and caching are ok)
- This state may be accessed outside this module only thru this gateway
- The state may be written from here only, but read more widely
- The AWT thread is the only thread that can call paint, update, mouseMoved, …
- The exception caught here is thrown only from these two places

**Challenge**:
Capturing intent while satisfying adoptability concerns

## Interacting analyses – a simplified view

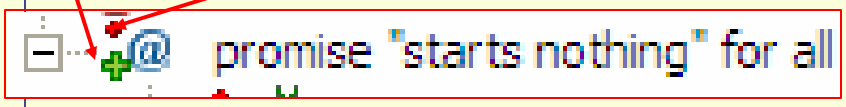


## Scaling up: expressing larger models

Scoped promises
(Tim Halloran)

## The "red dot" – composable partial results

Consistency of model and code is **contingent** on a "trusted" result




- Support for separate development and modular assurances
  - Library/framework code
  - Externalized code
  - Outsourced code
  - Expectations regarding code not yet written

## Motivation

- Assurance at scale:

- Problem 1: Parsimonious model expression
  - Example:
    - Log4j's **`DateFormatManager`**

- Problem 2: API contracts
  - Examples:
    - Log4j's **`BoundedFIFO`**
    - Java GUIs **`AWT, SWING, etc.`**

Logging Services

**`org.apache.log4j.lf5.util.`**
**`DateFormatManager`**

## Using @promise

```
25  *
26  * @lock L is this protects Instance
27  * @promise "@singleThreaded" for new(**)
28  * @promise "@borrowed this" for new(**) | *(**)
29  */
```

Code Assurance Information
- Lock Assurance (38 issues)
  - Model DateFormatManager.L at DateFormatManager.java line 26 (37 issues)
    - 8 "@synchronized" constructor(s) with thread-local receivers.
    - 29 protected field access(es)
- Uniqueness Assurance (62 issues)
  - 23 method body(s) respect uniqueness constraints
  - 39 method call(s) respect uniqueness constraints

## Part 2: Teamwork, contracts, APIs

- Assurance at scale:

- Problem 1: Parsimonious model expression
  - Example:
    - Log4j's **`DateFormatManager`**

- Problem 2: API contracts
  - Examples:
    - Log4j's **`BoundedFIFO`**
    - Java GUIs **`AWT, SWING, etc.`**

Logging Services

**`org.apache.log4j.helpers.BoundedFIFO`**

## Solution #1

- Auxiliary file (xml) for external code

*java.lang.System.promises.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE package SYSTEM "promises.dtd">

<package name="java.lang">
  <class name="System">
    <method name="arraycopy"
            params="java.lang.Object,int,java.lang.Object,int,int">
      <promise keyword="borrowed" contents="arg0, arg2"/>
      <promise keyword="reads" contents="arg0.Instance"/>
      <promise keyword="writes" contents="arg2.Instance"/>
    </method>
  </class>
</package>
```

Logging Services

## Solution #2

org.apache.log4j.helpers.
**BoundedFIFO**

```
/**
 * @lock BufLock is this protects Instance
 * @promise "@borrowed this" for new(**) | *(**)
 * @promise "@singleThreaded" for new(**)
 * @promise "@requiresLock BufLock" for *(**) & !(resize(**))
 * @assume "@borrowed arg0"
 *   for "System.arraycopy(Object, int, Object, int, int)"
 */
public class BoundedFIFO {
  /** @unshared @aggregate [] into Instance */
  LoggingEvent[] buf;
  ...
  synchronized public void resize(int newSize) {
  ...
  System.arraycopy(buf, first, tmp, 0, len1);
  ...
  }
}
```

(aside) Apache Bug 26224