A 10-year Perspective on Software Engineering Self-Adaptive Systems

SEAMS 2013
David Garlan

May 2013
San Francisco, CA

A 10-year Perspective ...

- In 2002 Alex Wolf, Jeff Kramer and I organized the first ACM SIGSOFT Workshop on Self-Healing Systems (WOSS’02)
  - Charleston, South Carolina (at FSE-10)
  - 35 participants, 22 presentations
  - 2-days
  - Evolved into today’s SEAMS
- It seems appropriate to reflect on what has gone on since that event
  - What were the challenges then?
  - What are they now?
  - Has there been progress?

© David Garlan 2013
Talk Outline

- Partial history of the field
  - Precursors; WOSS’02; what followed
- Personal reflections on the Rainbow System
  - What we did right; what we got wrong
- Challenges moving forward
  - Some open problems and ideas for progress

Timeline

WOSS: Workshop on Self-Healing Systems
SEAMS: SW Engineering for Adaptive and Self-Managing Systems
“The goal of this workshop is to bring together researchers and practitioners from many [areas] to discuss the fundamental principles, state of the art, and critical challenges of self-healing systems.

... we intend to focus on the software engineering aspects, including the software languages, techniques and mechanisms that can be used to support dynamic adaptive behavior.”
ACM SIGSOFT*  
Workshop on  
Self-Healing Systems  
(WOSS’02)  

November 18-19, 2002  
Charleston, SC  

* With partial support from the NASA High Dependability Computing Project

Workshop Logistics

Sessions organized by affinity groups
  ■ 3-4 papers per 1.5 hour session

Session chairs, as noted on program
  ■ See your chair before your session

Emphasis on discussion
  ■ 10-minute research overviews from each paper

Half-hour breaks between sessions

Joint lunch on Monday

Open (uncommitted) sessions at end of each day (for extra discussion, topics that don’t have another home, etc.)
Software Engineering Today

Common assumptions
- Known and stable system requirements
- Known and stable operating environment
- Control over the development of assembled parts
- Development time and run time are completely separate
- Systems can be taken “down” for “maintenance”

Consequences
- Focus on improving development time processes, techniques, notations
- Provide high assurance through testing, rigorous specification, modeling, verification, etc.
- Expect end users to do manual installation and upgrades

Isn’t This Good Enough?

Increasingly, systems
- are composed of parts built by many organizations
- must run continuously
- operate in environments where resources change frequently
- are used by mobile users

For such systems, traditional methods break down
- Exhaustive verification and testing not possible
- Manual reconfiguration does not scale
- Off-line repair and enhancement is not an option
What Has to Change?

Goal: systems automatically and optimally adapt to handle
- changes in user needs
- variable resources
- faults
- mobility

But how?
Answer: Move from open-loop to closed-loop systems

Many Approaches

Programming language support
Algorithms (e.g., self-stabilizing, machine learning)
Architecture-based adaptation
Operating systems support
Domain-specific techniques (e.g., distributed databases, pub-sub architectures, ...)
Adaptable middleware
Support for user mobility
Fault tolerant system design (e.g., graceful degradation)
Biologically-inspired models
Inferring correct system behavior through observation
Why Have a Workshop?

Understand the relationships between these different approaches
Identify the software engineering challenges and opportunities
Create a common vocabulary (or possibly a reference model)

Affinity Groups (in order of appearance)

Architecture-based adaptation
Systems: operating systems, distributed systems, databases
Middleware & mobility
Programming languages
User-centric approaches: requirements specification, inference
New paradigms: neural nets, biologically-inspired computing, homeostatic systems
Outcomes

- It was a motley crew representing a wide range of areas of expertise
- We had a lot of fun
- We decided to do it again...

Timeline

- IBM Autonomic
- ICAC...
- TASS
- SASO...
- IJAC
- SOAR...
- DASADA
- WOSS’02
- WOSS’04
- SEAMS’06...
- We Are Here
- SEAMS’013...

WOSS: Workshop on Self-Healing Systems
SEAMS: SW Engineering for Adaptive and Self-Managing Systems

DASADA: Dynamic Assembly for System Adaptability, Dependability, and Assurance

ICAC: Intl. Conf. on Autonomic Computing
SASO: Self-Adaptive and Self-Organizing Systems
SOAR: Self-Organizing Architectures

IBM Autonomic

TASS: ACM Transactions on Autonomous and Adaptive Systems
IJAC: International Journal of Autonomic Computing
Forces Behind the Expansion

- **New system requirements**
  - Maintain availability and utility in the presence of variable environments, faults, attacks, changing requirements, mobility, reliance on systems outside our direct control, ...

- **Traditional solutions increasingly inadequate**
  - Humans are expensive and error prone
  - Embedded mechanisms (e.g., timeouts, exceptions) didn’t work

- **Recognition that control systems are not just for hardware**
  - Same principles can be applied to software systems

---

IBM MAPE-K

![Diagram of IBM MAPE-K](image-url)
The Google File System


Summary – Part 1

- Since the first WOSS in 2002 the field of self-adaptive systems has exploded, driven by technological and economic forces
  - Conferences
  - Publications
  - Journals
  - Industrial attention
- From the outset it represented an area of convergence for many disciplines
  - Fault-tolerance, architecture, AI, etc.
- But with a distinct software engineering flavor of its own
Talk Outline

- Partial history of the field
  - Precursors; WOSS’02; what followed
- Personal reflections on the Rainbow System
  - What we got right; what we got wrong
- Challenges moving forward
  - Some open problems and ideas for progress

Rainbow Goals

- Engineer self-adaptation to support:
  - Cost-effectiveness (relative to handcrafted solutions)
  - Automation of routine adaptations performed by humans
  - Legacy systems
  - Domain-specific adaptations
  - Ease of changing adaptation strategies and policies
  - Reasoning about the effects of self-adaptation actions and strategies
Rainbow Approach

- A framework that
  - Allows one to add a **control layer** to existing systems
  - Uses **architecture models** to detect problems and reason about repair
  - Can be **tailored to specific domains**
  - Separates concerns through **multiple extension points**: probes, actuators, models, fault detection, repair
- A language (**Stitch**) for programming repair actions that
  - Addresses self-adaptation concerns ...

---

The Rainbow Framework

![Rainbow Framework Diagram]

© David Garlan 2013
Rainbow Framework Overview

- **System Layer**
  - Strategy Executor
  - Adaptation Manager
  - Architecture Evaluator

- **Architecture Layer**
  - Model Manager
  - Architecture Evaluator

- **Translation Infrastructure**
  - System API
  - Resource Discovery
  - Probes
  - Gauges

- **Target System**
  - System Layer
  - Effects

**Extracts system information**
**Relates system info to model**
**Decides on the best adaptation**
**Carries out that adaptation**
**Bridges abstraction (aggregate info up, map action down)**
**Changes state in target system**
**Customization points**

© David Garlan 2013
Self-Adaptation Example: Znn.com

Adaptation Condition: client request-response time must fall within threshold

Possible actions:
- enlistServers
- dischargeServers
- restartWebServer
- lowerFidelity
- raiseFidelity

Possible actions:
- restartLB
**Stitch: A Language for Specifying Self-Adaptation Strategies**

- **Control-system model**: Decision tree in which election of next action in a strategy depends on observed effects of previous action
- **Value system**: Utility-based selection of best strategy allows context-sensitive adaptation
- **Asynchrony**: Explicit timing delays capture "settling time"
- **Uncertainty**: The effect of a given tactic/strategy is known only within some probability
Strategy Selection

- Given:
  - Quality dimensions and weights (e.g., 4)
  - A strategy with
    - N nodes
    - Branch probabilities as shown
    - Tactic cost-benefit attributes

- Propagate cost-benefit vectors up the tree, reduced by branch probabilities
- Merge expected vector with current conditions (assume: \([1025, 3.5, 0, 0]\))
- Evaluate quality attributes against utility functions
- Compute weighted sum to get utility **score**

Algorithm

Given tree \(g\) with node \(x\) and its children \(c\):

\[
EAAV(g) = sysAV + AggAV(root(g))
\]

\[
AggAV(x) = cbav(x) + \sum_{c \text{ prob}(x,c)} AggAV(c)
\]

*Znn.com: Rainbow Customizations (Part 2)*

Objectives: timely response (uR), high-quality content (uF), low-provision cost (uC)

weights:

- uR: 0.3
- uF: 0.4
- uC: 0.2
- uSF: 0.1

utility:

- 0: 1.00
- 500: 0.90
- 1500: 0.50
- 4000: 0.00

Model Manager MM
Architecture Evaluator AE
Adaptation Manager AM
Strategy Executor SX

© David Garlan 2013
**Znn.com Experiment Run Results**

### Aggregate JMeter Data

<table>
<thead>
<tr>
<th>URL</th>
<th>#Samples</th>
<th>Average (ms)</th>
<th>Median (ms)</th>
<th>90% Line (ms)</th>
<th>Min (ms)</th>
<th>Max (ms)</th>
<th>Errors</th>
<th>Throughput (KB/sec)</th>
<th>Error%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0415K No Rainbow</td>
<td>1200</td>
<td>798</td>
<td>1983</td>
<td>37514</td>
<td>93</td>
<td>32102</td>
<td>0.00%</td>
<td>8.4/sec</td>
<td>8.9%</td>
</tr>
<tr>
<td>0415L SimpleReduce1</td>
<td>1200</td>
<td>1502</td>
<td>16</td>
<td>2187</td>
<td>0</td>
<td>2202</td>
<td>0.17%</td>
<td>29.5/sec</td>
<td>78.2%</td>
</tr>
</tbody>
</table>

% of K

| #K > 10sec | 217     |
| #L > 10sec | 80      |
| #K > 1 sec | 1120    |
| #L > 1 sec | 228     | 26.6%     |

### System Adapts

Data shows that our adaptation approach improves overall system performance.

- **Control run**
  - Latency = 2 secs

- **Adaptation run**
System Administration Evaluation

- **Sys-admin interview**
  - **Results:** Stitch concepts seem natural fit for sys-admin routines
  - Methodology: priming, interview, compose Stitch script from scenarios
  - White problem scenarios: scripts represented in Stitch
  - Almossawi problem scenarios: structure matches Stitch strategies

- **Analysis using CMU sys-admin example: Netbwe**
  - **Results:**
    - Rainbow captured adaptation concerns
    - Stitch hoisted policies buried in Perl code
  - Distinguishable adaptation tasks
    - Core commands as operators
    - Coarser-grained sequence of commands (step) with conditions of applicability and intended effects
    - Adaptations with intermediate condition-actions and observations

Rainbow Reflection: What we got right

- **Embodies an explicit control systems architecture**
- **Supports model-based adaptation (in this case through architectural models)**
  - Models are updated by gauges and form the basis for all decision making
- **Framework allows domain specialization**
  - Style of system, repertoire of strategies, ...
- **Repair language (Stitch) incorporates:**
  - Control perspective guiding each step
  - Utility to select “best” strategy
  - Timing to account for asynchrony
  - Probabilities to handle uncertainty
Rainbow Reflection: What we got wrong

- Fixed set of strategies
  - What do you do if no strategy can help?
- Tactic outcomes specified in utility space
  - Limits amount of reasoning one can do about strategies
- Repair triggered by constraint violation
  - Does not easily accommodate homeostatic adaptation
- Humans not in the loop
  - Not prohibited, but not easily supported
- Complexity in number of plug-ins for installation
  - Deployment is difficult

Summary – Part 2

- Rainbow attempted to capture the self-adaptive control loop through
  - a framework that can be specialized for specific domains
  - a language for programming adaptations
- Recognized the first class nature of
  - Control, Timing, Uncertainty, Utility, Analysis
- But left many issues unaddressed or inadequately supported
  - Humans in the loop, dynamically generated strategies, tactic specification, and deployability
Talk Outline

- Partial history of the field
  - Precursors; WOSS’02; what followed
- Personal reflections on the Rainbow System
  - What we got right; what we got wrong
- Challenges moving forward
  - Some open problems and ideas for progress

Some Challenges

1. Uncertainty
2. Combining Reactive and Deliberative Adaptation
3. Humans in the Loop
4. Architecting for Adaptability
5. Proactivity
6. Concurrency, preemption, synchronization
7. Distribution
1. Uncertainty

- Uncertainty is everywhere
  - Monitor: Sensing is imprecise and partial
  - Analyze: Detection and diagnosis are often ambiguous
  - Plan: The effect of a tactic/strategy is cannot be accurately predicted
  - Execute: The system may not behave the way we expect, or may be subject to other influences outside our control
  - Knowledge: Run-time models are only approximate
  - Environment: Hard to predict or control what’s outside the system (loads, attacks, ...)

- How do we bring this under control without overwhelming our ability to reason about the adaptation?

Ideas ...

- Adaptive probing
  - Reduce uncertainty when problems occur

- Adaptive diagnosis
  - Expand window of observation when cause is uncertain

- Machine learning
  - Improve our understanding over time

- Make uncertainty first class in our models (both design models and run-time models)
  - Exploit emerging probabilistic model checking tools
2. Combining Reactive and Deliberative

- Reactive adaptation (ala Rainbow) is efficient and supports design-time analysis but may not cover all situations
- Deliberative adaptation allows one to reason a way to a solution in unfamiliar situations, but may be slow
- Can we combine these?
  - Supervisory planning to set longer-term goals (ala Magee and Kramer)?
  - In parallel and collaboratively?
System 2

Analytical
Deliberative
Slow
High resource consumption
Conscious
More often right

System 1

Intuitive
Reactive
Fast
Low resource consumption
Unconscious
Useful if not always right

Example: 24 * 17
Example: 2 + 2

Example: Arrows

© David Garlan 2013
3. Humans in the Loop

- Taking deliberative approaches even further, how can we involve humans in the adaptation loop?
  - Providing situational awareness to operators
  - Using them to improve adaptation by combining autonomic and human guidance for mixed-initiative control
  - Learning from human behavior
- How does that impact our ability to reason about adaptive systems?
  - Humans add an even larger source of uncertainty, compounding Challenge 1

Ideas ...

- Perhaps approach the problem by considering humans in various roles in the control loop:
  - Sensors (providing inputs about the state of the system and the environment)
  - Actuators (carrying out actions that may not be automatable)
  - As knowledge/model augmentation
  - As decision makers (developing new tactics/strategies)
- Learn from new breed of interactive robots
4. Architecting for Adaptability

- It is routine now to consider various “ilities” when architecting systems
- What does it mean to architect a system (under control of an autonomic layer) for adaptability?
  - Example: monitorability
  - Example: controlability
- What tradeoffs are involved and how can one balance these?
  - Example: monitorability may impact performance
  - Example: controlability may impact security

© David Garlan 2013

Ideas ...

- Make adaptability part of architecture design reviews
- Create a catalog of adaptability-enhancing design patterns/tactics
- Some may be generic
  - Example: pub-sub mechanisms support monitorability of communication
- Others will be domain-specific
  - Example: Stateless servers support flexible server reconfiguration in web-oriented domains

© David Garlan 2013
5. Proactivity

- Today most self-adaptive systems are reactive
  - A problem occurs, and the system addresses it
- But in many cases it may be better to do things before the problem occurs
  - Example: Adapt the number of servers in anticipation of high anticipated load
  - Example: “Moving target” approaches to security
  - Example: periodic rebooting of suspicious subcomponents
- How do we rationally balance the cost and benefits of a proactive approach?

Ideas ...

- Change the basis for strategy selection from instantaneous utility to aggregate utility
- Develop better ways to predict future behavior
  - Game theory for adversarial environments
  - Predictive calculi (cf. Poladian thesis)
  - Use machine learning
- Learn from the “planning under uncertainty” community
  - They have insight into when is it better to balance short-term pain with long-term gain
6. Concurrency, Preemption, Synch

- Many autonomic systems execute a single adaptation at a time
  - Easier to reason about
- But many potential benefits to concurrent adaptation
  - Performance through parallelism
  - Rapid response when new opportunities arise
- How do we make this manageable?
  - Guarantee concurrent adaptations don’t interfere
  - Know when to terminate a long-running but unproductive adaptation
  - Interrupt a low-priority adaptation with a high-priority one

7. Distribution

- Centralized adaptation architectures simplify control but may not scale
  - Systems of systems built by different parties
  - Ultra-large scale systems with no central control or authority
- How can we coordinate adaptation among many managed systems?
Ideas ...

- Explore alternative integration architectures
  - Hierarchical – e.g., master-slave
  - Cooperative – e.g., pub-sub based
  - Nearest-neighbor topologies – e.g., combined with self-stabilizing algorithms
- Social networks in which nodes are managed systems
  - Systems are part of “social” ensembles with similar interests and trust relations
Other Challenges

- **Assurances**
  - Resiliency of adaptive mechanisms themselves
  - Run-time simulation, model checking, testing

- **Cyber-Physical Systems**
  - Combining physical and software control

- **Security**
  - Many flavors
  - Adversarial nature of the environment

- **Benchmark problems and metrics**
  - Znn is a start, but need more

- **Compositionality**

---

Other Challenges

- **Reconciling multiple mechanisms**
  - Low and high level
  - Reactive and proactive
  - Immediate and deliberative
  - Automated and human-directed
  - Centralized and distributed
Summary

- The field has grown substantially from the early days of WOSS’02 to the point where it is a relatively well-established subfield
- First generation systems, like Rainbow, have taught us a lot
- Many challenges remain

Acknowledgements

- Funders
  - DARPA: Initial investigations
  - NSF: Refinements and currently Fault Localization
  - CMU-Portugal: Analysis, Planning
  - Samsung Electronics: Applications to manufacturing control
  - NSA, Navy, Army
- Graduate students and colleagues (many)
  - Especially Owen Cheng and Bradley Schmerl
- More information: www.cs.cmu.edu/~able
The End

SEAMS

- SEAMS 2012 in Zürich, Switzerland
- SEAMS 2011 in Hawaii, USA
- SEAMS 2010 in Cape Town, South Africa
- SEAMS 2009 in Vancouver, British Columbia, Canada
- SEAMS 2008 in Leipzig, Germany
- SEAMS 2007 in Minneapolis, Minnesota, USA
- SEAMS 2006 in Shanghai, China
- WOSS 2004 in Newport Beach, CA
- WOSS 2002 in Charleston, SC
Select References

- **Rainbow**

- **Diagnosis**

- **Prediction**