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.

Table 1: Evaluation Summary

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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 | 1. Too many tasks for ML on IoT data. 2. Avoiding Data Swamp | 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 a shared data lake | 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)

## Problem

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.

## Artifacts and Approach

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.

## Success Criteria and Testing

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). The authors could generate these statistics with an official performance benchmarking suite (e.g., database stress test).

# Software Framework for Collaborative IoT Analytics (2019)

## Problem

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 in discovering, sharing, and collaborating results between business units.

## Artifacts and Approach

Banerjee & Chandra (2019) present a generic management framework for preventing Data Lakes from becoming Data Swamps. Their artifacts assist a healthcare facility in defining 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 also claim to be the first team to implement collaborative features around IoT data (Banerjee & Chandra, 2019, p. 46). This statement is surprising since enterprise data catalogs are not a recent invention. Lastly, their artifacts would be better with additional details and business cases versus deferring to the reader.

## Success Criteria and Testing

The authors do not provide metrics or mechanisms for measuring success. If they had, then examining the amount of collaboration and reuse would be essential. Businesses could track those Key Performance Indicators (KPI) through support ticket volumes. Alternatively, the organization could construct a fictitious data catalog. Then test cases can confirm that those study participants can promptly find resources. It could also be beneficial to start with a well-defined list of business questions. This methodology helps drive more aggressive prioritization, which delivers the most value upfront.

# Graphical Programming for Insect Intelligent Building (2020)

## Problem

Insect Intelligent Buildings (I2B) is a modern design methodology for controlling Smart Buildings such as data centers, hospitals, and IIoT factories. Chen et al. (2019) state that facility managers and operators use I2B apps to control HVAC systems, regulate water pressure, and monitor other electromechanical equipment. Ideally, the managers and operators could build simple apps that meet their specific use-cases. However, most apps require C-programming to manage Programmable Logic Controllers (PLCs). This high barrier to entry necessitates trained software engineers for every change and discourages experimentation.

## Artifacts and Approach

Chen et al. (2019) propose a new visual programming language that enables drag-and-drop automation. This feat began with a deep analysis of nouns and verbs that make up the domain. Each of those essential constructs became a toolbox item within the graphical editor. Next, build visual workflows (e.g., a timer that activates hallway lights) that compiles into C code. Finally, the researchers solicit feedback on the tooling and incorporate recommendations into future versions.

## Success Criteria and Testing

The authors have three metrics quantifying success: training time, programming time, and completion degree. Their goal is to train new users within 15 minutes and then complete a medium complexity task within 30 minutes. Chen et al. (2019) validate these results with four participants split into two groups. Each participant has domain experience with I2B, not any programming concepts. Finally, end-users built the mock applications and could finish over 95% of the logic without instructor assistance.

Their visual programming language focuses on simplicity over functionality. This approach leads to several additional test cases around debugging and troubleshooting. For instance, does the solution have execution logs? Can the new developers utilize this information to repair broken workflows? In addition, intelligent Buildings use products from multiple vendors. How extensible is the language, and does it handle those various use-cases? Each of these questions drills into the suitability of their language.

# References

Arif, M., Zhou, R., Ho, H., & Jones, T. (2021). Cinnamon: A Domain-Specific Language for Binary Profiling and Monitoring. *International Symposium on Code Generation and Optimization* (pp. 103-114). Virtual Conference: IEEE/ACM. doi:10.1109/CGO51591.2021.9370313

Banerjee, S., & Chandra, M. (2019). A Software Framework for Procedural Knowledge-Based Collaborative Data Analytics for IoT. *International Workshop on Software Engineering Research & Practices for the Internet of Things* (pp. 41-48). Montreal, Quebec. Canada: IEEE/ACM. doi:10.1109/SERP4IoT.2019.00014

Chen, W., Yang, Q., Zhao, S., Xing, J., & Zhou, Q. (2020). Graphical Programming Language and Its Supporting Tool for Insect Intelligent Building. *Scientific Programming*, 1-18. doi:10.1155/2020/9634389