How Users Interpret Bugs in Trigger-Action Programming

Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang[†], Michael L. Littman[†], Blase Ur
University of Chicago, † Brown University
{wbrackenbury,akdeora,jjritchey,jvallee,hewj,blase}@uchicago.edu, {wang,mlittman}@cs.brown.edu

ABSTRACT

Trigger-action programming (TAP) is a programming model enabling users to connect services and devices by writing if-then rules. As such systems are deployed in increasingly complex scenarios, users must be able to identify programming bugs and reason about how to fix them. We first systematize the temporal paradigms through which TAP systems could express rules. We then identify ten classes of TAP programming bugs related to control flow, timing, and inaccurate user expectations. We report on a 153-participant online study where participants were assigned to a temporal paradigm and shown a series of pre-written TAP rules. Half of the rules exhibited bugs from our ten bug classes. For most of the bug classes, we found that the presence of a bug made it harder for participants to correctly predict the behavior of the rule. Our findings suggest directions for better supporting end-user programmers.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; • **Software and its engineering** → *Error handling and recovery*;

KEYWORDS

Trigger-action programming, end-user programming, IoT, Internet of Things, IFTTT, bugs, debugging

ACM Reference Format:
Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, Blase Ur. 2019. How Users Interpret Bugs, in Trigger-Action Programming. In *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019), May 4–9, 2019, Glasgow, Scotland Uk.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3290605.3300782

1 INTRODUCTION

The power of data-driven services and the Internet of Things (IoT) centers on interconnections between components [53]. While users could plausibly specify how IoT devices and online services interact using a variety of interaction methods [10], both industry and academia have repeatedly turned to end-user programming, especially trigger-action programming (**TAP**). In TAP, a user creates rules of the form "IF a **trigger** occurs, THEN perform an **action**." For example, users can write rules such as, "IF someone tells the voice assistant they are sad THEN turn the lights blue." This paradigm is the basis for the popular website IFTTT ("If This, Then That") [26], Microsoft Flow [33], Zapier [27], Mozilla's Things Gateway [24], Stringify [44], SmartRules [43], NinjaBlocks [54], and many others. The TAP approach can be used to connect physical IoT devices, as in the example above, or as a substitute for shell scripting in workplace environments (e.g., allowing novice programmers to write rules that automatically back up data). Academic researchers have recognized the effectiveness of this paradigm for end-user development, conducting a number of studies that rely on TAP or a close variant [4, 9, 13, 18, 19, 22, 23, 34, 50–52].

Rules in TAP typically connect a single event to a single action (e.g., "IF I am tagged in a Facebook photo, THEN save that photo to Dropbox"). However, many important behaviors require greater expressiveness. For example, one might want lights to turn on when they arrive home, but only at night. Expressing such a behavior requires that rules support conjunctions ("and" clauses) within triggers. Deployed platforms differ in their support for this. Stringify [44] and SmartRules [43] support conjunctions in a single trigger, while Microsoft Flow [33] and Zapier [27] provide only partial support through trigger filters. While some studies found participants could write rules with conjunctions regardless of their prior programming experience [50], others highlighted inconsistencies in how users interpret such rules [22].

To reason about TAP rules unambiguously, it is necessary to fully specify the programming model. We are the first to do so here, delineating three temporal paradigms that could govern how TAP rules are expressed. We identify the syntactic components and limitations of each of these paradigms, providing the first of our three key contributions.

As TAP is deployed in more complex environments, the potential for programming bugs becomes more likely. We identify ten programming bugs that might arise in TAP by re-examining the literature, translating bugs from other domains into TAP, and discussing use cases with scientists deploying a TAP system. To our knowledge, three of these ten bugs are novel within the context of TAP. Of the remaining seven, no single previous paper had identified more than three. We grouped these ten bugs into three classes: bugs in control flow, timing-related bugs, and errors in user interpretation. This taxonomy is our second contribution.

To gauge whether these ten bugs impair user understanding of TAP rules, we conducted a 153-participant online study, our third contribution. We assigned each participant to one of the three temporal paradigms and had them complete a tutorial about TAP. We then showed them a series of TAP rules with accompanying scenarios. In half of these instances, the rules exhibited one of the bugs from our taxonomy. In the other half, the rules were written correctly. We found participants were significantly more likely to accurately predict the outcome of bug-free rules, compared to those exhibiting a bug. We also qualitatively coded participants' explanations of why they chose each answer to better understand the manifestation of these bugs, to unpack users' mental models of TAP systems, and to identify possible interventions.

We begin by discussing related work in Section 2. We then present our systematization of TAP temporal paradigms (Section 3) and our taxonomy of TAP bugs (Section 4). We then describe the methodology (Section 5) and results (Section 6) of our user study. Finally, in Section 7, we discuss directions for helping users avoid these bugs.

2 RELATED WORK

End-user development has long been an area of interest within context-aware systems and IoT environments [10, 13, 14, 25, 30, 35, 42], as well as for web mash-ups [7, 32, 36]. The ubiquity of TAP-like formulations of end-user programming may reflect users' mental models. Dey et al. found that users often specified behaviors for context-aware applications in an if-then style [13], while Ur et al. found that many desired behaviors in smart homes could be expressed with TAP [50]. Pane et al. found similar results for programming solutions expressed by non-programmers [45]. These results may reflect, however, the framing of tasks by researchers, rather than inherent mental models [8]. Nonetheless, multiple studies have found that users can write TAP programs regardless of prior programming experience [19, 50], as demonstrated by thriving real-world TAP ecosystems like IFTTT [40, 51].

There are drawbacks to current TAP interfaces, however. Many do not support desired complex functionality [4], many TAP interfaces lack feedback during rule creation [19], and temporal issues in TAP can lead to inaccurate mental models [22]. Initial attempts to mitigate some of these problems leverage techniques typical of software engineering, including formal verification [34, 57], information-flow control [49], privilege isolation [15], and dynamic instrumentation [52].

3 TEMPORALITY IN TAP

The temporality of triggers and actions is a crucial source of ambiguity for TAP. Prior work by Ur et al. [50] and by Huang and Cakmak [22] noted this difficulty, distinguishing between triggers based on **events** (things that occur at a moment in time) and **states** (conditions that are true over a period of time). Here, we adopt the following terminology:

- An **event trigger** occurs at a particular instant in time (e.g., "it begins to rain").
- A **state trigger** is a condition that remains true or false over time (e.g., "it is currently raining").

We also distinguish between the following action types:

- An **event action** occurs or begins at a particular instant in time (e.g., "turn on the light").
- A **state action** identifies the desired state of a device in a time period (e.g., "the light should be on").

Prior work has identified ambiguities caused by temporality [22]. It has not, however, formalized or united the conflicting interpretations. We do so here, proposing three temporal paradigms within which users can write TAP rules with unambiguous meaning. Rulesets in one temporality may be translated to any other.

Table 1 summarizes the paradigms: Event–Event; State–State; and Event–State. This systematization is important because the literature is inconsistent in handling temporality. Some prior studies considered only event triggers [6, 26], some considered only state triggers [55, 56], and still others distinguished between (and used) both [5, 22, 43, 47, 50]. Alarmingly, other work has used both types of triggers without specifying how such rules would function [11, 13, 16, 19].

The Event–Event Paradigm

As TAP is often event-driven, combinations of events would seem to be a viable candidate for triggers. A crucial observation, however, is that events do not compose well—they are unlikely to occur at the exact same instant in time [22, 50]. Therefore, the Event–Event paradigm uses time windows (e.g., "IF Sally enters the bedroom AND the sun sets WITHIN 2 hours") to account for events not occuring simultaneously. When any event that is part of the trigger occurs, the TAP system would look back in its history to determine whether all other events in the trigger had occurred within the time window. If they had, it would execute the rule's action.

Paradigm	Format	Examples
Event–Event →Event	IF event (AND event | AND AFTERWARDS event)* (WITHIN time window) THEN event	• IF Sally enters the bedroom AND the sun sets WITHIN 2 hours THEN turn on the bedroom lights. • IF Sally enters the bedroom AND AFTERWARDS the sun sets WITHIN 2 hours THEN turn on the bedroom lights.
Event–State →Event	IF event (WHILE state (AND state)*)* THEN event	• IF the sun sets WHILE Alice is currently at home THEN turn on the bedroom lights. • IF the sun sets WHILE Alice is currently at home AND it is currently raining THEN turn on the bedroom lights.
State–State →State	IF state (AND state)* THEN state PRIORITY priority	• IF Bill is currently in the room THEN the temperature should be 70° PRIORITY 8. • IF Charlie is currently in the room AND it is currently night THEN the temperature should be 65° PRIORITY 9.
State–State →Event	IF state (AND state)* THEN event	• IF I am currently hungry AND no pizza has been ordered in the last 3 hours THEN order a pizza.

Table 1: The three TAP temporal paradigms, defined by the trigger temporality. The action temporality follows from the trigger temporality. The third paradigm, State–State, has two variations because the action can be either a state or an event. The asterisk (*) denotes components that may appear an arbitrary number of times, including zero. PRIORITY indicates the ordering of rule precedence, with higher numbers indicating higher precedence.

To enable particular orderings of events, we also support "and afterwards." For example, "IF the sun sets AND AFTERWARDS Sally enters the bedroom WITHIN 2 hours" will only trigger if Sally enters the bedroom after the sun sets.

The Event–State Paradigm

The Event–State paradigm requires that each trigger contain one event, with zero or more states [43, 50]. When an event trigger occurs, the system would check whether the trigger's states are true. The rule triggers if and only if all specified states are true at the moment the event trigger occurs. Unfortunately, prior research has shown that the distinction between events and states is lost on many users [22]. Nonetheless, the Event–State paradigm is used in the real-world deployments of Stringify and SmartRules [27, 44], as well as in many academic studies [5, 22, 43, 47, 50].

The State–State Paradigm

In the State–State paradigm, triggers only contain states (e.g., "IF it is currently raining AND I am currently at home"). Because state triggers are true over an interval of time, state actions are most natural (e.g., "the lights should currently be blue"), though it is also possible to include events as actions. Because multiple states can be true simultaneously, rules affecting the same capability (e.g., controlling the lights) must include a prioritization to resolve conflicts. We expand on this in Section 4 through the Priority Conflict bug.

Because some actions (e.g., sending an email) are discrete events that cannot be expressed as state actions in a straightforward way, in Table 1 we provide State–State formulations for both state actions and event actions. Additionally, supporting event actions within State–State introduces complications. Consider the rule "IF I am hungry, THEN order a pizza." While one is hungry, this rule continually evaluates to true and thus continually triggers, ordering many pizzas.

Other Logical Operators

Neither "or" nor "not" operators are necessary in TAP semantics. Triggers involving "or" can be expressed as separate rules. Note that we used "or" in our user study to separate rules in a ruleset, as discussed in Section 6. The "not" operator is not needed because the negation of a state trigger can be encoded in the state itself (e.g., "it is not raining"), whereas the negation of an event requires a notion of history (e.g., a given event has *not* occurred in a specified time period).

4 BACKGROUND

In this section, we taxonomize ten bugs that might arise in TAP. Four of these bugs (Priority Conflict, Secure-Default Bias, Extended Action, and Missing Reversal) have been discussed at length in TAP literature, while three (Infinite Loop, Repeated Triggering, and Nondeterministic Timing) have been identified but not fully discussed. The remaining three (Contradictory Action, Time-Window Fallacy, and Flipped Triggers) are, to our knowledge, new to the TAP literature.

Our methodology for uncovering these bugs consisted of three activities: re-examining the literature on TAP, attempting to translate bugs observed in novice programming and distributed systems to TAP, and discussing desired use cases and implementation pain points with scientists deploying their own TAP system. Naturally, no bug taxonomy is complete. We anticipate new bugs could be discovered in long-term field studies that uncover complex interactions over time. Logging the use of TAP systems and evolution of rules over time could also provide insight into new bugs. However, we expect our taxonomy captures common bugs.

Bug Name	Paradigm	Description
Control-flow bugs		
Infinite Loop	All	One rule triggers another rule, which then triggers the first, ad infinitum.
Contradictory Action	All	An infinite loop over an extended period (e.g., alternating between heating and cooling).
Repeated Triggering	All	A rule repeatedly triggers because an event occurs many times or a state remains true.
Timing bugs		
Nondeterministic Timing	All	The order in which nearly concurrent events are processed changes the system's behavior.
Extended Action	State–State	Rules fail to account for the extended timing of an action (e.g., brewing coffee).
Inaccurate user expectation		
Missing Reversal	All	Even when no rule exists to undo an action (e.g., turning on lights), users assume one exists.
Secure-Default Bias	All	Users assume an action is performed (e.g., locks are locked) because doing so is more secure.
Time-Window Fallacy	Event–Event	The specified time window is ignored in favor of a more intuitive interpretation.
Priority Conflict	State–State	Users ignore the stated priorities of rules in favor of what would make the rules match intent.
Flipped Triggers	Event–Event, Event-State	In trigger conjunctions, assuming a reversal of the triggers could still trigger the rule.

Table 2: A taxonomy of the ten potential TAP bugs we examined and their temporal paradigms.

Control-Flow Bugs

Some TAP bugs may impair proper control flow [52].

Infinite Loop bugs arise when rules trigger each other, leading to loops. For example, if a user wants to add items to their to-do list when a file is placed into a to-do folder and save a file in their to-do folder when an item is added to their to-do list, a naive implementation could lead to a never-ending storm of new files as one rule triggers the other. This bug has been encountered within end-user programming of robots [1]. Static or dynamic analysis could detect this bug.

Contradictory Action bugs describe an infinite loop over an extended period of time. For example, a naive set of rules that turn on the heat if the temperature drops below a threshold and turns on air conditioning above the threshold could result in a system that does not converge to an ideal temperature. The TAP literature does not identify this specific form of bug, though Brich et al. studied the related issue of unintended side effects in TAP programming [4]. This bug is difficult to detect in static or dynamic analysis.

Repeated Triggering is when users expect a rule to only trigger once, yet it triggers multiple times. For example, traveling on a road that varies between 0.9 miles and 1.1 miles from a pizza shop can cause "IF I come within 1 mile of a pizza shop THEN order me a pizza" to order many pizzas. This bug is particularly common in the State–State paradigm, where triggers are states that are true over long periods of time. Repeated Triggering was briefly noted in work on mashup programming [7]. Static analysis can detect possible Repeated Triggering, but cannot determine if it is intentional.

Timing Bugs

Timing bugs, common in distributed systems [21, 31], are also possible in TAP. Prior work found novice programmers in non-TAP contexts struggle to reason about concurrency [38].

Nondeterministic Timing bugs arise due to nondeterminism of the order in which a system processes simultaneous triggers. Given the rules "IF the clock strikes 8:00 PM AND it starts raining WITHIN ten minutes THEN turn off the lights" and "IF the clock strikes 8:01 PM AND it starts raining WITHIN ten minutes THEN turn on the lights," rain that starts at 8:05 PM will trigger both simultaneously. Whether the light is now on or off depends on the order in which the system processes the rules. This bug is frequent in the distributed systems literature, and Huang and Cakmak describe it briefly within TAP [22]. Static and dynamic analysis can detect potential Nondeterministic Timing bugs.

Extended Action bugs are due to actions that occur over time, rather than instantaneously, such as brewing coffee [22]. A rule designed to brew coffee when no coffee is ready would continue to trigger when coffee is brewing (but not yet ready), potentially leading to excess coffee. Although Huang and Cakmak described extended actions [22], the particular Extended Action bugs that result are an extension of their work.

Inaccurate User Expectations

Bugs in users' mental models, which can arise from novice programmers ascribing systems intelligence beyond what they possess [2, 3, 12, 46], may also arise in TAP [22].

Missing Reversal bugs occur when a user creates a rule that performs some action, yet neglects to write a rule undoing that action. For example, "IF I walk into the living room THEN turn on the lights" turns on the lights when a person enters, yet performs no action when they leave. Prior work has found that users expect such rules to automatically revert themselves when appropriate [22, 55]. Huang and Cakmak discussed Missing Reversal at length [22], noting users expect systems to have a default regardless of semantics. Static analysis can detect, but not always fix, a Missing Reversal.

Secure-Default Bias bugs occur when users assume a system defaults to a safe state [55], such as windows remaining locked at night. However, in the most widely deployed paradigm, Event–State, devices do not have default states. Yarosh and Zave discuss this bug in the context of locks [55].

Time-Window Fallacy bugs occur when users ignore the time window specified by a ruleset constructed in the Event–Event paradigm in favor of a more intuitive interpretation, particularly for rulesets that are more naturally expressed in other temporal paradigms. This bug was not previously noted, likely because prior work left the composition of multiple events as ambiguous.

Priority Conflict occurs when multiple State–State rules act on the same device, causing the need for rule prioritization. Prior work has found that users struggle to prioritize rules to match their intent [55]. Prioritization is essential when multiple rules act on the same device, and when rules act on distinct devices with similar results (e.g. turning on lights and raising blinds to cause more light). The Priority Conflict bug has been discussed extensively in the TAP literature [4, 13, 16, 34, 41, 56]. Static analysis can detect this.

Flipped Triggers occur when users struggle specifying which part of a trigger should be an event and which should be a state. For example, a user might confuse "IF the garage door opens WHILE it is raining, THEN close the garage door" with the subtly different "IF it starts raining WHILE then garage door is open, THEN close the garage door." However, these rules behave differently. This bug does not appear in previous TAP literature, although the Event-State paradigm has been studied. Formal methods can produce a rule's complement, but cannot determine when it is appropriate.

5 METHODOLOGY

To gauge whether the bugs we identified lead to misinterpretation of TAP rulesets, we conducted an online user study. We recruited participants on Amazon's Mechanical Turk, requiring they be 18+ years of age, located in the USA, and have at least a 95% approval rating with at least 100 tasks completed. The study took approximately one hour, for which we paid $10. We randomly assigned each participant to one of the three temporal paradigms described in Section 3. The participant then completed a tutorial about the semantics of their assigned paradigm, answering comprehension questions.

The bulk of the study was a series of scenarios. Each described an intended goal, provided a set of TAP rules attempting to achieve that goal, and gave a detailed description of what occurs. Based on the ruleset shown, participants answered a multiple-choice question about the scenario outcome (e.g., whether the lights would be on or off). Whether participants chose the answer accurately predicting the outcome is their **correctness**. Participants also rated their **confidence** and provided a free-text explanation of their choice.

For each bug in our taxonomy, we created two scenarios in different domains, termed Scenario One and Scenario Two. For inspiration, we examined the TAP literature and discussed use cases with scientists who have implemented TAP-like rules for the Globus system [20]. For each scenario, we created rulesets in each temporal paradigm to which that bug applied. In addition, for each scenario in each paradigm, we created both a **buggy** and a **fixed** ruleset. In the buggy version, the rules shown exhibited the bug, while in the fixed version, they did not. All other aspects remained constant.

For each bug applicable to their assigned paradigm, the participant saw one scenario with its fixed ruleset, and the other with its buggy ruleset. Of the ten bugs, seven applied to Event-State, while eight each applied to State-State and Event-Event. Thus, Event-State participants saw 14 scenarios, whereas all others saw 16 scenarios. For each bug, which scenario was fixed and which was buggy was randomized, as was the overall order of all 14–16 scenarios.

The study concluded with questions about demographics, background in computer programming, and familiarity with both TAP and IoT devices. As we hypothesized that facility with logical thinking might impact performance, we also asked the three-question Cognitive Reflection Test [17].

Example Scenario Task

We provide the full study instrument in our online appendix. Here, we provide an example of one scenario in one paradigm. We presented scenarios in a visual format similar to the IFTTT interface as IFTTT is widely used [40, 51].

Scenario One for Flipped Triggers had the following stated goal: "You do not want your dog Fido playing outside in the rain, since he will track mud back indoors." In the Event-State paradigm, the buggy rule was the following:

*IF **[It starts raining]** WHILE **[Fido is outside]***
*THEN **[Call Fido inside]***

Events were always shown in green IFTTT-like boxes and states in purple boxes. We avoided using loaded colors (e.g., red, orange) for items, but otherwise the choice is arbitrary.

We described the scenario as follows: "At 2:00 PM, it begins to rain, and continues raining. At 2:30 PM, Fido goes outside. At 2:31 PM, will Fido have been called inside?" Participants could choose from the following:

- Fido will have been called inside.
- Fido may or may not have been called inside; the answer depends on other factors.
- Fido will not have been called inside.

Participants then rated their confidence and answered "Why?"

In this example, the correct outcome is "Fido will not have been called inside," contrary to the stated scenario goal. This rule exhibits the Flipped Triggers bug. To trigger the rule in this scenario, the event portion of the trigger must be swapped with the state portion, as it is in the fixed version.

For bugs like Repeated Triggering, we needed to measure the participant's expectation of the *number* of times an action occurs, rather than just whether it occurs. Therefore, we presented participants with four choices (e.g., exactly one pizza will have been ordered, the number of pizzas ordered cannot be determined, no pizzas will have been ordered, and more than one pizza will have been ordered).

Analysis

Our analysis had three goals. First, we wanted to understand if the temporality, the choice of scenario, and whether fixed or buggy rules were seen influenced correctness. Second, we wished to understand participants' TAP mental models and recognition of the bugs. Third, we wanted to test the impact of demographics and prior experiences.

Toward the first goal, we built a mixed-effects logistic regression model for each bug. The dependent variable was whether each participant answered each scenario correctly (1) or incorrectly (0). The independent variables (IVs) were which *temporal paradigm* the participant was assigned, which *scenario* was being tested, and the *ruleset* (buggy or fixed). Each of these IVs was categorical with respective baseline categories "Event-State," "Scenario One," and "fixed." We included terms capturing the interaction between the ruleset and the two other IVs. We used a mixed-effects model with the participant ID as a random effect because each participant answered two scenarios per bug. If a bug applied to only one paradigm, we excluded the paradigm IV.

For the Secure-Default Bias bug, our correctness comparison differed slightly. Scenario Two was hypothesized to trigger the Secure-Default Bias because it involved criminal alerts and locks, while Scenario One was hypothesized not to do so because it involved weather alerts and the color of lights in a garden. For this bug, the scenario IV and its interactions were thus especially important.

To pursue our second goal, we performed qualitative coding on the free-text "Why?" responses. A first coder performed open coding and created a codebook, assigning one class of code for whether the response demonstrated recognition of the bug, and another class of code for any extra assumptions encoded in the justification. The second coder independently used that codebook to code all data, and the coders met to resolve discrepancies. Cohen's κ was 0.795 and 0.919 for the two classes of codes, respectively.

Toward our third goal, we built two additional linear regression models. The dependent variables were the participant's overall correctness (# correct answers / # scenarios seen) and mean confidence self-rating across questions. The IVs encompassed the participant's demographics, CRT score, correctness on tutorial comprehension questions, and prior experience with TAP, computer programming, and the IoT.

Limitations

Our study has several limitations. First, we used a convenience sample that may not capture actual users of TAP interfaces. Second, the framing of our system introduces confounds. We chose an abstraction centered on devices because it has been widely deployed to date, priming participants to think in a certain way [8]. We explained in the tutorial, "you will be asked to decide whether or not certain actions take place in these scenarios based on the rules presented" and to "assume no other rules are present," but prior experiences with IoT devices (e.g., Amazon Echo) using machine learning might bias participants to assume all IoT systems are "smart."

Some rulesets we tested are more complex than those seen in widely deployed TAP instantiations like IFTTT [40, 51]. As TAP is deployed for a larger number of systems in the future, however, ruleset complexity promises to increase. Finally, interpreting pre-written programs and creating programs are very different. The former can unpack mental-model errors, but program creation is the true goal. Our work should be augmented by studies in which participants create TAP rules.

6 RESULTS

We first introduce our participants and summarize which factors impacted their correctness. We then describe aspects of participants' mental models that manifested across scenarios. Subsequently, we detail results for each of the ten bugs. Finally, we describe demographics and experiences correlated with participants' correctness and confidence.

Participants

We had 153 participants ranging in age from 21–69. Among participants, 83 identified as female, 68 as male, and 2 preferred not to answer. Education levels ranged from high school degrees to post-graduate degrees, the largest categories of which were "4-year college degree" (57) and "some college with no degree" (34). Of the 153 participants, 23 majored in, held a degree in, or had held a job in CS-related fields, and 42 had studied computer programming. In total, 63 participants had 15 different types of IoT devices, including voice assistants (47), thermostats (12), and lightbulbs (8).

Overall Performance

Our logistic regression models for each bug allowed us to evaluate whether the temporal paradim and the presence or absence of the bug itself (buggy vs. fixed ruleset) impacted the correctness of participants' multiple-choice answers about the outcome of each scenario. It also allowed us to check whether correctness varied across the two scenarios for each bug. Table 3 summarizes the significant factors in these regressions, while Appendix A in the online supplementary materials presents the full models.

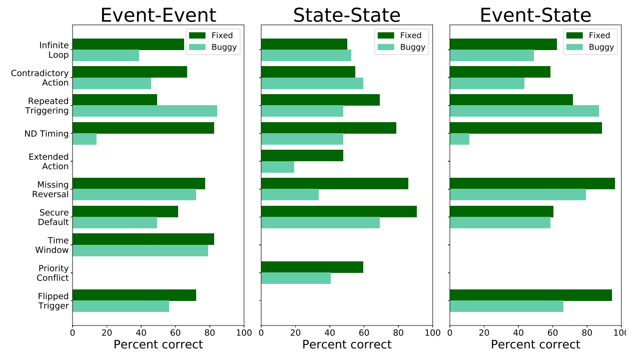

Figure 1: Correctness of responses by bug and by paradigm for fixed (dark green) and buggy (light green) rulesets.

For seven of the ten bugs, participants who saw the buggy ruleset were significantly less likely to choose the correct outcome, compared to those who saw the fixed ruleset (see Figure 1). For two bugs, Repeated Triggering and Time-Window Fallacy, we did not observe a significant effect. For the final bug, Secure-Default Bias, whether the ruleset was buggy was not significant. However, the scenario term in the regression actually encapsulated this bug, as detailed later in the paper, and this term was significant. Thus, the Secure-Default Bias also led to significantly less correct answers. In total, we found evidence that eight of the ten bugs we examined impaired participants' ability to predict scenario outcomes.

The temporal paradigm also significantly impacted correctness. Compared to Event–State, the baseline in our regression, participants in the Event–Event paradigm were significantly less correct for three bugs, while participants in the State–State paradigm were significantly more correct for one bug. Despite our best efforts to make equally difficult scenarios for all bugs other than Secure-Default Bias, responses to Scenario Two were significantly less correct than for Scenario One for two bugs, and significantly more correct for one bug. We also observed a handful of significant interactions between terms, detailed later in this section.

Overall Mental Models and TAP Assumptions

We qualitatively coded participants' free-text explanations of why they chose particular outcomes. As we detail individual bugs, we discuss bug-specific assumptions participants made. However, we also observed some high-level trends in participants' mental models of TAP.

Some participants consistently struggled to distinguish between events and states, even in paradigms like Event–Event or State–State with only one or the other. This confusion also underlies the Flipped Triggers bug. Participants often tended to think in either one manner or the other, underscoring the importance of defining unambiguous semantics for TAP (Section 3) and clearly communicating them to users.

	Paradigm		Ruleset Seen			Scenario	
Bug	**SS**	**EE**	**Buggy**	**B:SS**	**B:EE**	**Two**	**B:Two**
Infinite Loop			↓				
Contradictory Action			↓			↓	↑
Repeated Triggering		↓		↓			
Nondeterministic Timing			↓	↑			
Extended Action (SS)	NA	NA	↓	NA	NA		
Missing Reversal		↓	↓			↓	
Secure-Default Bias	↑					↓	
Time-Window Fallacy (EE)	NA	NA		NA	NA		
Priority Conflict (SS)	NA	NA	↓	NA	NA		↑
Flipped Triggers (ES, EE)	NA	↓	↓	NA		↑	↓

Table 3: Factors that led to significantly more (↑) or significantly less (↓) correct answers per bug. The Event-State paradigm, Fixed rulesets, and Scenario One were the baseline categories. *SS* is State-State and *EE* is Event-Event. *B: refers to the interaction between buggy rulesets and the stated factor. Some factors do not apply (*NA*) if a bug only affects certain paradigms.**

Participants also expressed uncertainty about the interaction between a TAP system and outside actions, such as someone turning off lights by flipping a switch. Assumptions about the outside world were a key impediment to correct predictions of scenario outcomes. While this confusion may have been heightened by the artificial nature of an online user study, interactions between TAP and physical controls are a likely source of confusion in real systems, too.

Finally, particular semantics we tested also caused confusion. Event–Event includes "afterwards" to force an ordering of triggers, yet this caused confusion. While negation can be fully encoded in carefully designed triggers without including a "not" operator (Section 3), we tested some scenarios with a "not" operator, and participants struggled.

Control-Flow Bugs

In this and the following sections, we give detailed results for each bug. The control-flow bugs challenged participants.

Infinite Loop was tested in scenarios predicting whether email attachments would synchronize with a computer folder or text messages. Buggy rulesets automatically forwarded items from one to the other, creating an infinite loop. Fixed rulesets checked the sender or item to avoid duplicates.

Participants correctly predicted outcomes significantly more often with fixed rules — 59.9% of the time — than with buggy rules — 46.1% of the time (OR=0.361, p=0.032). The paradigm and scenario did not have a significant impact.

Of participants who correctly predicted the outcome of buggy rulesets, 81.4% of free-text responses displayed recognition of the bug. For example, "This is a rule with an event as an action that does not specify a time period in the trigger. Therefore, the rule is going to keep triggering. It is going to be going in a loop." The remaining responses were too vague to ascertain whether or not the bug was recognized.

Surprisingly, 17.1% of participants who were incorrect nonetheless mentioned the bug in their justification. Of this 17.1%, 57.1% mentioned other assumptions that led to an incorrect prediction. In particular, they assumed the user may have read the email before the trigger was evaluated (2 participants), that the system automatically would recognize when files were duplicates (2), that the rule would not trigger on emails sent by another rule (1), or that rules were mutually exclusive (1). That is, some participants ascribed to the system internal intelligence and consistency [46].

Contradictory Action bugs were tested in scenarios controlling temperature in a house and generators in a power plant. Buggy rulesets caused one rule to eventually trigger a contradictory rule (e.g., heating the house eventually triggered a rule that cools the house by opening windows).

While participants correctly predicted the outcome 60.5% of the time with fixed rules, they did so only 48.7% of the time with buggy rules (OR=0.190, p=0.001). The scenario (OR=0.182, p<0.001) and the interaction of the scenario and whether the rules were buggy (OR=6.160, p<0.001) also impacted correctness (negatively and positively, respectively).

Our coding found 55.4% of justifications for correct responses mentioned the bug. One noted, "After 10 minutes with the window open the interior would be the same 60 degrees as outside so it would trigger the thermostat, after another 10 minutes [...] it would [...] trigger the other rule." An additional 24.3% of justifications for correct answers did not mention the bug, while 20.3% were ambiguous.

None of the justifications for incorrect answers mentioned the bug. While 86.3% of these responses did not indicate a clear reason for giving the wrong answer, the remaining 13.7% demonstrated misunderstandings. Three participants thought that rules controlling generators could not trigger while the generators were turning on. Two believed the system would not continuously poll a rule after its triggers had initially been checked and found to be false.

Repeated Triggering was tested with scenarios predicting how many pizzas would be ordered or how many alerts from a motion sensor would be sent. Buggy rulesets had no rate-limiting clauses, making rules trigger repeatedly. Fixed rulesets augmented triggers so rules triggered only once. Repeated Triggering occurs either when a state is continuously true or an event occurs multiple times. We thus used similar, yet somewhat different, scenarios in each paradigm.

The impact of the Repeated Triggering bug appears to depend on the paradigm. In our regression model, we did not find the choice of a fixed or buggy ruleset to significantly impact correctness, yet we observed a significant negative interaction between buggy rules and the State–State paradigm (OR=0.119, p=0.005). As shown in Figure 1, participants were more likely to answer correctly for fixed rules than for buggy rules in the State–State paradigm, yet the opposite was true for both the Event–Event and Event–State paradigms. Compared to Event–State, answers overall in Event–Event were less likely to be correct (OR=0.299, p=0.014).

The free-text justifications partially explain these mixed quantitative results. In Event–Event, 15.9% of participants who answered incorrectly struggled with the "not" operator we included in this study, believing the end of the time window was when the truth of the trigger was evaluated. We thus advise against supporting "not" operators. An additional 9.1% appeared to misread the description, while another 6.8% expressed confusion about the trigger.

Timing Bugs

Both Timing Bugs proved challenging for participants.

Nondeterministic Timing involved evaluating scenarios with simultaneous phone calls or texts, as well as individuals arriving at work the instant the clock struck 9:00. Buggy rulesets left ambiguous the order in which rules triggered. Compared to 83.6% correctness for fixed rules, correctness for buggy rules was only 22.4% (OR=0.159, p<0.001). Furthermore, buggy rules and State–State had a significant positive interaction (OR=16.112, p<0.001). Among justifications from participants who answered correctly for buggy rules, 22.6% explicitly mentioned this nondeterminism, lower than for other bugs. Of participants who did not mention the bug, 16.6% assumed an ordering that made intuitive sense to them, 12.5% did not believe there was physically enough time for the notifications to trigger, and 8.3% assumed the full time window had to pass before rules triggered.

Of participants who were incorrect, 29.7% nonetheless recognized that there might be buggy behavior. The majority, 51.4%, assumed an incorrect ordering that made intuitive sense to them, while 37.1% incorrectly assumed the "afterwards" keyword would resolve uncertainty.

Extended Action involved scenarios in which industrial robots ran chemical reactions and coffee brewed. Buggy rulesets would trigger additional rules while these extended actions were in process, yet not complete. Participants correctly predicted outcomes more often (47.6% of the time) with fixed rules than with buggy rules (19.0% of the time) (OR=0.164, p=0.015). Only 5.9% of participants who answered incorrectly for buggy rules mentioned the bug in their response (e.g., "More than one cup could brew"). Participants often ignored the rules' semantics in favor of an intuitive interpretation; 41.1% assumed actions would not buffer, while 20.6% believed they would. Others misread the question (17.6%), assumed extended actions would cease if a state trigger became false (5.9%), or confused events and states (5.9%).

Errors Based on Inaccurate User Expectation

Four of the five bugs we tested related to potentially incorrect user expectations proved problematic for participants.

Missing Reversal was tested in scenarios involving controlling living room lights or house windows. Buggy rulesets turned lights on and closed the windows, but did not include any rules to reverse those actions. Participants were correct 86.2% of the time for fixed rules, but only 63.8% of the time for buggy rules (OR=0.156, p=0.046). The scenario (OR=0.254, p=0.018) and, relative to Event–State, the Event–Event paradigm (OR=0.152, p=0.022) negatively impacted correctness.

Overall, 79.4% of participants who answered correctly for buggy rules mentioned the bug in their justification (e.g., "There is no rule to open the windows after it stops raining"). The remaining 20.6% either re-stated the scenario or made vague statements (e.g., "it's just obvious").

Surprisingly, 40.0% of participants who answered incorrectly nonetheless mentioned the potential for buggy behavior. Six of them anticipated out-of-system manual interactions (e.g., "The rule never turns the lights off, only on. So, they must be turned off manually"), which our tutorial said to ignore. Five believed additional rules might exist ("There may or may not be a rule to re-open the windows"), also disallowed by our tutorial. Two others treated events as states.

Secure-Default Bias was tested differently than other bugs. Instead of comparing buggy and fixed rules, we compared two scenarios with parallel wording. An innocuous Scenario One discussed "weather alerts" and changed the color of garden lights, whereas a security-critical Scenario Two discussed "criminal alerts" and controlled door locks.

Participants correctly predicted scenario outcomes more often in the innocuous scenario — 65.8% — than when shown the security-critical scenario — 61.2% (OR=0.367, p=0.043). Furthermore, compared to Event–State, State–State answers were more likely to be correct (OR=10.075, p=0.002), though our qualitative coding sugggests this difference may derive from ambiguous wording in our survey.

Overall, 72.8% of justifications from participants who answered correctly for buggy rules mentioned the bug. One stated, "The rule to unlock the door from 7-8pm has a higher priority than the criminal alert rule and so the doors will be unlocked then, regardless of the criminal alert." The remaining participants in this category wrote vague justifications.

Among participants who were incorrect, 26.7% explicitly displayed a Secure-Default Bias. In contrast, the majority who did not recognize the bug displayed a few different misunderstandings. For example, 18.2% misunderstood the State–State priority system, while 13.6% incorrectly believed the order in which events occurred would lead one to "override" the other (e.g., "Because the criminal alert was issued before 7pm, it would automatically override the 'unlock' command at 7"). This was unique to the Secure-Default Bias, suggesting that it may also be a manifestation of the bug.

Time-Window Fallacy, exclusive to Event–Event, was tested in scenarios involving alerts based on the arrival of data or a person at the door. In buggy rulesets, the time window extended beyond when the activity was occurring. Thus, the rule would still trigger. The fixed version set a smaller time window that better matched the activity.

Neither the ruleset nor any other factor we tested significantly impacted correctness, which was high overall. Furthermore, 84.4% of participants who answered correctly for buggy rules explicitly mentioned the bug in their justification. Overall, this potential bug appears not to be problematic.

Priority Conflict was tested in scenarios about controlling lights. Buggy rulesets incorrectly applied equal priority to conflicting rules. Fixed rulesets used unique priorities.

Participants correctly predicted scenario outcomes more often — 59.5% — for fixed rules than for buggy rules — 40.5% (OR=0.154, p=0.004). Furthermore, buggy rulesets interacted significantly with the scenario (OR=9.89, p=0.013).

Among participants who answered correctly for buggy rules, 88.2% of their justifications mentioned the bug. One noted it well: "It could go either way because all the rules are met and there is no difference in priority." Of participants who were incorrect, only 24.4% mentioned potential bugs.

Flipped Triggers was tested in scenarios involving calling a dog in from the rain and pairing Bluetooth devices. While the fixed rules triggered in the scenario given, the buggy rules would have triggered only if the event and state (or the two events) in the trigger were swapped. We thus searched for incorrect expectations of the swapped version also triggering the rule despite the rule stating otherwise.

Indeed, 82.7% of participants answered correctly for fixed rules, compared to only 60.9% for buggy rules (OR=0.197, p=0.022). The scenario (OR=5.894, p=0.009), the Event–Event paradigm (OR=0.155, p=0.006), and the interaction between the ruleset and the scenario (OR=0.189, p=0.035) all had significant interactions. Overall, 89.6% of participants who answered correctly for buggy rules mentioned either the bug or the correct semantics of the triggers in their response. For example, "Fido was not outside when it began to rain. It started raining first and then Fido went outside."

Among participants who were incorrect, only 13.9% mentioned the possibility of the bug. While 35.1% seemed to misunderstand the "afterwards" keyword, the majority exhibited the Flipped Triggers bug.

Impact of Demographics and Prior Experience

To test the effects of demographic factors and prior experiences with relevant technologies, we built a linear regression model with the overall percentage of scenarios a participant answered correctly as the dependent variable (Appendix A). We built an analogous model for average confidence ratings.

Participants who had prior programming experience answered more correctly ($\beta = 0.083, p = .038$), but were less confident in their answers ($\beta = -0.363, p = .006$). In contrast to prior work that found programming experience was not a significant factor in TAP correctness [50], this discrepancy may derive from differences in the studies. The prior study focused on simpler scenarios without intentionally buggy behavior. Our study instead focused on challenging situtations. In these situations, users with programming experience may look for edge cases, which we observed in free-text justifications. For example, asked whether a dog would be called indoors, one participant noted, "This depends on the dog. Just because you call him in doesn't mean he will come in."

Participants' logical thinking, measured with the CRT [17], was also correlated with greater correctness ($\beta = 0.038$, $p = .006$). Unsurprisingly, participants were more confident in their answers if they had used IFTTT ($\beta = 0.312, p = .009$) or did better on the tutorial questions ($\beta = 0.106, p = .026$).

7 CONCLUSIONS AND DISCUSSION

Despite recent interest, most literature treats TAP temporality imprecisely, preventing systematic evaluations in complex situations. We thus defined semantics that unambiguously express TAP rules in three temporal paradigms. We also synthesized a taxonomy of ten bugs likely to impact TAP systems. Of the ten, three were completely novel, and no single prior paper had collected more than three. This taxonomy can serve as a roadmap for developing end-user debugging tools. To gauge if these bugs impair users' ability to predict TAP outcomes, we conducted a 153-participant online study. Participants interpreted TAP rules in scenarios designed to elicit these ten bugs. Eight of these bugs impaired participants' ability to correctly predict scenario outcomes.

Identifying Bugs

Significant research in software engineering addresses bug detection, but less work focuses on debugging in end-user programming. The Priority Conflict, Infinite Loop, Extended Action, and Nondeterministic Timing bugs can potentially be detected by static analysis. Prior work has already shown success in combatting Priority Conflict bugs [34, 41], though current static analysis techniques may be insufficient to fully handle Infinite Loop bugs. To address Extended Action bugs, a system could highlight that the action is extended, asking users about expected behaviors. To address Nondeterministic Timing bugs, the system could analyze rules that might conflict, asking for a priority ordering. Our results showed that some participants were unable to detect these bugs by themselves. Many studies have noted, however, that feedback during rule creation can enable end-users to correct simple bugs when notified [19, 28, 29, 37, 48].

The Contradictory Action and Repeated Triggering bugs can only be detected at runtime. This is because current formal methods cannot model a system's self-interference. Thus, Contradictory Action bugs should be met with pattern analysis in system logs. Repeated Triggering bugs require heuristic methods to resolve. Static analysis can suggest when these bugs might be present, but some situations are intentional (e.g. ordering many pizzas for a party).

The remaining bugs cannot be detected automatically, either because the situation is ambiguous (Secure-Default Bias, Time-Window Fallacy), or because suggestions for fixes may be incorrect (Missing Reversal, Flipped Triggers). Whie the Time-Window Fallacy did not impair understanding in our study, the others did. Recent work [57] combining formal methods with user-specified properties (e.g., "the window is open AND it is raining SHOULD NEVER OCCUR TOGETHER") may help users identify and repair such bugs.

Communicating Bugs to Users

Correctness cannot be achieved with bug detection alone. Interfaces that support effective end-user debugging are critical. A lack of feedback during rule creation is a key impediment to creating correct TAP rules [19]. The simplest type of feedback would be an interface highlighting rules that might suffer from a bug. For less challenging bugs, this may be sufficient [28, 48]. Automatically suggesting potential fixes (that may or may not match user intent) during rule creation might help. Seeing a proposed fix might be sufficient to show the user why the fix is necessary or not. For example, a system could show the user a rule's Flipped Triggers reversal, hoping that this will be enough for a user to decide whether they need that variant. The user must be deeply involved in this process, though, because correctness is often subjective, and suggested rules may not match their intent.

The limited work thus far on debugging smart homes has focused on providing retroactive insight into why unintended behaviors triggered. For example, Mennicken et al. proposed a calendar-based interface for this purppose [39], in line with work on interrogative debugging [29] for end-user programming. Plausibly, users could also provide feedback to the system, specifying undesirable behaviors and asking why are occurring. Model checking techniques could also identify when rules would cause these behaviors [34, 57].

However, we propose that interfaces could instead proactively simulate the rule outcomes. For example, a system could ask the user whether the lights should be on or off in a given scenario, then use the answer to choose among synthesized rules or prioritize existing rules. Interfaces could visualize concrete differences between when subtly different rules would have triggered in the past, or might in the future.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. 1837120 and 1835890.

REFERENCES

[1] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. 2015. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In Proc. ICRA.

[2] Jeffrey Bonar and Elliot Soloway. 1983. Uncovering principles of novice programming. In Proc. POPL.

[3] Jeffrey Bonar and Elliot Soloway. 1985. Preprogramming knowledge: A major source of misconceptions in novice programmers. Human-Computer Interaction 1, 2 (1985), 133–161.

[4] Julia Brich, Marcel Walch, Michael Rietzler, Michael Weber, and Florian Schaub. 2017. Exploring end user programming needs in home automation. ACM TOCHI 24, 2 (2017), 11.

[5] Federico Cabitza, Daniela Fogli, Rosa Lanzilotti, and Antonio Piccinno. 2017. Rule-based tools for the configuration of ambient intelligence systems: a comparative user study. Multimedia Tools and Applications 76, 4 (2017), 5221–5241.

[6] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S. Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In Proc. WWW.

[7] Jill Cao, Kyle Rector, Thomas H. Park, Scott D. Fleming, Margaret Burnett, and Susan Wiedenbeck. 2010. A debugging perspective on end-user mashup programming. In Proc. VL/HCC.

[8] Meghan Clark, Mark W. Newman, and Prabal Dutta. 2017. Devices and data and agents, oh my: How smart home abstractions prime end-user mental models. ACM IMWUT 1, 3 (2017), 44.

[9] Yngve Dahl and Reidar-Martin Svendsen. 2011. End-user composition interfaces for smart environments: A preliminary study of usability factors. In Design, User Experience, and Usability. Theory, Methods, Tools and Practice. 118–127.

[10] Scott Davidoff, Min Kyung Lee, Charles Yiu, John Zimmerman, and Anind K. Dey. 2006. Principles of smart home control. In Proc. UbiComp.

[11] Luigi De Russis and Fulvio Corno. 2015. Homerules: A tangible end-user programming interface for smart homes. In Proc. CHI Extended Abstracts.

[12] Daniel C Dennett. 2017. Brainstorms: Philosophical essays on mind and psychology. MIT press.

[13] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive prototyping of context-aware applications. In Proc. Pervasive.

[14] W. Keith Edwards and Rebecca E. Grinter. 2001. At home with ubiquitous computing: Seven challenges. In Proc. UbiComp.

[15] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. 2018. Decentralized action integrity for trigger-action IoT platforms. Proc. NDSS.

[16] Jacopo Fiorenza and Andrea Mariani. 2015. Improving trigger action programming in smart buildings through suggestions based on behavioral graphs analysis. Technical Report. Politecnico di Milano.

[17] Shane Frederick. 2005. Cognitive reflection and decision making. Journal of Economic Perspectives 19, 4 (2005).

[18] Manuel García-Herranz, Pablo Haya, and Xavier Alamán. 2010. Towards a ubiquitous end-user programming system for smart spaces. Journal of Universal Computer Science 16, 12 (2010), 1633–1649.

[19] Giuseppe Ghiani, Marco Manca, Fabio Paternò, and Carmen Santoro. 2017. Personalization of context-dependent applications through trigger-action rules. ACM TOCHI 24, 2 (2017), 14.

[20] Globus. 2019. https://www.globus.org/.

[21] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, et al. 2014. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In Proc. SOCC.

[22] Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In Proc. UbiComp.

[23] Ting-Hao Kenneth Huang, Amos Azaria, and Jeffrey P. Bigham. 2016. Instructablecrowd: Creating if-then rules via conversations with the crowd. In Proc. CHI Extended Abstracts.

[24] Matthew Hughes. 2018. Mozilla's new Things Gateway is an open home for your smart devices. TheNextWeb.

[25] Jan Humble, Andy Crabtree, Terry Hemmings, Karl-Petter Åkesson, Boriana Koleva, Tom Rodden, and Pär Hansson. 2003. "Playing with the bits" User-configuration of ubiquitous domestic environments. In Proc. UbiComp.

[26] If This Then That. Accessed 2018. If This Then That. https://ifttt.com.

[27] Thorin Klosowski. 2016. Automation Showdown: IFTTT vs Zapier vs Microsoft Flow. LifeHacker.

[28] Andrew J Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, et al. 2011. The state of the art in end-user software engineering. ACM CSUR 43, 3 (2011), 21.

[29] Andrew J. Ko and Brad A. Myers. 2004. Designing the Whyline: A debugging interface for asking questions about program behavior. In Proc. CHI.

[30] Tiiu Koskela and Kaisa Väänänen-Vainio-Mattila. 2004. Evolution towards smart home environments: Empirical evaluation of three user interfaces. Personal Ubiquitous Comput. 8, 3–4 (July 2004), 234–240.

[31] Tanakorn Leesatapornwongsa, Jeffrey F. Lukman, Shan Lu, and Haryadi S. Gunawi. 2016. TaxDC: A taxonomy of non-deterministic concurrency bugs in datacenter distributed systems. ACM SIGPLAN Notices 51, 4 (2016), 517–530.

[32] Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & sharing how-to knowledge in the enterprise. In Proc. CHI.

[33] Nat Levy. 2017. Microsoft updates IFTTT competitor Flow and custom app building tool PowerApps. GeekWire.

[34] Chieh-Jan Mike Liang, Lei Bu, Zhao Li, Junbei Zhang, Shi Han, Karlsson F. Börje, Dongmei Zhang, and Feng Zhao. 2016. Systematically debugging IoT control system correctness for building automation. In Proc. BuildSys.

[35] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-user development: An emerging paradigm. Springer.

[36] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A. Lau. 2009. End-user programming of mashups with Vegemite. In Proc. IUI.

[37] Michelle L. Mazurek, J.P. Arsenault, Joanna Bresee, Nitin Gupta, Iulia Ion, Christina Johns, Daniel Lee, Yuan Liang, Jenny Olsen, Brandon Salmon, Richard Shay, Kami Vaniea, Lujo Bauer, Lorrie Faith Cranor, Gregory R. Ganger, and Michael K. Reiter. 2010. Access control for home data sharing: Attitudes, needs and practices. In Proc. CHI.

[38] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in Scratch. In Proc. ITiCSE.

[39] Sarah Mennicken, David Kim, and Elaine May Huang. 2016. Integrating the smart home into the digital calendar. In Proc. CHI.

[40] Xianghang Mi, Feng Qian, Ying Zhang, and XiaoFeng Wang. 2017. An empirical characterization of IFTTT: Ecosystem, usage, and performance. In Proc. IMC.

[41] Alessandro A. Nacci, Bharathan Balaji, Paola Spoletini, Rajesh Gupta, Donatella Sciuto, and Yuvraj Agarwal. 2015. Buildingrules: A trigger-action based system to manage complex commercial buildings. In UbiComp Adjunct Proceedings.

[42] Mark W. Newman, Ame Elliott, and Trevor F. Smith. 2008. Providing an integrated user experience of networked media, devices, and services

through end-user composition. In Proc. Pervasive.

[43] Obycode, LLC. 2018. SmartRules. http://smartrulesapp.com/.

[44] Ed Oswald. 2016. IFTTT competitor Stringify gets a major update. TechHive.

[45] John F. Pane, Brad A. Myers, and Leah Miller. 2002. Using HCI techniques to design a more usable programming system. Technical Report. Carnegie Mellon University.

[46] Roy D. Pea. 1986. Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research 2, 1 (1986), 25–36.

[47] Amir Rahmati, Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. 2017. IFTTT vs. Zapier: A comparative study of trigger-action programming frameworks. arXiv:1709.02788 (2017).

[48] T.J. Robertson, Shrinu Prabhakararao, Margaret Burnett, Curtis Cook, Joseph R. Ruthruff, Laura Beckwith, and Amit Phalgune. 2004. Impact of interruption style on end-user debugging. In Proc. CHI.

[49] Milijana Surbatovich, Jassim Aljuraidan, Lujo Bauer, Anupam Das, and Limin Jia. 2017. Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of IFTTT recipes. In Proc. WWW.

[50] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical trigger-action programming in the smart home. In Proc. CHI.

[51] Blase Ur, Melwyn Pak Yong Ho, Stephen Brawner, Jiyun Lee, Sarah Mennicken, Noah Picard, Diane Schulze, and Michael L. Littman. 2016. Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes. In Proc. CHI.

[52] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. 2018. Fear and logging in the Internet of Things. In Proc. NDSS.

[53] Wireless Watch. 2017. Huawei could rescue Amazon's Alexa from the smart home. The Register. https://www.theregister.co.uk/2017/01/26/huawei_throws_amazon_alexa_a_mobile_lifeline_to_reach_beyond_the_home/.

[54] Jong-bum Woo and Youn-kyung Lim. 2015. User experience in do-it-yourself-style smart homes. In Proc. UbiComp.

[55] Lana Yarosh and Pamela Zave. 2017. Locked or not?: Mental models of IoT feature interaction. In Proc. CHI.

[56] Pamela Zave, Eric Cheung, and Svetlana Yarosh. 2015. Toward user-centric feature composition for the Internet of Things. arXiv:1510.06714 (2015).

[57] Lefan Zhang, Weijia He, Jesse Martinez, Noah Brackenbury, Shan Lu, and Blase Ur. 2019. Synthesizing and repairing trigger-action programs using LTL properties. In Proc. ICSE.