A Study on the Comprehensibility of Behavioral Programming Variants

Adiel Ashrov[1][a], Arnon Sturm[2][b], Achiya Elyasaf[2][c] and Guy Katz[1][d]
[1]*The Hebrew University of Jerusalem, Israel*
[2]*Ben-Gurion University of the Negev, Israel*

Keywords: Scenario-Based Programming, Behavioral Programming, Empirical Software Engineering.

Abstract: Behavioral Programming (BP) is a software engineering paradigm for modeling and implementing complex reactive systems. BP's goal is to enable developers to incrementally model systems in a manner naturally aligned with their perception of the system's requirements. This study compares two BP variants: classical, context-free BP, and the more recently proposed Context-Oriented BP (COBP). While BP advocates simplicity and modularity, COBP introduces context-aware constructs for handling context-dependent behaviors. A practical question arises: which variant leads to reactive systems that are more comprehensible for developers? Through a controlled experiment with 109 participants, we evaluated both variants across two dimensions: comprehension of execution semantics and identification of requirements from implementations. The results indicate that BP generally leads to better comprehension and higher confidence; while COBP demonstrates advantages in complex, context-dependent behaviors. These findings provide guidance for choosing between BP variants based on system complexity and context-dependent requirements.

1 INTRODUCTION

As reactive systems evolve, developers often need to enhance them. Some enhancements extend the system's functionality, for example, by adding a new game strategy or implementing a new feature (Ashrov and Katz, 2023). Other enhancements, sometimes referred to as *guard rules* or *override rules* (Katz, 2021), prevent the system from entering undesired states or performing unwanted actions — such as preventing illegal moves in a game or ensuring safety constraints in a robotic system. Both kinds of enhancements must be carefully integrated with existing behaviors, while maintaining the system's overall requirements and constraints, and this requires developers to comprehend the existing system before it can be modified.

Reactive systems are increasingly prevalent in domains like robotics (Kaelbling, 1987), IoT (Curasma and Estrella, 2023), and safety-critical applications (Dafflon et al., 2015).These systems grow more complex as enhancements are introduced over time. This, in turn, renders them increasingly difficult to comprehend, both at the code level (Storey, 2005; Brooks, 1983) and at the requirements level (Morandini et al., 2011), and thus, more difficult to enhance. The need to bridge this gap has led to the development of specialized paradigms for modeling reactive systems, with the goal of providing better comprehensibility.

Behavioral Programming (BP) (Harel et al., 2012b) is one such paradigm for modeling complex reactive systems. The main focus of the paradigm is to enable users to naturally model their perception of the system's requirements (Gordon et al., 2012). In BP, developers define b-threads that describe what must, may, or must not happen in the system. These b-threads are then interwoven at runtime to generate a cohesive system behavior. The paradigm is particularly well-suited for implementing reactive systems for several key reasons: (i) its natural alignment with how humans think about system behaviors enables developers to clearly specify safety constraints; (ii) its modular b-thread architecture allows incremental addition of new behaviors without modifying existing code (Harel et al., 2015); and (iii) its amenability to formal verification enables automated checking of safety and liveness properties (Harel et al., 2011; Harel et al., 2014).

Two main variants of BP have emerged in the literature: classical, context-free BP, which focuses on

[a] https://orcid.org/0000-0003-4510-5335
[b] https://orcid.org/0000-0002-4021-7752
[c] https://orcid.org/0000-0002-4009-5353
[d] https://orcid.org/0000-0001-5292-801X

Ashrov, A., Sturm, A., Elyasaf, A. and Katz, G.
A Study on the Comprehensibility of Behavioral Programming Variants.
DOI: 10.5220/0013440800003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 252-267
ISBN: 978-989-758-742-9; ISSN: 2184-4895

modeling desirable and undesirable behaviors; and *Context-Oriented Behavioral Programming (COBP)*, where developers can also define the context in which behaviors are relevant (Elyasaf, 2021b). Both variants aim to support the natural specification of system behaviors, but their distinct approaches to organizing and expressing these behaviors raise an important question: how do these paradigms affect the comprehensibility of reactive systems, particularly when they are enhanced to meet evolving requirements?

RQ: Is a reactive system more comprehensible when enhanced using classic behavioral programming or context-oriented behavioral programming?

To address this research question, we designed and conducted a controlled experiment involving 109 engineering students. The experiment aimed to empirically evaluate and compare the comprehensibility of reactive systems enhanced using BP versus COBP. We assessed participants on two criteria: (i) their ability to understand the system's behavior and recognize gaps between the implementation and the stated requirements; and (ii) their ability to identify requirements from a given implementation. Thus, our study contributes to understanding how the choice between BP and COBP may impact system comprehensibility.

The experiment results show that classical BP generally outperformed COBP in comprehensibility metrics such as correctness and confidence. In particular, the BP group's subjects were generally more successful in understanding the system's behavior and detecting misalignment between implementation and requirements. In addition, the BP group's subjects were more often successful in correctly identifying specifications from a given code, and were more confident in their answers. However, COBP demonstrated an advantage for certain complex, context-dependent guard rules. Overall, these findings highlight the trade-offs between BP's simplicity and COBP's flexibility for context-driven tasks. We provide an extensive analysis of the experiment and its conclusions in the following sections.

The rest of the paper is organized as follows. In Sect. 2 we provide the background and discuss related work. In Sect. 3 we provide an overview of the reactive system we used in the experiment (called *Taxi*). In Sect. 4 we describe the design of the controlled experiment. In Sect. 5, we detail the experiment's results, and then discuss them in Sect. 6. In Sect. 7 we survey the threats to validity, and then conclude and discuss future work in Sect. 8.

2 BACKGROUND

2.1 Behavioral Programming

Behavioral Programming (Harel et al., 2012b) is a paradigm for modeling complex reactive systems. The approach aims to enable users to naturally model their perception of the system's requirements (Gordon et al., 2012). BP is well-studied and has been applied in various domains (Gritzner and Greenyer, 2018; Harel et al., 2016; Harel and Katz, 2014; Greenyer et al., 2016). At the paradigm's core lies the notion of a *b-thread*: a description of a single behavior, desirably a single requirement, which specifies either a desirable or undesirable behavior of the modeled system. Each b-thread is created independently and has no direct contact with other b-threads. Instead, it interacts with a global execution mechanism (Harel et al., 2010), which can execute a set of b-threads cohesively.

A b-thread can be abstractly described as a transition system, in which the states are referred to as *synchronization points*. Upon reaching a synchronization point, the b-thread suspends its execution and declares three types of events: *requested events* it wishes to trigger, *blocked events* that are forbidden from its perspective, and *waited-for events* that are not explicitly requested, but about which the b-thread should be notified if they are triggered. The execution infrastructure then waits for all b-threads to synchronize and selects an event for triggering that is requested and not blocked. Subsequently, the mechanism notifies the b-threads that requested or waited for the triggered event. These notified b-threads then resume their execution until they reach the next synchronization point, at which point the process repeats.

Fig. 1 (adapted from (Elyasaf, 2021b)) illustrates a behavioral model of a simple system designed to control the lights and air-conditioner at a smart home. In the smart home example, we have the following *physical* requirements: **R1:** there is a house with a single room; **R2:** the room has a sensor, a smart light, and a smart air conditioner; and **R3:** the room can either be occupied or empty. In addition, we have the following *behavioral* requirements: **R4:** when the room is occupied, the light and then the air conditioner should be turned on; **R5:** when the room is empty, the light and then the air conditioner should be turned off; and **R6:** when in EMERGENCY mode, the lights should be kept on.

Requirements **R1**, **R2**, and **R3** are supported by sensors and actuators connected to the environment. The sensor threads inject the external events into the system: NONEMPTYROOM, EMPTYROOM,

EMERGENCY, and ENDEMERGENCY. The SMARTROOMON b-thread implements requirement **R4**. It waits for the NONEMPTYROOM event, and requests the event LIGHTSON followed by the AIRCONDITIONERON event. Symmetrically, the SMARTROOMOFF b-thread implements **R5** and waits for the EMPTYROOM event, and then requests the LIGHTSOFF event followed by the AIRCONDITIONEROFF event. Finally, the EMERGENCYLIGHTS b-thread addresses requirement **R6**: it waits for the EMERGENCY event and blocks the LIGHTSOFF event until the emergency ends. These requirements are implemented using the BPjs flavor of BP (Bar-Sinai et al., 2018).

We notice a problematic pattern by examining the implementation in Fig. 1. In the b-thread SMARTROOMON, we start by waiting for the NONEMPTYROOM event that signals that the room is not empty. Afterward, in every synchronization point, we wait for the event EMPTYROOM. We wish to avoid performing actions related to a non-empty room, if the state has changed. The b-thread is not aligned with **R4** because it also specifies how we know that the room is empty again, which is not defined in the requirement. Therefore, if the requirement that specifies how we know that the room is empty changes, the first two b-threads will change as well. Since in BP, each b-thread should be aligned with a single requirement, this pattern is problematic. A possible solution is introducing the notion of *Context* to BP, as described next.

2.2 Context-Oriented Behavioral Programming

Context-Oriented Behavioral Programming (COBP) (Elyasaf, 2021b) is an extension of Behavioral Programming that facilitates the development of context-aware systems, focusing on the natural and incremental specification of context-dependent behaviors. This is achieved by integrating BP with context idioms that explicitly define when b-threads are relevant and what information they require. The core idea involves incorporating a behavioral model with a data model that defines the context. There is a link between the two models in the form of *update* and *select* operations, and this connection enables developers to address context-dependent requirements. COBP has been applied to model various reactive systems such as IoT (Elyasaf et al., 2018), games (Elyasaf et al., 2019), and cellular automata (Elyasaf, 2021b). For additional details on COBP, please refer to (Elyasaf, 2021b).

We now specify the smart home example using COBPjs (Elyasaf, 2021a), a JavaScript package based on BPjs that implements COBP. We start by defining the data model, which includes one room with one member: *occupied*. Additionally, there is a member indicating whether the emergency mode is active. We then define three queries related to the room's state and emergency mode, to determine which of the bound b-threads is relevant. The data model specification addresses the physical requirements presented in Sect. 2.1; it appears in Fig. 2.

Next, we specify b-threads to implement the behavioral requirements (see Fig. 3). The b-threads are now bound to the appropriate query (i.e., context). Whenever there is a new answer to a query, a new instance of the b-thread is spawned, and the answer is given as a local variable to the b-thread. We say that this local variable is the b-thread context. Similarly, when a local variable is no longer a valid answer to the query, we say that the context has ended, and the b-thread execution is terminated. When the program starts, there is one answer to the EMPTYROOM query; therefore, only the SMARTROOMOFF b-thread is executed. If a person enters the room, the EMPTYROOM context is updated, and the SMARTROOMOFF b-thread will be terminated.

Compared to the BP code presented earlier, the COBP code aligns better with the requirements. There is a separation between the questions of "How do we know that we are in context A?" and "What do we do in context A?". For example, consider **R5**: "When the room is empty, the light and then the air conditioner should be turned off." This requirement does not specify how we know that we are in the context of EMPTYROOM; it only specifies what to do in this context. This difference becomes more apparent when there are several manners to enter/exit a context, and when many behaviors are bound to the context.

2.3 BP vs. COBP

Feature Comparison. While BP and COBP afford the same expressive power, COBP's integrated data model (i.e., the context) enables data-sharing between b-threads, and consequently the encoding of context-dependent behavior. In BP, this ability is not supported natively, but can be achieved using various workarounds (Katz et al., 2015; Harel et al., 2013), e.g., by embedding the entire context in each event. Such solutions add significant amounts of code to the model and often weaken the alignment between b-threads and requirements. In contrast, COBP's context-specific idioms and extended semantics facilitate the introduction of b-threads that are aligned with context-dependent requirements. In addition, in

```
bp.registerBThread( "SmartRoomOn", function() {
    while(true) {
        bp.sync({ waitFor: bp.Event("NonEmptyRoom") });
        lastEvent = bp.sync({ request: bp.Event("LightsOn"),
                              waitFor: bp.Event("EmptyRoom") });
        if (lastEvent.name !== "EmptyRoom")
            bp.sync({ request: bp.Event("AirConditionerOn"),
                      waitFor: bp.Event("EmptyRoom") });
}});

bp.registerBThread( "SmartRoomOff", function() {
    while(true) {
        bp.sync({ waitFor: bp.Event("EmptyRoom") });
        lastEvent = bp.sync({ request: bp.Event("LightsOff"),
                              waitFor: bp.Event("NonEmptyRoom") });
        if (lastEvent.name !== "NonEmptyRoom")
            bp.sync({ request: bp.Event("AirConditionerOff"),
                      waitFor: bp.Event("NonEmptyRoom") });
});

bp.registerBThread( "EmergencyLights", function() {
    while(true) {
        bp.sync({ waitFor: bp.Event("Emergency") });
        bp.sync({ waitFor: bp.Event("EndEmergency"),
                  block: bp.Event("LightsOff") });
}});
```

Figure 1: A BPjs model for controlling the lights and air conditioner at a smart home. Note that we wait for the event that signals that the room state has changed in the relevant synchronization points.

terms of incrementality, both paradigms can model desired/undesired behaviors that are not bound to a specific context. The incrementality in COBP is improved since it can address new context-dependent requirements with new b-threads or updates to context and without the modification of existing b-threads. Table 1 contains a comparison of BP and COBP features.

Cognitive Dimensions Comparison. We focus here on the *"cognitive dimensions"* framework for characterizing programming languages (Green and Petre, 1996; Green, 1989), as it aligns well with our goal of comparing the structural and notational aspects of BP variants. There exist additional relevant frameworks, such as *"cognitive load theory" (CLT)* (Sweller, 1988), which evaluates the mental effort and its impact on learning and problem-solving efficiency during comprehension tasks; we leave these for future work.

We compare the BP variants along the three following dimensions: (i) *Hidden/explicit dependencies*: the degree to which relationships between different parts of the model are visible and clear to the developer. These dependencies can occur in BP/COBP, where a b-thread requesting a series of events might affect a different b-thread waiting for these events which is not immediately visible. In addition, in COBP, a triggered event might update the context, affecting context-dependent b-threads that the developer may not readily see; (ii) *Role-expressiveness*: the ease of understanding each component's role in the system. In BP, when there is a perfect alignment between the requirements and the implementation, it should be simple to understand the role of the b-thread because it should address a single requirement. In COBP, this ought to be true for context-dependent requirements as well; and (iii) *Hard mental operations*: cognitive challenges where notation itself makes tasks harder. In BP/COBP, a hard mental operation could be understanding which event should be triggered by inspecting a synchronization point across multiple b-threads that request/wait-for/block sets of different events. In COBP, a hard mental operation could be to maintain a mental image of the current context, i.e., the schema, update functions, and relevant queries in the data model, while trying to comprehend the behavior of a b-thread in that context.

```
ctx.populateContext([
  ctx.Entity('r1', 'Room', { occupied: false }),
  ctx.Entity('emergency', 'Emergency', { state: false })
]);

ctx.registerEffect('PersonLeavesRoom', function (data) {
    let room = ctx.getEntityById(data.id)
    room.occupied = false
});

ctx.registerEffect('PersonEntersRoom', function (data) {
    let room = ctx.getEntityById(data.id)
    room.occupied = true
});

ctx.registerQuery("NonEmptyRoom", entity =>
  entity.type === 'Room' && entity.occupied === true);

ctx.registerQuery("EmptyRoom", entity =>
  entity.type === 'Room' && entity.occupied === false);

ctx.registerQuery("Emergency", entity =>
  entity.type === 'Emergency' && entity.state === true);
```

Figure 2: The data model of the smart home example in COBPjs. An effect function is activated when the event defined in its name is selected for triggering. The effects that modify the emergency property are similar to those in this example.

Table 1: Comparison of BP and COBP features.

Feature	BP	COBP
Incrementality	Yes	Improved
Architecture	Behavioral Model	Behavioral model and Data Model
Data sharing between b-threads	Partial	Yes
Context-specific idioms	No	Yes
Define context-dependent behavior	Indirect	Direct
Alignment with context-aware requirements	No	Yes

2.4 Related Work

Prior work has explored the integration of context-awareness into Behavioral Programming (BP) to model complex, context-aware reactive systems (Elyasaf et al., 2018; Elyasaf et al., 2019). COBP, introduced in (Elyasaf, 2021b), extends BP by enabling developers to define context-dependent requirements explicitly, offering improved separation of concerns and reduced coupling. The COBP paper compared BP and COBP by highlighting their distinct characteristics and BP's limitations in modeling context-based requirements. In addition, prior work by (Ashrov et al., 2017) validated the comprehensibility of systems enhanced using BP versus BP's structured idiom set through a controlled experiment. Building on these efforts, the present work is the first to empirically compare the comprehensibility of a reactive system modeled using BP vs. COBP.

BP has proven effective in modeling reactive systems across various domains, including cache coherence protocols, robotics, web applications, and IoT systems (Harel et al., 2016; Corsi et al., 2024; Ashrov et al., 2015; Harel and Katz, 2014; Harel et al., 2012a). Override rules, or runtime monitors, have been implemented in diverse domains such as robotics, drones, and autonomous systems (Phan et al., 2017; Desai et al., 2018; Schierman et al., 2015). Recent work has also demonstrated the feasibility of designing safety constraints for deep learning systems using BP (Katz, 2021; Ashrov and Katz, 2023). While these studies focus on functionality, our study uniquely examines the impact of BP and COBP on developers' ability to comprehend systems

```
ctx.bthread( "SmartRoomOn", "NonEmptyRoom", function(room) {
    sync({ request: Event('LightsOn', {id: room.id}) });
    sync({ request: Event('AirConditionerOn', {id: room.id}) });
});

ctx.bthread( "SmartRoomOff", "EmptyRoom", function(room) {
    sync({ request: Event('LightsOff', {id: room.id}) });
    sync({ request: Event('AirConditionerOff', {id: room.id}) });
});

ctx.bthread( "EmergencyLights", "Emergency", function(entity) {
    let room = ctx.getEntityById('r1');
    sync({ block: Event('LightsOff', {id: room.id}) });
});
```

Figure 3: The behavioral layer of the extended smart home example in COBPjs. Note that for every new answer to the defined query, a new instance of the b-thread is created, with the answer provided as a parameter to the b-thread function.

equipped with such rules.

3 TAXI — A REACTIVE SYSTEM

For our controlled experiment, we used the *Taxi* environment (Dietterich, 2000), which involves a taxi navigating to passengers in a grid world, picking them up, and dropping them off at one of four possible locations. This event-based, reactive system is popular in the *Reinforcement Learning* (RL) (Sutton and Barto, 1999) community as a toy environment, but is also sufficiently complex and could be extended as part of our experiment. A full description of the environment can be found at the Gymnasium package website (Taxi, 2024).

We designed various enhancements to the basic Taxi system. These included both *guard rules*, designed to prevent unwanted behavior by the system, and also extensions that introduced new behavior that the original system did not support. For example, we added a rule to avoid in-place circles (inspired by (Corsi et al., 2024)), and a rule for avoiding a turn in a direction where there is a wall. We also introduced a rule that prevented pickup/dropoff at a square that is not a destination square (a *regular* square), which is part of the Taxi's specifications. We named this guard rule GUARDAGAINSTILLEGALACTIONS, and it appears in Fig. 4. Next, we extended Taxi with the ability to refuel, avoid barriers placed in its path, and pick up forgotten packages. These enhancements were included in the questionnaire subjects answered in the experiment (Ashrov et al., 2024a; Ashrov et al., 2024b).

4 THE CONTROLLED EXPERIMENT

In this section, we describe our BP vs. COBP experiment's design and execution, following the guidelines from (Wohlin et al., 2012).

4.1 Hypotheses

Our experiment assessed comprehensibility through two primary dimensions: *comprehension* and *identification*.

- Comprehension was evaluated across two aspects:
  - Execution semantics: The order of handled events in the system.
  - Alignment: The correspondence between requirements and their implementation.
- Identification focused on participants' ability to determine the requirement addressed by a given implementation.

Each dimension was measured using three dependent variables: correctness, confidence, and response time. The following hypotheses guided the study, framing the comparison between BP and COBP:

Overall Achievement Hypotheses (H^1)

- H_0^1: There is no difference in overall achievement correctness between BP and COBP
- H_1^1: BP has greater overall achievement correctness than COBP.

We hypothesized that BP would show better overall performance due to its simpler model and more straightforward learning curve.

```
// Global events used by both b-threads
var directionEvents = [ Event('Up'), Event('Down'),
                        Event('Right'), Event('Left') ]
```

```
// Guard rule - BP
bthread( "GuardAgainstIllegalActions", function() {
  while (true) {
    stateEvent = sync({ waitFor: Event('State') })
    if (taxiIsOnRegularsquare(stateEvent) === true)
      sync({ waitFor: directionEvents,
             block: [ Event('Pickup'), Event('Dropoff') ] })});
```

```
// Guard rule - COBP
ctx.bthread( "GuardAgainstIllegalActions",
  "TaxiIsOnRegularsquare", function(taxi) {
       sync({ block: [ Event('Pickup'), Event('Dropoff') ] })});
```

Figure 4: The guard rule GUARDAGAINSTILLEGALACTIONS implemented in BP and COBP. The specification is: *'There are four designated pick-up and drop-off locations. Avoid picking up or dropping off passengers at non-designated locations'* (Taxi, 2024). These guard rules are relevant when the taxi is at a regular square.

Comprehension Task Hypotheses ($H^{2.1}$-$H^{2.2}$)

Execution Semantics ($H^{2.1}$)

- $H_0^{2.1}$: There is no difference in execution semantics comprehension correctness between BP and COBP.
- $H_1^{2.1}$: BP has greater execution semantics comprehension correctness than COBP.

We hypothesized that understanding system behavior would be easier in BP compared to COBP. While COBP introduced a clear separation between context and behavior, potentially leading to more aligned implementations, this added complexity can increase the cognitive load on developers, thereby hindering their ability to fully comprehend the system's behavior.

Requirements Alignment ($H^{2.2}$)

- $H_0^{2.2}$: There is no difference in requirements alignment comprehension correctness between BP and COBP.
- $H_1^{2.2}$: There is a difference in requirements alignment comprehension correctness between BP and COBP.

We hypothesized a trade-off between BP and COBP: BP's compact syntax and direct definitions potentially simplify translating requirements into specifications. Conversely, COBP's context idioms may allow system specifications to align more closely with the original requirements but at the cost of increased complexity.

Specification Identification Hypotheses (H^3)

- H_0^3: There is no difference in specification identification correctness between BP and COBP.
- H_1^3: There is a difference in specification identification correctness between BP and COBP.

We hypothesized a trade-off: BP excels at capturing high-level requirements but lacks explicit context modeling, potentially resulting in mixed context-behavior specifications that are harder to interpret. On the other hand, COBP allows explicit modeling of context-dependent behaviors, enabling a more detailed specification but requiring a deeper understanding of the interaction between behavior and context.

For each hypothesis (H^1, $H^{2.1}$, $H^{2.2}$, H^3), we also evaluated participant confidence and task completion time using the following format:

Confidence Hypotheses:

- H_{C0}^x: There is no difference in participant confidence between BP and COBP.
- H_{C1}^x: There is a difference in participant confidence between BP and COBP.

Time Hypotheses:

- H_{T0}^x: There is no difference in task completion time between BP and COBP.
- H_{T1}^x There is a difference in task completion time between BP and COBP.

Where x refers to the hypothesis number $(1, 2.1, 2.2, 3)$.

To evaluate these hypotheses, we designed tasks that directly correspond to the dimensions of comprehension and identification. Each task was structured to measure the dependent variables under controlled conditions, allowing us to compare BP and COBP systematically.

4.2 The Experiment's Design

4.2.1 Independent Variable

The independent variable is the *BP variant* used to specify the reactive system, with two alternatives: BP and COBP.

4.2.2 Dependent Variables

The following dependent variables result from assessing correctness, confidence, and response time for the overall performance of the subjects, for the comprehension task, and the identification task.

- *Total correctness*: measures the correctness of all the tasks. The variable is measured on a scale of 0-1, which is the percentage of correct answers.
- *Total Avg. confidence*: measures the Avg. confidence level across the entire questionnaire (values range is [0,5]).
- *Total time*: measures the time it took to solve the two tasks of the questionnaire. Time is calculated in minutes.
- *Comprehension execution correctness*: measures the correctness of the relevant sub-task (scale of [0,1]).
- *Comprehension alignment correctness*: measures the correctness of the relevant sub-task (scale of [0,1]).
- *Comprehension confidence*: measures the Avg. confidence level of the subjects in the comprehension task (values range is [0,5]).
- *Comprehension time*: which measures the time it took the subject to provide an answer for the entire comprehension task. The time is self-measured by the subjects and in minutes.
- *Identification correctness*: measures the solution correctness for the identification task (scale of [0,1]).
- *Identification confidence*: measures the Avg. confidence level of the subjects in the identification task (values range is [0,5]).
- *Identification time*: measures the time it took the subject to provide an answer for the identification task. The time is self-measured by the subjects and in minutes.

4.2.3 Subjects

The participants in the experiment were students enrolled in the course on *Software Quality Engineering* at The Ben-Gurion University of the Negev. The course covers software quality processes and measures, as well as different software testing approaches. The students in the course belong to two distinct groups: (i) the *Information Systems Engineering (ISE)* program students in the third year of their studies; and (ii) the *Software Engineering (SE)* program students in the fourth year of their studies. Both the ISE and SE programs are bachelor's level programs. There is a difference in the curriculum of the two populations: the ISE program focuses on the analysis, design, development, implementation, and management of information systems in organizations and society, whereas the SE program emphasizes engineering methods for software construction and hands-on software development experience.

We selected this population because the subjects have experience in understanding requirements, comprehending code, and identifying gaps between the two. This is a reasonable assumption since they have implemented and maintained several software systems according to requirements during their studies through homework assignments and projects. Moreover, these skills are required for successfully testing software systems, a task the students learn and practice as part of the course.

Participation in the experiment was voluntary. Nonetheless, students were motivated to participate by being offered bonus points toward their course grades based on their performance. Additionally, all participants signed a consent form that explicitly informed them they could withdraw from the experiment at any time. The design and execution of the experiment were approved by the ethics committees at Ben-Gurion University of the Negev and the Hebrew University of Jerusalem.[1]

4.2.4 Training

At the end of the semester, the students were provided with a lecture on BP where they learned about the semantics of the package. Subsequently, they were referred to an online course (Provengo, 2024) about Provengo, a software testing package, which is based on the principles of BP (Bar-Sinai et al., 2023). Afterward, they were assigned a homework task written

[1]Ethics committee approval number: (BGU: SISE-2024-38), (HUJI: CSE-2024-03).

in Provengo where the students were required to test a shopping cart software system.

In addition, the experiment session's initial phase familiarized participants with their assigned BP variant. This was achieved through a lecture covering each variant's core concepts and presenting examples of requirements, the code that implemented them, and the expected behavior. One group learned about COBP, and the other group learned about BP. Note that the lecture on COBP was composed of a section on BP and an additional section on COBP, while the lecture on BP only contained a section on BP. Lastly, the subjects received an online link to the presentation as a handout that could be used throughout the experiment.

4.2.5 Tasks

We designed two versions of the same experimental form, one written in BP (Ashrov et al., 2024a) and the other written in COBP (Ashrov et al., 2024b). Both forms evaluate participants' understanding of the same reactive system (the Taxi environment, see Sect. 3). Each participant experienced only one of the variants. The experiment form consists of four parts:

1. A pre-task questionnaire, in which we checked the comprehension of BP core principles and mechanisms.
2. The first task, in which participants were presented with a requirement and its implementation. They were then asked to answer what is the expected behavior of the implementation and whether it is aligned with the requirements. Table 2 outlines the structure of the questions in the first task, categorizing them by (i) programming concept; (ii) presence of a bug (i.e., alignment/misalignment between the requirement and implementation); (iii) number of code modules in the question; (iv) average code lines per module.
3. The second task, in which participants were presented with an implementation and were asked to identify the requirements fulfilled by the proposed b-threads. Table 3 outlines the structure of the questions in the second task, categorizing them by (i) programming concept; (ii) number of code modules in the question; (iii) average code lines per module.
4. A post-task questionnaire about the participants' perception of the assigned variant and its usefulness.

To ensure objective evaluation, we established ground truth answers before the experiment. For comprehension questions, answers were validated by executing code in BPjs/COBPjs. For alignment and identification questions, we documented the intended requirements-implementation mappings, and multiple researchers reviewed the answer key to ensure accurate and objective grading.

The comprehension and identification tasks in our experiment mirror common development scenarios in the industry. Developers frequently need to understand existing behavioral specifications when maintaining software systems, especially when adding new safety constraints or extending system functionality. Similarly, developers often need to identify the requirements implemented by existing code when documenting legacy systems or when they perform onboarding to new projects.

4.2.6 Execution

Prior to conducting the experiment, we performed a pilot with third-year students from the *Holon Institute of Technology* (HIT) learning about the visual representation of reactive systems. The goal was to confirm that the proposed training was sufficient for solving questions in BP/COBP and to validate the readability and clarity of the questionnaire. The experiment took place at the Ben-Gurion University of the Negev, during a dedicated three-hour session for COBP and a three-hour session for BP. Nevertheless, the COBP execution exceeded the time limit, lasting 3.5 hours, because the COBP lecture contained more information than the BP version and required an intermission.

While random assignment would have provided a more balanced design, we opted for self-enrollment to accommodate scheduling constraints, as the groups met in different time slots. Students were not informed that each group would solve a different version. To mitigate potential selection bias, we verified no significant differences in BP background between groups (See Table 4 and Table 5) and conducted separate analyses for ISE and SE students to maintain statistical power. The results across both programs suggest that the uneven group sizes did not substantially impact our findings. In future studies, we aim to employ stratified randomization to further enhance balance and robustness. In total, 109 students participated in the experiment, with 66 enrolling in the COBP group and 43 enrolling in the BP group.

5 EXPERIMENTAL RESULTS

In this section, we present the results of our controlled experiment. The complete experimental data, including raw results, and statistical analyses, is available in our online repository (Ashrov et al., 2024c).

Table 2: Structure of questions in the first task (comprehension), detailing the question number, programming concept, bug presence, and for each notation (BP/COBP), the number of code modules and average lines of code per module.

Question	Concept	Bug	BP		COBP	
			#Modules	Avg. code	#Modules	Avg. code
Q1	Encapsulation	No	1	6	1	6
Q2	Guard — Block Internal Event	Yes	2	5.5	2	5.5
Q3	Guard — Block Single Event	Yes	3	3.33	5	3.4
Q4	Guard — Block Multiple Events	No	3	4	6	4
Q5	Guard — Override Event	No	2	7.5	8	3.125
Q6	Enhancement	No	2	5	6	3.83
Q7	Enhancement with Guard	Yes	3	5	8	3.875

Table 3: Structure of questions in the second task (identification), detailing the question number, programming concept, and for each notation (BP/COBP), the number of code modules and average lines of code per module.

Question	Concept	BP		COBP	
		#Modules	Avg. code	#Modules	Avg. code
Q1	Enhancement	1	12	7	4.42
Q2	Enhancement — Block Multiple Events	2	5	8	4.125
Q3	Enhancement — Wait and Block Multiple Events	3	3.67	8	4.125
Q4	Guard — Block Single Event	3	4	5	3
Q5	Guard — Block Multiple Events	5	3.4	7	3.85
Q6	Guard — Override Event	1	8	4	4.5
Q7	Guard — Complex Override Event	1	10	4	3.25

5.1 Overview

The ISE and SE students study different curricula and are at different stages in their studies. A *Mann-Whitney* analysis revealed statistically significant differences in their performance (p-values < 0.05) with medium to large effect sizes, justifying separate analyses. Combining the data did not change the significance or effect sizes, confirming that separate analyses highlight meaningful distinctions aligned with their training and expertise.

The ISE group consisted of 61 third-year students (37 in COBP, 24 in BP) focused on information systems analysis and design, while the SE group included 48 fourth-year students (29 in COBP, 19 in BP) specializing in software engineering methods. Both groups had completed at least four semesters of programming courses. A pre-questionnaire assessment of their BP background revealed no statistically significant differences between BP and COBP groups within either population (see Table 4 and Table 5).[2]

Given the non-normal distribution of the data, we employed the Mann-Whitney test to assess statistical significance and *Cohen's r* to calculate effect sizes. These methods were applied consistently across all comparisons to ensure robust and meaningful analysis.

5.2 Total Questionnaire Results

In the following tables, total correctness is the sum of correct answers divided by the total number of questions. Total average confidence refers to the average confidence across all questions, where confidence was measured on a 5-Likert scale. Total time indicates the time taken to complete all tasks, measured in minutes. The rows represent the various metrics. Each cell presents the average and the standard deviation in brackets. The underlined numbers indicate the best results. Bold entries indicate that the metric was statistically significant. The effect size quantifies the

[2]We present the means of the student's results in the various criteria (and not the median of each variable) as these aggregate several measures.

magnitude of the difference between groups, providing insight into its practical significance.

Table 6 and Table 7 show the overall measures of the questionnaire for the ISE and SE students, respectively. The number of participants is indicated next to each group's name.

The BP group's correctness was superior to COBP in both the ISE and SE populations. The differences are statistically significant, with medium effect sizes for both groups, indicating a meaningful advantage for BP. These findings lead to the rejection of the null hypothesis H_0^1 for both ISE and SE groups.

The BP group was also more confident in their answers, with a statistically significant advantage for the ISE group (medium effect size). This supports the rejection of the null hypothesis H_{C0}^1 for ISE students, while for SE students, the difference was not statistically significant, and we retained the null hypothesis.

In terms of time, the COBP group answered the questionnaire faster than the BP group. This difference was statistically significant for SE students, with a medium effect size, leading to the rejection of H_{T0}^1 for SE. However, no significant time difference was observed for the ISE group, resulting in the retention of the corresponding null hypothesis.

These results demonstrate BP's overall superiority in correctness and confidence, particularly among ISE students, while highlighting COBP's potential advantages in time efficiency for SE students.

5.3 Comprehension and Identification Results

Table 8 and Table 9 summarize the results for comprehension and identification tasks across the ISE and SE populations.

Comprehension Task. The BP group demonstrated superior performance in both execution semantics and alignment sub-tasks, with higher correctness scores for both ISE and SE students. These differences were statistically significant for the alignment subtask, with small-to-medium effect sizes, leading to the rejection of $H_0^{2.2}$ for both groups. This confirms that BP supports better comprehension of alignment with requirements compared to COBP.

Identification Task. In the identification tasks, BP also outperformed COBP in correctness for both ISE and SE students. The differences were statistically significant and demonstrated medium effect sizes, supporting the rejection of H_0^3 for both groups. Additionally, BP participants reported higher confidence in their answers, with statistically significant differences and small-to-medium effect sizes. This led to the rejection of H_{C0}^3.

Time to Solution. COBP participants completed identification tasks faster than BP participants across both populations. The differences were statistically significant, with small-to-medium effect sizes for ISE and SE groups, leading to the rejection of H_{T0}^3 for both groups. However, this time advantage for COBP did not translate into higher correctness or confidence.

These results confirm BP's consistent advantage in comprehension and identification correctness and confidence, particularly for the alignment task. While COBP participants completed tasks more quickly, this may reflect a lack of deeper understanding of the paradigm, potentially leading to quicker but less accurate responses. This highlights the need for further investigation into the relationship between task complexity, participant familiarity with COBP, and response time.

5.4 Per Question Results

In the online Appendix (Ashrov et al., 2024c), we examine specific question results and observe a noteworthy pattern. For an enhancement question (task1-Q6) and guard questions involving complex event overriding (task1-Q5 and task2-Q7), the COBP group shows an advantage in understandability (though not statistically significant). The commonality in the questions where the COBP system proved more understandable than BP is that BP's b-threads consist of two primary components: (1) identifying when the context is active, and (2) defining the actions to perform within that context. In contrast, COBP's b-threads are inherently context-specific and focus solely on action specification. These findings suggest that, in certain cases, COBP may be better suited to defining complex override rules that are more aligned with requirements and easier to understand than BP.

6 DISCUSSION

Our results indicate that BP demonstrated an advantage over COBP in understanding a reactive system enhanced and guarded by the BP variant. Specifically, BP outperformed COBP in both comprehension and identification tasks. Participants who worked with BP provided more correct answers and were more confident in their responses than those with COBP. These findings support our hypothesis that there is a difference between the two variants in terms of system comprehensibility when extended and guarded using each respective approach (rejection of hypotheses H_0^1, $H_0^{2.2}$, and H_0^3).

Table 4: Pre-questionnaire results - ISE*.

Question	BP (24)	COBP (37)	Sig. (M-W)	Effect (r)
Total correctness	0.85 (0.22)	0.81 (0.20)	0.222	0.156
Total Avg. confidence	4.09 (0.53)	3.96 (0.59)	0.467	0.093
Total time	13.88 (6.02)	10.84 (3.82)	0.088	0.218

Table 5: Pre-questionnaire results - SE*.

Question	BP (19)	COBP (29)	Sig. (M-W)	Effect (r)
Total correctness	0.94 (0.07)	0.88 (0.15)	0.292	0.022
Total Avg. confidence	4.41 (0.51)	4.28 (0.58)	0.390	0.018
Total time	15.74 (13.36)	9.07 (3.86)	0.164	0.029

*Correctness is measured on a scale of 0-1. Confidence is on a five-point Likert scale, and time is measured in minutes.

Table 6: The total questionnaire results - ISE*.

DV	BP (24)	COBP (37)	Sig. (M-W)	Effect (r)
Total correctness	0.75 (0.14)	0.61 (0.13)	**0.000**	0.458
Total Avg. confidence	3.65 (0.54)	3.09 (0.61)	**0.000**	0.460
Total time	45.31 (10.46)	41.05 (9.01)	0.116	0.201

Table 7: The total questionnaire results - SE*.

DV	BP (19)	COBP (29)	Sig. (M-W)	Effect (r)
Total correctness	0.83 (0.10)	0.74 (0.12)	**0.007**	0.386
Total Avg. confidence	3.82 (0.69)	3.66 (0.35)	0.104	0.235
Total time	46.91 (7.99)	39.84 (9.42)	**0.014**	0.353

*Correctness is measured on a scale of 0-1. Confidence is on a five-point Likert scale, and time is measured in minutes.

We seek to examine the comprehension task in greater depth. In this task, we measure correctness in two sub-tasks: execution and alignment. BP participants answered more questions correctly in the execution task and alignment sub-tasks. Nevertheless, the COBP group showed an advantage in complex context-dependent guard b-threads. This could be attributed to Green's 'hard mental operations' (Green and Petre, 1996). When the enhancement/guard is simple, the direct approach of BP may require less mental effort from the developer compared to the overhead of understanding the context and bound behavior of COBP. However, when the enhancement/-guard is complex, the BP implementation becomes complicated because it is composed of identifying the context and specifying the action. This, in turn, presents a 'hard mental operation' for the developer who needs to comprehend a b-thread that is not aligned. On the other hand, COBP's clear separation of context and behavior allows developers to focus their mental effort on understanding each component individually, an approach that pays off when dealing with complex b-threads.

In the identification task, BP significantly outperformed COBP in correctness and confidence. BP's advantage can be attributed to two main factors: First, BP's visibility is greater, as described by Green's cognitive dimension (Green and Petre, 1996). The number of modules/b-threads in BP is small, whereas COBP had a larger number of code modules and context-related b-threads that required scrolling to view. This made the BP implementation more accessible to the developer (see Table 2 and Table 3). In addition, the larger number of code modules in COBP likely increased the developer's cognitive load, which could have reduced their ability to identify the implemented requirements. Nonetheless, in specific cases that involve a complex override b-thread, it is possible that the cognitive load presented by BP b-threads could have been greater than COBP because the BP b-threads were not perfectly aligned, leading to an advantage for COBP.

Another key factor in BP's advantages could be its easier learnability compared to COBP, which likely contributed to its better performance on the experimental tasks. COBP, as a more recent approach, introduces a new context-specific idiom interface that may present a steeper learning curve, causing participants to struggle more with their tasks. While COBP participants completed tasks faster, particularly in identification, this may reflect a lack of engagement rather than genuine efficiency, as they may have "given up"

Table 8: Dependent variable results of ISE per task in the questionnaire*.

Concept	DV	BP (24)	COBP (37)	Sig. (M-H)	Effect (r)
Comprehension	Execution correctness	0.74 (0.20)	0.65 (0.17)	0.068	0.233
	Alignment correctness	0.71 (0.20)	0.58 (0.19)	**0.012**	0.322
	Confidence	3.57 (0.61)	2.94 (0.62)	**0.000**	0.469
	Time	28.13 (7.89)	27.26 (7.21)	0.976	0.004
Identification	Correctness	0.81 (0.16)	0.60 (0.19)	**0.000**	0.493
	Confidence	3.74 (0.54)	3.25 (0.72)	**0.006**	0.349
	Time	17.18 (4.91)	13.83 (4.52)	**0.009**	0.334

Table 9: Dependent variable results of SE per task in the questionnaire*.

Concept	DV	BP (19)	COBP (29)	Sig. (M-H)	Effect (r)
Comprehension	Execution correctness	0.83 (0.17)	0.81 (0.15)	0.653	0.065
	Alignment correctness	0.77 (0.15)	0.68 (0.16)	**0.040**	0.297
	Confidence	3.65 (0.70)	3.58 (0.43)	0.471	0.104
	Time	29.60 (6.32)	26.10 (8.44)	0.071	0.260
Identification	Correctness	0.90 (0.10)	0.74 (0.24)	**0.025**	0.323
	Confidence	4.00 (0.74)	3.74 (0.39)	**0.018**	0.341
	Time	17.33 (3.65)	13.72 (3.70)	**0.001**	0.458

*Correctness is measured on a scale of 0-1. Confidence is on a five-point Likert scale, and time is measured in minutes.

on fully comprehending the paradigm. The rejection of H_{T0}^{3} underscores this point, suggesting that COBP's faster responses were not accompanied by higher correctness or confidence. This observation highlights how COBP's recent idioms and learning curve challenges contributed to participants' difficulties in effectively applying its approach.

While both the SE and ISE groups rejected $H_0^{2.2}$ in favor of BP, the gap between BP and COBP differed. SE students showed narrower gaps in alignment performance and execution correctness compared to ISE students. This may stem from differences in educational focus, with SE students emphasizing the technical aspects of software engineering, and ISE students focusing on broader information systems. Additionally, SE students' fourth-year status and greater experience may have enhanced their understanding of BP and COBP principles, contributing to their stronger performance.

The findings of this study have practical implications for developing reactive systems. COBP's strength in managing context-dependent behaviors makes it suited for systems requiring dynamic adjustments, like smart devices and robotic controllers. Conversely, BP's simplicity and modularity are ideal for applications where maintainability and comprehension are paramount, such as safety-critical systems. This study thus provides actionable guidance for practitioners in selecting between BP and COBP, which can be applied in several practical scenarios: (i) When developing safety-critical systems with straightforward guard rules but high comprehensibility requirements, BP's simpler model may help reduce potential maintenance errors and make code reviews more efficient. (ii) For reactive systems with complex contextual requirements, COBP's explicit context modeling can help manage this complexity more effectively. (iii) In systems that start simple but are expected to grow in complexity over time, teams might begin with BP for its easier learning curve, then gradually transition to COBP as context-dependent behaviors become more prevalent.

7 THREATS TO VALIDITY

In this section, we discuss the potential threats to the validity of our study and how we addressed them. We consider four main categories of validity threats: construct validity, internal validity, conclusion validity, and external validity (Wohlin et al., 2012).

Construct Validity. Measuring correctness, confidence, and time is a common practice for measuring comprehension in software engineering (Ashrov et al., 2017; Rajlich and Cowan, 1997). Two main threats to construct validity were identified: (i) potentially measuring understanding of BP/COBP syntax rather than comprehension of enhanced systems, and (ii) whether our experimental instruments validly measure the intended constructs. We addressed these threats by designing domain-specific questions, conducting a pilot study with students having no prior

BP/COBP exposure, and consulting with COBP experts during questionnaire design. The pilot study was particularly valuable in validating that our instruments measured system comprehension and helping refine questions to better target understanding of the enhanced system rather than just paradigm mechanics.

Internal Validity. The students had some familiarity with BP because they had attended a lecture on BP in their course and solved a home assignment with Provengo that was based on BP. We mitigated this risk by explaining BP to both groups at the beginning of the experiment to ensure a shared baseline understanding. In addition, self-selection into groups based on session timing could introduce a bias, as students choosing the same time slot might share characteristics (e.g., time constraints or social connections). We mitigated the knowledge level and potential selection bias by verifying that there were no significant differences in BP background between groups (See tables 4 and 5). Regarding the commitment of the subjects, the compensation of bonus points based on the students' performance increased their motivation and commitment. While offering bonus points is a common practice to motivate participation, it could lead to varying levels of motivation among students. We mitigated this threat by: (i) keeping the bonus points to a modest percentage of the total course grade (maximum 5%), (ii) making it clear to students that the bonus was supplementary and not essential for passing the course, and (iii) structuring the bonus to be proportional to effort and engagement. Finally, the training session for COBP was longer, which might have caused a fatigue effect that also impacted the performance of the COBP group.

Another internal validity threat concerns the self-reported nature of confidence scores. Individual personality differences and gender-based variations in self-assessment tendencies could have influenced these scores. While this study anonymized responses to reduce potential biases, future research should consider collecting demographic information and pairing self-reports with objective performance metrics to better understand and control these effects.

Finally, While we analyzed ISE and SE students separately due to their distinct backgrounds and performance differences, an alternative approach would have been to treat program enrollment as an independent variable (covariate). Given our experimental setup and the significant differences between populations, separate analyses provided clearer insights into how each group engaged with the BP variants. Future research could explore using the program as a covariate with a larger, more balanced sample size.

Conclusion Validity. While Likert scales are ordinal, we used mean values for composite confidence scores across multiple questions, a practice validated for group comparisons (Carifio and Perla, 2007; Norman, 2010). Moreover, We followed the assumptions of the statistical tests (normal distribution and data independence) and the effect size when we analyzed the results. Finally, the answers to the questions were defined before the experiment and were reviewed by experts.

External Validity. One external validity issue is subject selection. The subjects were undergraduate students from two different populations with diverse experience in software engineering and modeling. Kitchnham et al. (Kitchenham et al., 2002) argue that this is acceptable if the research questions are not focused on experts. This is the case in our experiment since we were looking for subjects with little or no experience in the paradigms. Another external validity threat concerns the generalizability of our findings, as the study relied on a single case, the Taxi environment, to evaluate comprehension. This domain was chosen for its balance between simplicity and complexity, making it suitable for an educational context. However, using a single, relatively straightforward domain limits the applicability of our results to broader, more complex real-world scenarios. An additional external validity threat is that our study focused specifically on enhancements and guard clauses, while other common maintenance types such as bug fixes and refactoring were not investigated. We acknowledge these threats and state that generalizing the results should be done carefully, and further studies may be required.

8 SUMMARY

In this paper, we conducted a controlled experiment with 109 participants to compare the comprehension of a reactive system enhanced and guarded using BP versus COBP. The results of this experiment show a general advantage for BP in understanding system behavior, identifying gaps between system behavior and requirements, and comprehending requirements from existing specifications. BP's simplicity makes it preferable for systems requiring clear guard/override rules, or for educational tools where ease of comprehension is paramount. In contrast, COBP showed a slight advantage in complex b-threads involving simultaneous context definition and intricate

override actions, such as smart devices adapting to changing conditions or robotic controllers navigating through dynamic environments. These findings provide actionable guidance for selecting the appropriate paradigm based on system complexity and the nature of guard rules. Specifically, our results suggest:

- Use BP when the system requires high maintainability and clear code comprehension.
- Consider COBP for systems with complex contextual requirements, especially when behaviors need to adapt dynamically to changing conditions.
- Factor in team expertise and system evolution — BP's simpler learning curve may benefit rapid development, while COBP's context-awareness better supports long-term scalability.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments. This work was partially funded by the European Union (ERC, VeriDeL, 101112713). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. Additionally, this work was partially supported by the Israeli Smart Transportation Research Center (ISTRC).

REFERENCES

Ashrov, A., Gordon, M., Marron, A., Sturm, A., and Weiss, G. (2017). Structured Behavioral Programming Idioms. In *Proc. 18th Int. Conf. on Enterprise, Business-Process and Information Systems Modeling: (BPMDS)*, pages 319–333.

Ashrov, A. and Katz, G. (2023). Enhancing Deep Learning with Scenario-Based Override Rules: a Case Study. In *Proc. 11th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 253–268.

Ashrov, A., Marron, A., Weiss, G., and Wiener, G. (2015). A Use-Case for Behavioral Programming: An Architecture in JavaScript and Blockly for Interactive Applications with Cross-Cutting Scenarios. *Science of Computer Programming*, 98:268–292.

Ashrov, A., Sturm, A., Elyasaf, A., and Katz, G. (2024a). A Study on the Comprehensibility of Behavioral Programming Variants — BP Questionnaire. https://forms.gle/nLtGkPGdNM4USJtG8.

Ashrov, A., Sturm, A., Elyasaf, A., and Katz, G. (2024b). A Study on the Comprehensibility of Behavioral Programming Variants — COBP Questionnaire. https://forms.gle/4fQ7N3bTvDd6XuNQ6.

Ashrov, A., Sturm, A., Elyasaf, A., and Katz, G. (2024c). Supplementary Materials: A Study on the Comprehensibility of Behavioral Programming Variants. https://doi.org/10.5281/zenodo.14845899.

Bar-Sinai, M., Elyasaf, A., Weiss, G., and Weiss, Y. (2023). Provengo: A Tool Suite for Scenario Driven Model-Based Testing. In *Proc. 38th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, pages 2062–2065.

Bar-Sinai, M., Weiss, G., and Shmuel, R. (2018). BPjs—a Framework for Modeling Reactive Systems using a Scripting Language and BP. Technical Report. http://arxiv.org/abs/1806.00842.

Brooks, R. (1983). Towards a Theory of the Comprehension of Computer Programs. *Int. Journal of Man-Machine Studies*, 18(6):543–554.

Carifio, J. and Perla, R. J. (2007). Ten Common Misunderstandings, Misconceptions, Persistent Myths and Urban Legends about Likert Scales and Likert Response Formats and their Antidotes. *Journal of Social Sciences*, 3(3):106–116.

Corsi, D., Yerushalmi, R., Amir, G., Farinelli, A., Harel, D., and Katz, G. (2024). Constrained Reinforcement Learning for Robotics via Scenario-Based Programming. In *Proc. 31st Int. Conf. on Neural Information Processing (ICONIP)*.

Curasma, H. P. and Estrella, J. C. (2023). Reactive Software Architectures in IoT: A Literature Review. In *Proc. Int. Conf. on Research in Adaptive and Convergent Systems (RACS)*, pages 1–8.

Dafflon, B., Vilca, J., Gechter, F., and Adouane, L. (2015). Adaptive Autonomous Navigation using Reactive Multi-agent System for Control Law Merging. In *Proc. Int. Conf On Computational Science (ICCS)*, pages 423–432.

Desai, A., Ghosh, S., Seshia, S. A., Shankar, N., and Tiwari, A. (2018). Soter: Programming Safe Robotics System using Runtime Assurance. Technical Report. http://arxiv.org/abs/1808.07921.

Dietterich, G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Elyasaf, A. (2021a). COBPjs: Context-oriented behavioral programming in JavaScript. https://github.com/bThink-BGU/COBPjs.

Elyasaf, A. (2021b). Context-Oriented Behavioral Programming. *Information and Software Technology*, 133.

Elyasaf, A., Marron, A., Sturm, A., and Weiss, G. (2018). A Context-Based Behavioral Language for IoT. In *Proc. 21st ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MODELS) Workshops*, pages 485–494.

Elyasaf, A., Sadon, A., Weiss, G., and Yaacov, T. (2019). Using Behavioural Programming with Solver, Context, and Deep Reinforcement Learning for Playing a Simplified Robocup-Type Game. In *Proc. 22nd*

ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C), pages 243–251.

Gordon, M., Marron, A., and Meerbaum-Salant, O. (2012). Spaghetti for the Main Course? Observations on the Naturalness of Scenario-Based Programming. In *Proc. 17th Conf. on Innovation and Technology in Computer Science Education (ITICSE)*, pages 198–203.

Green, T. (1989). Cognitive Dimensions of Notations. *People and Computers V*, pages 443–460.

Green, T. and Petre, M. (1996). Usability Analysis of Visual Programming Environments: a 'Cognitive Dimensions' Framework. *Journal of Visual Languages & Computing*, 7(2):131–174.

Greenyer, J., Gritzner, D., Katz, G., and Marron, A. (2016). Scenario-Based Modeling and Synthesis for Reactive Systems with Dynamic System Structure in ScenarioTools. In *Proc. 19th ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems (MODELS)*, pages 16–23.

Gritzner, D. and Greenyer, J. (2018). Synthesizing Executable PLC Code for Robots from Scenario-Based GR (1) Specifications. In *Proc. Workshops on Software Technologies: Applications and Foundations (STAF)*, pages 247–262.

Harel, D., Kantor, A., and Katz, G. (2013). Relaxing Synchronization Constraints in Behavioral Programs. In *Proc. 19th Int. Conf. on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, pages 355–372.

Harel, D., Kantor, A., Katz, G., Marron, A., Weiss, G., and Wiener, G. (2015). Towards Behavioral Programming in Distributed Architectures. *Journal of Science of Computer Programming (J. SCP)*, 98:233–267.

Harel, D. and Katz, G. (2014). Scaling-Up Behavioral Programming: Steps from Basic Principles to Application Architectures. In *Proc. 4th SPLASH Workshop on Programming based on Actors, Agents and Decentralized Control (AGERE!)*, pages 95–108.

Harel, D., Katz, G., Marelly, R., and Marron, A. (2016). An Initial Wise Development Environment for Behavioral Models. In *Proc. 4th Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 600–612.

Harel, D., Katz, G., Marron, A., and Weiss, G. (2012a). Non-Intrusive Repair of Reactive Programs. In *Proc. 17th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS)*, pages 3–12.

Harel, D., Katz, G., Marron, A., and Weiss, G. (2014). Non-Intrusive Repair of Safety and Liveness Violations in Reactive Programs. *Transactions on Computational Collective Intelligence (TCCI)*, 16:1–33.

Harel, D., Lampert, R., Marron, A., and Weiss, G. (2011). Model-Checking Behavioral Programs. In *Proc. 9th ACM Int. Conf. on Embedded Software*, pages 279–288.

Harel, D., Marron, A., and Weiss, G. (2010). Programming Coordinated Behavior in Java. In *Proc. European Conf. on Object-Oriented Programming (ECOOP)*, pages 250–274.

Harel, D., Marron, A., and Weiss, G. (2012b). Behavioral Programming. *Communications of the ACM*, 55(7):90–100.

Kaelbling, L. (1987). An Architecture for Intelligent Reactive Systems. *Reasoning about Actions and Plans*, pages 395–410.

Katz, G. (2021). Augmenting Deep Neural Networks with Scenario-Based Guard Rules. *Communications in Computer and Information Science (CCIS)*, 1361:147–172.

Katz, G., Barrett, C., and Harel, D. (2015). Theory-Aided Model Checking of Concurrent Transition Systems. In *Proc. 15th Int. Conf. on Formal Methods in Computer-Aided Design (FMCAD)*, pages 81–88.

Kitchenham, B., Pfleeger, S., Pickard, L., Jones, P., Hoaglin, D., El Emam, K., and Rosenberg, J. (2002). Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734.

Morandini, M., Marchetto, A., and Perini, A. (2011). Requirements Comprehension: a Controlled Experiment on Conceptual Modeling Methods. In *Proc. Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 53–60.

Norman, G. (2010). Likert Scales, Levels of Measurement and the "laws" of Statistics. *Advances in Health Sciences Education*, 15:625–632.

Phan, D., Yang, J., Grosu, R., Smolka, S., and Stoller, S. (2017). Collision Avoidance for Mobile Robots with Limited Sensing and Limited Information about Moving Obstacles. *Formal Methods in System Design (FMSD)*, 51(1):62–86.

Provengo (2024). Provengo Online Course. https://provengo.github.io/Course.

Rajlich, V. and Cowan, G. (1997). Towards Standard for Experiments in Program Comprehension. In *Proc. 5th Int. Workshop on Program Comprehension (IWPC)*, pages 160–161.

Schierman, J., DeVore, M., Richards, N., Gandhi, N., Cooper, J., Horneman, K., Stoller, S., and Smolka, S. (2015). Runtime Assurance Framework Development for Highly Adaptive Flight Control Systems. Technical Report. https://apps.dtic.mil/docs/citations/AD1010277.

Storey, M.-A. (2005). Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In *Proc. 13th Int. Workshop on Program Comprehension (IWPC)*, pages 181–191.

Sutton, R. and Barto, A. (1999). Reinforcement Learning. *Journal of Cognitive Neuroscience*, 11(1):126–134.

Sweller, J. (1988). Cognitive Load During Problem Solving: Effects on Learning. *Cognitive Science*, 12(2):257–285.

Taxi (2024). The Taxi Environment. https://gymnasium.farama.org/environments/toy_text/taxi/.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*, volume 236. Springer.