An Architectural Framework for Quality-driven Adaptive Continuous Experimentation

Miguel Jiménez, Luis F. Rivera, Norha M. Villegas, Gabriel Tamura, Hausi A. Müller, Nelly Bencomo

{ miguel, rivera, hausi }@uvic.ca
{ nvillega, gtamura }@icesi.edu.co
nelly@acm.org

2019 IEEE/ACM Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution

WORK IN PROGRESS

May 2019
Remember these logos?
Skynet

Pools data from battle units, develops tactics and coordinates attacks

Controls everything which contains a Cyberdyne Systems CPU

Using the blueprints, designs and tests models built by Cyberdyne Systems

Manufactures battle units, occasionally updating them or producing more advanced models

Source: https://terminator.fandom.com/wiki/Skynet
Skynet

Pools data from battle units, develops tactics and coordinates attacks.

Controls everything which contains a Cyberdyne Systems CPU.

Skynet was doing Software Engineering at Run-time.

Manufactures battle units, occasionally updating them or producing more advanced models.

Source: https://terminator.fandom.com/wiki/Skynet
Continuous Evolution

Continuous Experimentation

- Data-driven adjustment of behavior—Evidenced-based engineering
- Reproduce results with high confidence
- Useful for quality-driven experimentation
- How to enable the system to plan and run experiments at run-time?


M. Jiménez et al. DevOps Round-trip Engineering: Traceability From Dev to Ops and Back, 2018
Quality-driven Experimentation

• Leverage CE and software patterns to improve the design

• We aim to reuse knowledge from system design and operation that is traditionally exploited at design-time

• How to represent patterns? Plan and run experimental trials at runtime? What if experiments are too expensive? (time and resources)

M. Jiménez et al., DevOps’ Shift-Left in Practice: An Industrial Case of Application, 2019
1. Model-driven Software Patterns

- **Continuous Design**
  - Autonomic managers for high-level engineering activities

- **Experimentation**
  - Models to plan low-level engineering activities

- **Patterns**
  - Models to coordinate and document software evolution

- **Software Changes**
  - Models to represent the system's state, specification notations and target platforms

- **Quality of Service**

- **Decision Documentation**

- **Software Application**
  - Services & Components

- **Virtual Infrastructure**
  - Infrastructure as Code

- **Physical Infrastructure**
  - Configuration as Code

- **Software Deployment**
  - Deployment & Configuration

- **Virtual Infrastructure**
  - Infrastructure as Code

- **Physical Infrastructure**
  - Configuration as Code

- **Software Application**
  - Local Code Repository

- **Software Deployment**
  - Local Code Repository

- **Virtual Infrastructure**
  - Local Code Repository

- **Physical Infrastructure**
  - Local Code Repository

- **Run-time Agent**
  - MART
  - Event Listener

- **Run-time Agent**
  - MART
  - Event Listener

- **Run-time Agent**
  - MART
  - Event Listener

- **Run-time Agent**
  - MART
  - Event Listener

- **Run-time Agent**
  - MART
  - Event Listener

- **Run-time Agent**
  - MART
  - Event Listener

Legend:
- **Data flow**
- **Artefact**
- **Component**
- **Environment**
2. A metamodel for experimental design

- Context information for explainability
- Implementation techniques
- Deployment strategies
- Monitoring (not trivial)
- Traceability
- Repeatability
Background Reference Models for Self-Adaptive Software

MAPE-K

DYNAMICO

Villegas et al. DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems, 2013

IBM. An architectural blueprint for autonomic computing, 2006
3. An Architecture for Quality-driven Adaptive Experiments

- Separation of concerns:
  - high-level experimental design objectives and plan
  - Infrastructure variants
  - Software variants
The Experimentation Feedback Loop (E-FL)

- Satisfaction of the high-level experimental design objectives
- **M**: experimentation results
- **A**: detects need to create / update experimental design
- **P**: synthesizes design for infrastructure and software variants
- **E**: instantiates P-FL and C-FL
- Dynamic hypotheses
The Provisioning Feedback Loop (P-FL)

- **M**: infrastructure metrics
- **A**: aggregates and classifies metrics into partial / complete results.
  
  Also, re-synthesize?
- **P**: translates experimental design into provisioning plan
- **E**: deploys the variants and sets infrastructure data in C-FL
The Configuration Feedback Loop (C-FL)

- **M**: software metrics
- **A**: aggregates and classifies metrics into partial / complete results.
  
  Also, re-synthesize?
- **P**: translates experimental design scenarios into deployment and reconfiguration plan
- **E**: deploys the original software to planned infrastructure variant.
  
  Adapts the running system to planned variant.

Deploys monitors and starts the testing technique in provisioned (adapted) infrastructure.