Week 4: Evaluate Constructive Research Designs

Nate Bachmeier

TIM-7241:Constructive Research Design

May 16, 2021

Northcentral University

# Evaluate Constructive Research Designs

Choose three constructive research papers from the Northcentral University Library. Then evaluate how artifacts integrate into their research designs. An assessment summary is present in Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Cinnamon | Collaborative IoT | I2B Programming |
| Authors | Arif et al. (2021) | Banerjee & Chandra (2019) | Chen et al. (2019) |
| What is the problem | Instrumenting binaries is complex | Too many ad hoc tasks for ML on IoT data | Customizing intelligent buildings is challenging |
| Who cares about the solution | Software engineers that integrate testing and debugging tools | Software engineering managers and data science practitioners | Building managers and operators |
| What contribution came from creating this artifact | Lower barrier to entry into instrumentation | Methodological processes for implementing best practices | 1. Lower barrier to entry into programming smart buildings 2. Authors propose DSL requirements |
| When/where/why was the artifact proposed | DSL for describing instrumentation and LLVM compiler extension | Reference architecture and a case study implementing it | Most systems rely on C scripts which are difficult for building managers to implement |
| Where is this applicable | Quality assurance and troubleshooting | Large corporations with decentralized teams (e.g., IIoT and Health Care) | Industrial IoT, data centers, and factories |
| Any strengths or weaknesses | 1. The language is concise and requires a 5-10% code size vs. Dyninst and Janus. 2. Scripts are language agnostic 3. It does not align with existing standards | 1. An abstract model that can integrate into existing environments 2. No data within the article 3. The first collaborative platform for IoT data (pg. 46) 4. No streaming data support | 1. It uses a visual drag/drop design 2. It would be challenging to debug 3. Focuses on simplicity over functionality |
| What testing took place | Implements five different scenarios to confirm the DSL’s *expressiveness* | The architecture is available within a health care facility.  They discuss several scenarios and their processes | There are three metrics:   1. Training Time 2. Programming Time 3. Completion degree |
| What data collection and processing methods utilized | Implements the five scenarios in Dyninst, Janus, and Pin  Next, an assessment of code length  There are no details on runtime efficiency | There are process diagrams but no data in the article | The researchers use four participants to complete two apps |

# Cinnamon: DSL for Binary Profiling and Monitoring (2021)

Software engineers need to extend application complication processes for numerous use-cases. These scenarios include code coverage instrumentation, performance profiling, injecting Aspect-Oriented Programming (AOP), and discovering runtime vulnerabilities. Arif et al. (2021) state that existing code instrumentation tools (e.g., Dyninst and Janus) are cumbersome, non-generalizable, and involve significant engineering costs. For instance, building a simple use-after-free utility in Dyninst necessitates 260 lines of low-level code. This high barrier-to-entry prevents software engineers and quality assurance teams from building custom instrumentation solutions.

Arif et al. (2021) study these challenges through Cinnamon, a Domain-Specific Language (DSL) that simplifies binary profiling and monitoring tasks. The language integrates into the LLVM compiler chain and its Internal Representation (IR) layer. This approach makes the instrumentation scripts reusable across different programming languages (e.g., Java and C++). Further, it uses a concise vocabulary that reduces the instrumentation script’s codebase by 90% over Janus.

Cinnamon’s expressiveness and ease of use are the most critical components of success. The authors validate its flexibility across five different use-cases (e.g., instruction counting and loop coverage). First, they implement the complete solutions in Cinnamon, Pin, Dyinst, and Janus. Then an assessment of the code length confirms that their DSL is the best choice. However, this test plan is overly narrow in scope. Additional documentation needs to cover performance metrics (e.g., compilation time and runtime overhead). Without this information, it is challenging to determine the Total Cost of Ownership (TCO).

# Software Framework for Collaborative IoT Analytics (2019)

Software engineers and data scientist-practitioners need to perform numerous steps while operationalizing IoT data. Each sensor flows through an analytic pipeline that pre-processes, transforms, models, runs inferences, and finally visualizes the data. Large organizations typically have dozens (or hundreds) of pipelines with ownership decentralized across different teams. This scenario introduces challenges discovering, sharing, and collaborating results between business units.

Banerjee & Chandra (2019) present a generic management framework for preventing Data Lakes from becoming Data Swamps. Their artifacts come from assisting a health care facility to define standard patterns and practices for evolving data sets. Afterward, the two researchers generalized the reference architectures to increase applicability for more industries (e.g., intelligent factories). Their solution lacks real-time streaming processing, an unexpected decision within IoT data processing.

The authors do not provide metrics or mechanisms for measuring success.