2015 12th Working IEEE/IFIP Conference on Software Architecture

Proactive Detection of Higher-Order Software Design Conflicts

Jae young Bang and Nenad Medvidovic
Computer Science Department,
Viterbi School of Engineering,
University of Southern California,
Los Angeles, California 90089, USA
Email: {jaeyounb, neno}@usc.edu

***Abstract*—Software architects who collaboratively evolve a software model rely on version control systems (VCSs) to synchronize their individual changes to the model. However, with the current generation of software model VCSs, architects remain unaware of newly arising conflicts until the next synchronization, raising the risk that delayed conflict resolution will be much harder. There are existing tools that proactively detect analogous conflicts at the level of source code. However, it is challenging to directly use them for software models because those tools are constructed to manage code-level rather than model-level changes. Furthermore, no empirical data is currently available regarding the impact of proactive conflict detection on collaborative design. In this paper, we report on our design-level proactive conflict detection research, which specifically targets a class of higher-order conflicts that do not prevent merging but do violate a system's consistency rule. We present FLAME, an extensible, operation-based collaborative software design framework that proactively detects conflicts. We also present a user study result involving FLAME conducted with 42 participants. The study indicated that the participants who used FLAME were able to create higher quality models in the same amount of time, and to detect and resolve higher-order conflicts earlier and more quickly.**

I. Introduction

Collaborative software design is challenging. Software architects make design decisions, reify those decisions into software models [1], and evolve the models as a team [2]. To support this collaborative evolution of models, a number of design environments have emerged. There are group editors that provide a shared "whiteboard" [3]–[5] or synchronize the models in realtime [6], but the major research effort has been toward the asynchronous, copy-edit-merge style software model version control systems (VCSs). Those VCSs provide each architect her individual workspace by loosely synchronizing the models in an on-demand fashion to parallelize the architects' work and maximize their productivity [7].

The loose synchronization of the software model VCSs, however, exposes the architects to the risk of causing *conflicts* [6]. Today's software model VCSs [7] detect conflicts only when the architects synchronize their models. As a result, the architects often make changes to the model without fully understanding what issues may arise when they merge with others' changes. It is also possible that new changes made after a conflict has been introduced need to be reversed in the process of resolving the conflict, which results in wasted time and effort [8].

What if those conflicts could be detected earlier in a proactive fashion, that is, before an architect synchronizes her model and finally becomes aware of them? Collaborative software *implementation* faces a similar challenge of exposing software developers to causing conflicts at the level of source code, and the state-of-the-art techniques and tools are indeed capable of proactively providing the code-level conflict information [9]. However, while the previous, code-level proactive conflict detection research shed light on how to deal with the conflicts in general, it is challenging to directly apply that to the conflicts at the level of software models due to the following two reasons. First, the existing proactive conflict detection tools are not designed to manage changes made to graphical software models and are often limited to specific development environments into which they are integrated. Tools that are designed to manage textual changes made to source code are known not to work well with graphical software models [10]–[14], and may not be configurable to deal with various kinds of design conflicts that differ per design environment. Second, to our best knowledge, no empirical study has been reported to date on whether or to what extent proactive design conflict detection may impact the cost of collaborative software design.

We present our research that attempts to alleviate the risk of having software design conflicts by detecting them proactively. In general, design conflicts can be categorized into two different types: *synchronization* and *higher-order* conflicts [14]. A synchronization conflict is a set of contradictory modeling changes by multiple architects made to the same artifact or to closely related artifacts, which means they cannot be merged together. A higher-order conflict is a set of modeling changes by multiple architects that can be merged but together violate the system's consistency rules (e.g., cardinality defined by the metamodel). The two types differ in that they require different sets of detection techniques. While both types pose similar risks, in this paper, we focus on higher-order conflicts since they are often more difficult to detect and resolve [14]–[16].

The focus of our research is *not* regarding how to detect the higher-order design conflicts, which has already been widely studied. We focus on the ways to *proactively* detect those conflicts by exploiting the existing techniques, and further, on the impact of doing so on collaborative design cost.

The contribution of this paper is two-fold:

1) We present the architecture and implementation of a collaborative software design environment, *Framework for*

978-1-4799-1922-2/15 $31.00 © 2015 IEEE
DOI 10.1109/WICSA.2015.15

Logging and Analyzing Modeling Events (FLAME), which proactively detects potential higher-order conflicts by running the conflict detection activities in the background. FLAME is unique as it is the first reported proactive conflict detection framework that is specifically designed for collaborative software *design*. FLAME has two distinct characteristics: (1) it is *extensible* to allow architects to plug-in the most appropriate conflict detection tools for their modeling environment, and (2) it implements *operation-based* model construction [12], [17] to be able to perform conflict detection at the rate of each individual modeling change if necessary.

2) We report the result of an exploratory user study we conducted with 42 participants using FLAME to assess whether or to what extent providing proactive conflict detection impacts the cost of collaborative software design. We found that the participants who were provided with proactive conflict detection had more chances to communicate and were able to create higher quality models in the same amount of time and to detect and resolve higher-order conflicts earlier and more quickly.

The rest of the paper is organized as following: in Section II, we introduce the problem by defining the types of conflicts that could occur during collaborative software design. We introduce our solution to the problem, FLAME, regarding its architecture and implementation in Section III, and present the user study we conducted using it in Section IV. In Section V, we list and discuss the related work, and we conclude the paper in Section VI.

II. Problem

Collaborative software design suffers from several types of conflicts. To clearly illustrate the problem and to drive the discussion throughout this paper, we will use an example system called Next-Generation Climate Architecture (NGCA, depicted in Figure 1). NGCA is created based on the design documents of NASA's Computational Modeling Algorithms and Cyberinfrastructure [18], an infrastructure that supports computationally-heavy data comparisons between climate simulation model output and the actual climate data collected via remote sensors. Because the climate simulation models and databases that compose NGCA belong to different organizations scattered around the world, designing NGCA naturally becomes collaborative involving software architects from those organizations. Consider the following scenario with two software architects Jane and John designing NGCA:

> Jane and John are software architects who are involved in the NGCA design project. They make design decisions and document the decisions into a software model. The modeling environment they use is semantically-rich and domain-specific, and is capable of estimating three critical runtime properties of NGCA: memory usage, message latency, and energy consumption. The model is managed by a current generation, copy-edit-merge style software model VCS. One day, each of them makes changes to their respective local copies of the model, runs estimations locally, and moves to another design task after they do not find any issue in the estimated property values. However, a while later, when they try to synchronize their local copies of the model by merging the changes, they realize one or both of the following two situations:
>
> 1) Their changes were made to the same object and were incompatible; Jane removed the object while John updated an attribute of the same object. As a result, their changes cannot be merged together.
> 2) They are able to merge their changes to other parts of the model, but the model, with the changes from Jane and John merged together, estimates that the memory usage of the system at its runtime will surpass the threshold defined by the NGCA requirement.

The above scenario depicts examples of (1) a synchronization conflict and (2) a higher-order conflict respectively. We define those conflicts [14] as following:

- A **synchronization conflict** is a set of design decisions made by multiple architects that are not compatible and cannot be merged together. It occurs when multiple software architects make contradictory modeling changes on the same software modeling artifact or closely related artifacts. It is also called a context-free conflict [19]. It is a similar concept of a textual conflict [15] or a direct conflict [20]–[23] discussed in literature.
- A **higher-order conflict** is a set of design decisions made by multiple architects that do not prevent synchronization but together violate a consistency rule or a semantic rule of the system. In other words, a higher-order conflict manifests itself as an *inconsistency* in the merged model. It is also called a context-sensitive conflict [19]. An analogous concept at the source code level [15] is known as an indirect conflict [20]–[23].

A software model cannot realistically evolve without having inconsistencies [24], [25]. At the same time, having an inconsistency caused by a higher-order conflict that is undetected and unknown to the architects is a risk. For example, Jane and John, when they found the higher-order conflict, would have to revisit their previous changes in order to understand and resolve the conflict, recalling the rationale and assumptions they made along the way. Moreover, their work performed after the conflict has been introduced may need to be reversed during the process, which leads to wasted effort and increased development cost.

To deal with the risk of having undetected higher-order conflicts, one solution would be for the architects to synchronize and detect the conflicts highly frequently, e.g., for every change that they make. However, the cost of conflict detection in that case is likely to overwhelm its benefits. In today's collaborative software design environments, that is a trade-off decision that the architects have to make. The burden of conflict detection grows further when the detection technique is computationally expensive. While a few performance-oriented model analysis techniques are lightweight [26], [27], many other well-known techniques such as discrete-event simulation [28], Markov-chain-based reliability analysis [29], or queueing-network-based performance analysis [30] are often computation-intensive and time-consuming, rendering highly frequent conflict detection less affordable, especially as the size of the system model grows.

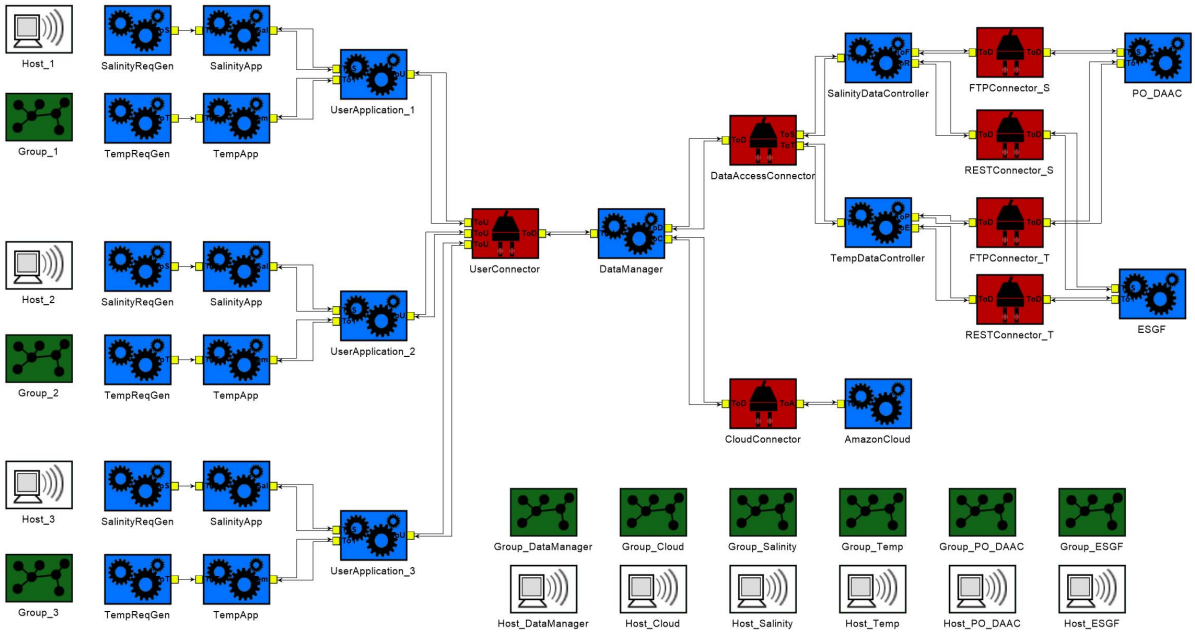


Fig. 1: A high-level model of Next-Generation Climate Architecture.

A number of tools for collaborative software *implementation* that proactively detect conflicts have emerged [9] to minimize the analogous risk of having undetected conflicts at the level of source code. Those tools perform trial merging and conflict detection activities in the background to detect conflicts early. We posit that a similar approach, when it is applied to collaborative software *design*, will reduce the risk that software architects face. However, challenges exist in directly reusing the existing proactive conflict detection tools for software design because these tools are constructed to manage code-level rather than model-level changes and are often integrated into a specific development environment.

An added challenge is that different software modeling environments depend on their unique combinations of modeling tool and consistency checkers, and any proactive conflict detection solution should be able to cope with the differences. For example, revisiting our scenario, the proactive conflict detection tool for NGCA should be able to (1) orchestrate the NGCA-specific modeling tool and consistency checkers to automatically perform the higher-order conflict detection activities in the background, (2) present conflict information specific to the environment (e.g., violations of the three runtime properties of NGCA), as well as (3) synchronize the graphical modeling changes. Unfortunately, the existing proactive conflict detection tools do not fully satisfy the above requirements.

III. Solution

A. Approach

To alleviate the risk of having undetected higher-order design conflicts, we have developed an extensible, operation-based collaborative software design framework, named *Framework for Logging and Analyzing Modeling Events* (FLAME). FLAME minimizes the duration of time during which the conflicts are present but unknown to software architects by proactively performing the conflict detection activity that includes a trial merging of modeling changes and execution of consistency checking tools in the background. FLAME subsequently presents the conflict information to the architects in case the architects' attention is required.

FLAME has two characteristics that distinguish it from the existing proactive conflict detection tools. First, FLAME is *extensible*. Software modeling environments differ in their modeling tools, languages, and the suitable consistency checkers. FLAME utilizes an event-based architecture in which highly-decoupled components exchange messages via implicit invocation, allowing flexible system composition and adaptation. FLAME exploits this event-based architecture to provide explicit extension points for plugging a variety of off-the-shelf tools, namely, modeling tools and conflict detection engines, that are most appropriate for the given modeling environment. Second, FLAME is *operation-based*. It synchronizes the models at the granularity of a single modeling operation such as creation, update, or removal of a modeling element. Conflict detection can become more fine-grained if the synchronization is done for each modeling operation rather than at the level of "diffs" between stored states of the model (e.g., saved files). This fine-grained synchronization is advantageous because an architect can find out which particular operation she performed has caused a conflict [17]. On the other hand, as discussed above, performing conflict detection for each operation could be a significant, even unacceptable tax on the system's performance. FLAME deals with this explicitly, by employing remote, cloud-based analysis nodes, as discussed below.

B. Architecture of FLAME

FLAME adds higher-order proactive conflict detection on top of a conventional copy-edit-merge collaborative software

design environment. Figure 2 depicts the high-level architecture of FLAME. On the architect-side, FLAME attaches a modeling tool-specific adapter, *FLAME Adapter*, to the modeling tool to capture each operation as it is made via the modeling tool's APIs. A *FLAME Client* is installed at each architect's location to establish a channel between the architect-side and the server-side through which the captured operations can be sent. For proactive conflict detection, each captured operation is immediately forwarded from the *FLAME Adapter*, through the *FLAME Client*, to the server-side.

The server-side *Client Manager*, which manages the connections between the server-side and *FLAME Clients*, receives the operation and forwards it to *Detector Manager*, which subsequently replicates and broadcasts the operation to all connected *Detection Engines*. A *Detection Engine* is similar to a *FLAME Client* in the way that it is connected to an instance of the modeling tool with a local copy of the model internally, but a *Detection Engine* does not have an architect behind the modeling tool initiating operations. Instead, it has an off-the-shelf conflict detection tool plugged into the modeling tool. An instantiation of FLAME may have more than one *Detection Engine*, each of which has a different conflict detection tool and may maintain a different version of the model.

When a *Detection Engine* receives an operation that has been broadcast by the *Detector Manager*, the *Detection Engine* applies the operation to its local model, automatically invokes the conflict detection tool, and analyzes the outcome as an architect would do. The result of the analysis is then delivered back to the architects via FLAME in the reverse order to that described above, i.e., from the *Detection Engine*, via the server-side components *Detector Manager* and *Client Manager*, and eventually to the architect-side *FLAME Clients*.

FLAME employs remote nodes to perform higher-order conflict detection in order to offload the potentially resource-intensive computations necessary for the detection from the architect-side or the server-side machines. If a computation-intensive type of conflict detection (recall Section II) were to be performed on an architect's machine or on the server for every modeling operation, it could overwhelm the machine and disturb the collaborative design activity. FLAME therefore moves the burden to the *Detection Engine*.

FLAME can utilize more than one *Detection Engine* in concert to parallelize the higher-order conflict detection. An architect may need to perform multiple conflict detection activities using several tools that implement different techniques (e.g., a combination of static and dynamic analysis) or different instances of the same technique (e.g., reliability and latency analysis via discrete event-based simulation [31]). FLAME distributes these conflict detection activities to multiple remote nodes on a cloud. It instantiates multiple *Detection Engines*, each of which is responsible for performing a single higher-order conflict detection activity using the corresponding conflict detection tool. This aspect of FLAME's architecture allows different *Detection Engines* to be instantiated as needed, possibly even at runtime. We should also note that the network delay that is introduced by distributing the conflict detection activities to multiple nodes is negligible compared to the amount of time necessary for actually performing the higher-order conflict detection.

C. Implementation of FLAME

In order to evaluate whether or to what extent proactive design conflict detection may impact the cost of collaborative software design, we implemented FLAME based on the architecture introduced in Section III-B. Figure 3 depicts FLAME's detailed, as-implemented architecture. It integrates three off-the-shelf software tools: (1) GME [32], a configurable software modeling tool for domain-specific modeling, (2) XTEAM [31], a model-driven design, analysis, and synthesis tool-chain, and (3) Prism-MW [33], an event-based middleware platform.

FLAME connects those off-the-shelf tools together to provide proactive conflict detection to software architects. GME allows architects to create a domain-specific modeling notation (e.g., for NGCA) in which the architects can specify different aspects of the target system. An architect modeling in FLAME uses GME to specify the structure of the system by creating a set of components, connectors, and the connections between them. She then specifies, for each component and connector, (1) how it stores data, (2) how it behaves and reacts to different events, (3) on which physical host it is deployed, and (4) other characteristics in the form of property lists. Each modeling operation the architect makes along the way is captured by the *FLAME Adapter* that immediately transfers the operation, through the server-side FLAME components, to *Detection Engines*, via Prism-MW. Prism-MW establishes event-based interaction channels for the transfer of the operations in FLAME. When a *Detection Engine* receives an operation, it applies the operation to its local model and invokes XTEAM to analyze the model and estimate one or more runtime properties of the modeled system such as memory usage (as in [31]), energy consumption (as in [34]), and message latency (as in [35]). The model analysis result produced by XTEAM is raw and needs to be consolidated since it could distract the architects if provided as-is. For example, the memory usage estimation outputs a simple memory usage log for each component and connector of the target system during the runtime simulation execution. The *Detection Engine* responsible for the memory usage estimation consolidates the result by computing the statistics necessary to determine whether a consistency rule regarding the memory usage has been violated, and forwards it to *FLAME Clients*, where the result is processed further before being eventually presented to architects, as described below.

FLAME provides an extension point at which a customized GUI capable of presenting the domain-specific conflict information can be plugged-in. In our current FLAME instance, as a proof-of-concept, we implemented a small, always-on-top GUI built to minimize the obtrusiveness while continuously delivering conflict information to the architects. The GUI uses color coding to indicate the presence of a higher-order conflict. For example, recalling our NGCA scenario, the GUI in Figure 4 changes the color of the "Memory" indicator from green to red (darker highlighting in grayscale) if the estimation surpasses the threshold.

IV. Evaluation

We conducted a user study using FLAME, with the primary goals (1) to assess how much earlier higher-order design conflicts can be detected and resolved by implementing proactive conflict detection as opposed to solutions that rely

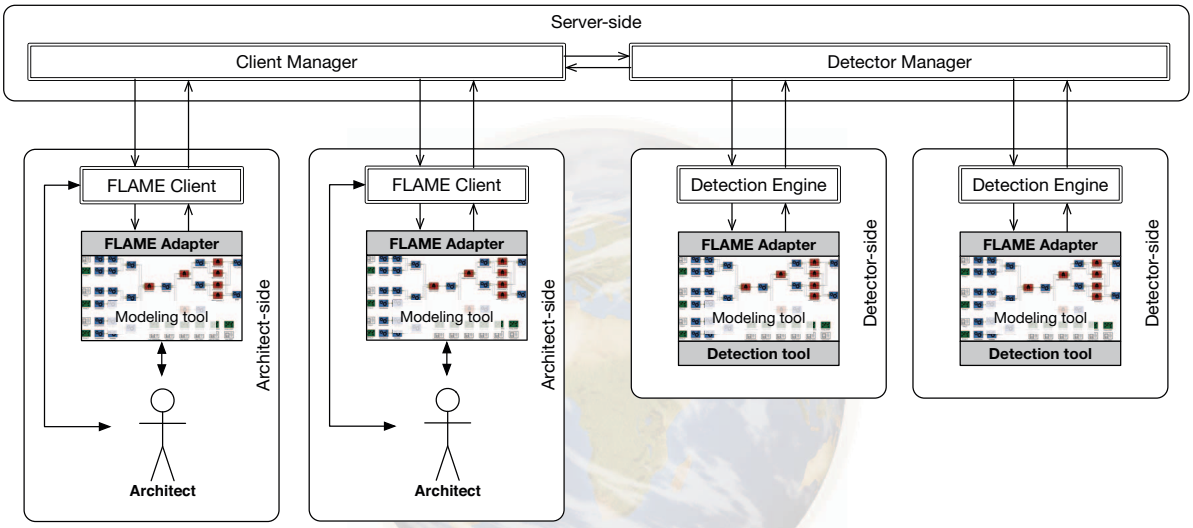


Fig. 2: High-level architecture of FLAME with two architects and two Detection Engines. Gray polygons are design environment-specific components.

on the traditional on-demand merging of models, and (2) to assess whether or to what extent proactive conflict detection impacts the cost of collaborative software design. We had two groups of participants each of which performed collaborative design tasks with and without proactive conflict detection, and compared the results. In this Section, we describe how we setup this user study, present the results by answering three questions, and discuss the threats to validity.

A. Study Setup

We based the user study on NGCA's design documents by recreating a collaborative software design scenario where higher-order conflicts arise. Two groups of participants performed design tasks with FLAME in its two modes: one that does and another that does not present the proactive conflict detection results to architects. By using FLAME for both groups, we were able to track the participants' collaborative design activities at the level of each modeling operation and were also able to track the higher-order conflicts from their creation, to detection, and eventually to resolution for both groups. To compare the cost of collaborative software design with and without proactive conflict detection, in this study, we estimated the cost by measuring (1) the extent of time the architects spent performing collaborative design activities and (2) the quality of the resulting model upon the completion of the design task. Those two measurements were further divided into granular dependent variables, shown in Table I, which will be discussed in detail later in this section.

The participants in our study were 42 students enrolled in the graduate-level Software Architecture class at the University of Southern California. No participant had prior experience with FLAME or the NGCA system. The participants spent four weeks performing two software design assignments using the NGCA-specific modeling environment and FLAME's constituent XTEAM subsystem in order to get familiar with its simulation-based analyses prior to the user study. This resulted in comparable familiarity of each participant with the modeling environment and the domain of the target system [36].

The participants were grouped into 21 teams of two. The teams were then divided into two groups by their team numbers (odd/even, randomly assigned): (1) the control group that used FLAME in the mode that does not present the proactive conflict detection results to simulate the behavior of a current generation software model VCS ("w/o PCD" in Table I) and (2) the experimental group that used FLAME in the mode that does present the proactive conflict detection results ("w/ PCD" in Table I). We also surveyed the participants' industry experience to assess their prior exposure to collaborative software design, but did not find that the prior industry experience of the two groups differed significantly.

Each team participated in a 2-hour-long session during a span of 18 days. The paper's first author administered all 21 sessions. Each session was divided into three smaller sessions: (1) the 1-hour-long FLAME tutorial, (2) the main, 30-minute-long design session during which we recorded the participants' design activities, and (3) the subsequent 30-minute-long design session for the participants to experience the alternative mode of FLAME. The rationale was for the participants to experience both modes of FLAME, without and with proactive conflict detection, and to collect their preferences of and assessment of the differences they experienced between the two modes. We did not record the design activity during the alternative-mode session since the participants' behavior in that session may have been influenced by having undergone the main session.

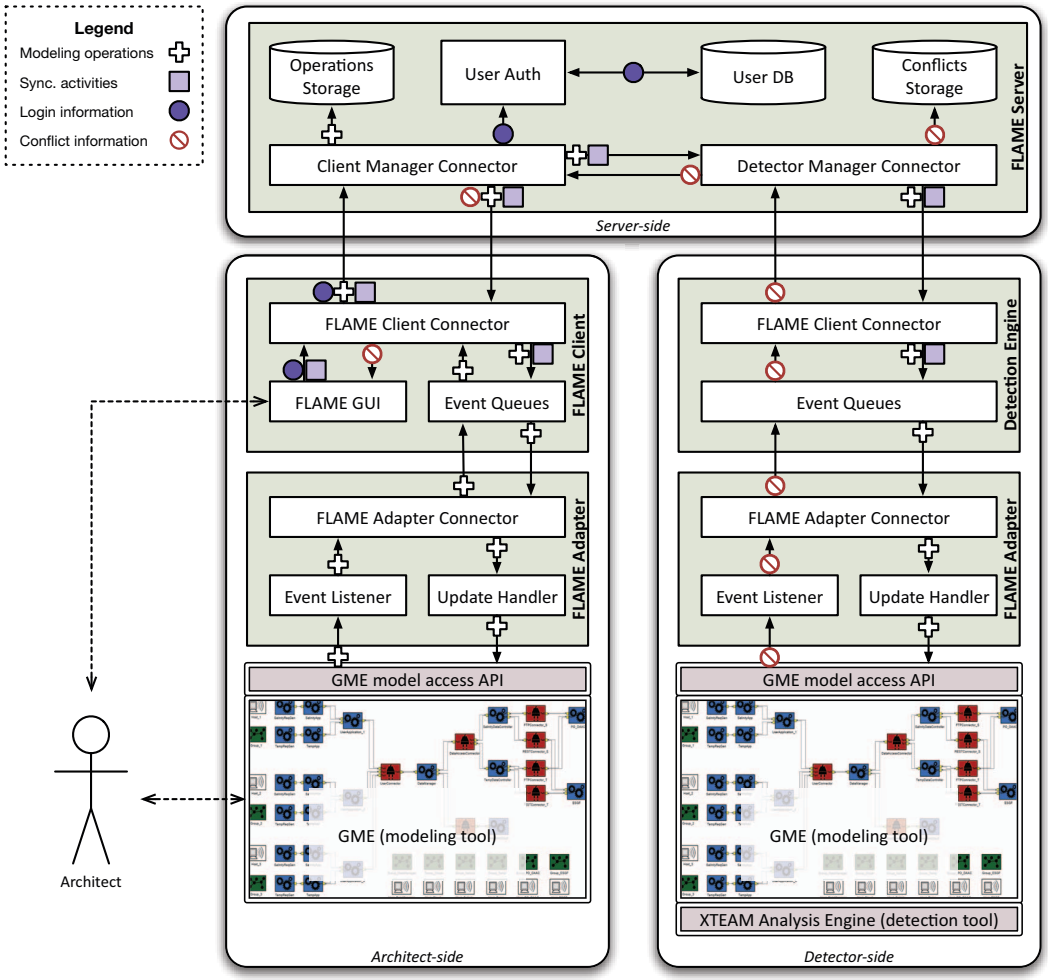


Fig. 3: Detailed architecture of FLAME. Double-lined polygons are off-the-shelf, domain-specific software integrated into FLAME.

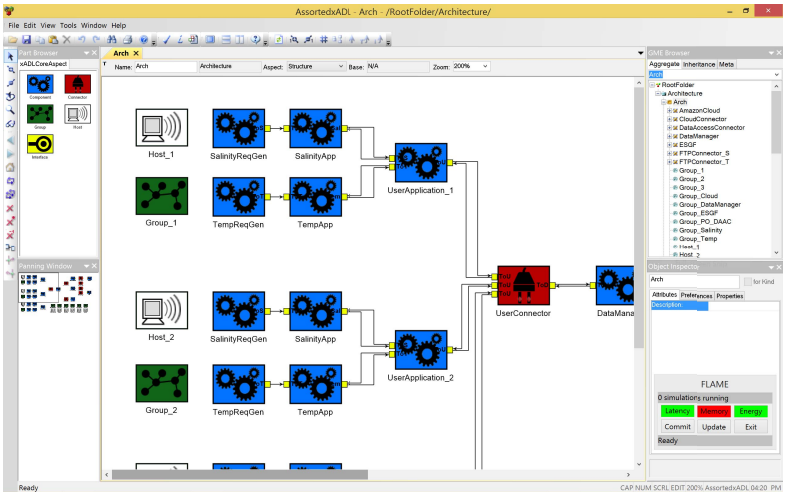
During the design sessions, each team was given a partially complete NGCA model and assigned with software design tasks to replace a set of components in the model, as well as a set of three system requirements regarding the three NGCA runtime system properties to satisfy. The two participants in each team were directed to make trade-off design decisions to two different non-overlapping parts of the model in order to avoid synchronization conflicts. The given tasks were designed in a way that the participants, in the course of decision making, could violate two of the system requirements: energy consumption and memory usage. The third requirement, message latency, was not designed to be violated.

Also, we restricted the communication between the two participants in each team to online communication media during the sessions to reproduce the communication challenges of geographically distributed collaborative software design. For example, the two participants were not allowed to speak with each other but had to initiate an email thread or use an instant messenger in order to discuss their conflict resolution strategy.

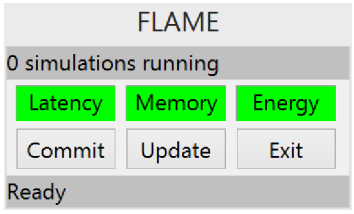
B. Findings, Insights, and Implications

We present the analyses of the user study data by addressing three research questions derived from the two primary goals of the study, defined at the beginning of this section.

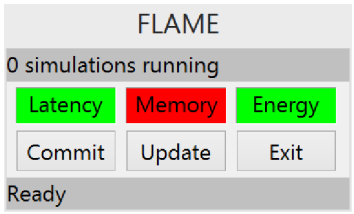
Q1: Did FLAME affect the amount of time architects spend in design activities? The top portion of Table I shows the frequency of design activities (DV01-DV03) performed during the main design sessions. During the same length of time (CV1; 30 minutes), the frequency of design activities differed significantly between the two groups of participants. Specifically, the group with proactive conflict detection (1) performed a higher number of modeling operations and (2) communicated more frequently. The increase in the number of operations can be explained by the increase in the confidence of participants in making new changes. Fear of conflicts [15] could make an architect take additional care when she makes new operations. The following quote is from a participant, which aligns with our reasoning: *"our confidence that the combined design would meet the requirements was much higher when using proactive conflict detection."* Second,



(a) An architect's screen with the FLAME GUI.



(b) All property requirements are satisfied.



(c) The memory requirement is *not* satisfied.

Fig. 4: FLAME GUI for NGCA.

the increase in communication frequency aligns with a previous empirical study in which a similar phenomenon was observed for detection and resolution of higher-order conflicts in collaborative software implementation [16]. One of the participants responded about how the increased amount of communication has helped her team by saying: *"I prefer the proactive environment because my teammate and I completed the task on time as we had enough communication."*

Q2: Did FLAME facilitate the way architects detect and resolve the higher-order design conflicts? The middle portion of Table I shows the number of conflicts that occurred and how long they lasted during the collaborative design sessions (DV04-DV07). We observed that, while the group of participants with proactive conflict detection dealt with a higher number of conflicts on average, no conflict was left in the last commit nor at the end of session (DV05-DV06). Furthermode, the average lifetime of the conflicts, from when they are introduced to when they are resolved, was significantly shorter (DV07). These results also corroborate those reported in the previous research conducted at the code-level [16], [21]. The following quotes from the participants further explain our observation: *"It was quicker and easier to detect conflicts and fix them immediately."* and *"[FLAME was] making it easier to identify errors and fix them before further changes are made."*

Q3: Did FLAME affect the quality of the resulting model? The bottom portion of Table I shows the two factors (DV08-DV09) we used to estimate the quality of the resulting model in addition to the number of unresolved conflicts (DV05-DV06). The participants were assigned with design tasks to modify the system in a way that would maximize the *throughput* of the system, which subsequently results in higher energy consumption and memory usage. We tracked the variations of those two factors from the beginning to the end of each collaborative design session, and recorded the maximum values of the factors that the team reached during the session. We observed that the group of participants with proactive conflict detection were able to design, on average, NGCA systems with higher throughput while leaving fewer unresolved conflicts than the group without proactive conflict detection, in the same amount of time (CV01). The increased number of operations (as shown in Q1) can be seen as evidence of the higher productivity of the participants. The following is a quote from a participant that can show the link between the number of operations and the participants' productivity: *"[FLAME] increased productivity as we were able to try more combinations [of modeling operations] in same amount of time."* The reduced effort in higher-order conflict resolution (as shown in Q2) could also have contributed in the higher productivity. The following quotes from the participants corroborate this conclusion: *"... proactive conflict detection drastically minimizes the integration effort."* and *"[FLAME] shows my partner and me any conflicts that we have without running the simulation as much as we did with the one without proactive [conflict detection]."*

C. Threats to Validity

As is commonly the case with controlled experiments, our user study has threats to its validity due to its design. First, our user study was conducted with students. While all students were graduate-level, their design behavior may not be identical to that of a real-world practitioner. Second, the design tasks assigned to the participants were not from an actual project. Instead, in order to recreate a realistic collaborative design scenario, we based it on the NGCA design documents. Third, the duration of observed design session per team was short (30 minutes), which could have caused bias from low familiarity with the target system domain (NGCA) or the model analysis framework (XTEAM). We minimized the bias by having the participants perform multiple design assignments using the NGCA model and XTEAM over the span of four weeks prior to their sessions. We also believe the higher-order conflicts may persist even longer in design sessions of longer duration, which

TABLE I: User Study Results

Collaborative Design Activities	w/o PCD	w/ PCD
CV01: Duration (minutes) of modeling session per team	30.00	30.00
DV01: Number of modeling operations made per team	48.18	60.80
DV02: Number of communication activities per team	11.00	19.50
DV03: Number of synchronizations per team	6.18	8.00
Conflicts	w/o PCD	w/ PCD
DV04: Number of detected conflicts at synchronizations per team	1.27	2.40
DV05: Number of teams with unresolved conflicts at last commit	3 of 11	0 of 10
DV06: Number of teams with unresolved conflicts at session end	3 of 11	0 of 10
DV07: Lifetime (seconds) of a higher-order conflict	671.00	363.40
Resulting Model Quality; the higher is better	w/o PCD	w/ PCD
DV08: System throughput factor: energy consumption	8182862	8547929
DV09: System throughput factor: memory usage	729.09	747.60

CV is a control variable, and DV is a dependent variable. "w/o PCD" and "w/ PCD" stand for without and with proactive conflict detection respectively. All values were rounded off at the third decimal.

would only increase the benefit of detecting them early. Last, the team size was small (two per team). In a bigger team, it may become ambiguous which architects are directly involved in a higher-order design conflict. This is a hard problem in general, and has not yet fully been answered.

Some aspects of the study execution were challenged by threats to validity. First, we did not vary the amount of time it takes a conflict detection tool to complete its model analysis, while it may influence the architects' reaction to proactive conflict detection. In our user study, we kept the conflict detection time relatively constant (38 seconds on average) in order to avoid introducing bias. Second, only a single kind of analysis tool (i.e., XTEAM) was used in the user study. In a real-world setting, architects may work in a design environment using several model analysis tools. We tried to recreate a more realistic design environment by integrating three XTEAM model analysis tools in FLAME. Third, we have not conducted a full-fledged scalability evaluation of FLAME. While migrating the conflict detection to *Detection Engines* would prevent the architect-side or the server-side machines from being resource-starved, there is another risk that each *Detection Engine* may become a bottleneck in conflict detection when more than two architects are simultaneously performing modeling operations. We believe this could be mitigated by involving additional, remote nodes dedicated to perform conflict detection, further offloading the burden of the detection from the *Detection Engines* to those nodes.

V. Related Work

The risk of having conflicts from using a copy-edit-merge style VCS is not unique to collaborative software design. Collaborative software *implementation* faces an analogous challenge at the level of source code. A number of techniques and tools have been reported, including those for proactive conflict detection [9].

Providing *workspace awareness* is an extensively studied aspect of conflict avoidance and detection. Workspace awareness is "the up-to-the-minute knowledge of other participants' interactions with the shared workspace" [37]. FASTDash prevents potential conflict situations (e.g. two developers editing the same file) by providing a visual presentation of the developers' activities on shared files [38]. Some workspace awareness tools analyze dependencies between program elements (files, types, or methods), and notify developers of conflicting changes made to elements depending on each other [23]. Palantír shows who edited which shared artifacts, in a less obtrusive way by integrating the presentation into the development environment [20]. Syde [39] informs developers of concurrent changes by maintaining an abstract syntax tree of the target object-oriented system, interpreting code changes into tree operations, and using them to filter conflict information. Tools in this group primarily detect synchronization conflicts and dependency-based higher-order conflicts.

Another group of proactive conflict detection tools perform deeper analyses such as compilation, unit testing, and so on. Safe-commit [22] proactively identifies "committable" changes that will not make test cases fail by running them in the background. Two tools in this group, Crystal [40] and WeCode [21], are closely related to FLAME. Both of them proactively perform merging, compilation, and testing of new changes developers make to source code in the background and notify the developers if any of the steps fails.

FLAME differs from them in two ways: (1) exploiting its event-based architecture, FLAME integrates off-the-shelf higher-order conflict detection tools and offloads the potentially resource-intensive conflict detection, and (2) it synchronizes and is capable of performing conflict detection per modeling operation, which enables earlier conflict detection and pinpointing the specific operations that caused a conflict [17].

VI. Conclusion

Higher-order software design conflicts are inevitable. The asynchrony of today's copy-edit-merge software model VCSs exposes the architects to the risk of making modeling changes without being aware of the presence of a higher-order conflict. That may lead to the reversal of those changes in the process of resolving the conflict, which results in wasted effort. In this paper, we presented our solution, FLAME, an extensible, operation-based collaborative software design framework that proactively detects higher-order software design conflicts. FLAME minimizes the risk by performing a trial synchronization and detection of higher-order conflicts for each modeling operation in the background. The architecture of FLAME is novel and specifically designed to support collaborative software design. FLAME integrates the appropriate modeling tool and consistency checkers for the domain of the target system, and offloads the potentially computationally-expensive conflict detection activities to a cloud, exploiting FLAME's event-based architecture. We also presented the result of a user study we conducted with 42 participants using FLAME. During the study, we observed that the participants who were provided with proactive conflict detection (1) had more opportunity to communicate with each other, (2) detected and resolved higher-order design conflicts earlier and more quickly, and (3) produced higher quality models in the same amount of time. To our knowledge, this user study provides the first reported empirical evidence that shows proactive conflict detection positively impacts the cost of collaborative software design. FLAME provides a foundation for exploring several other issues with design-level conflict detection. These include exploring different ways of delivering feedback to architects, the effect of variations in the immediacy with which feedback is delivered, prioritization of analyses and of the delivery of analysis results, and exploration of the utility of proactively analyzing software design models for properties of whose importance the architects may be unaware.

VII. Acknowledgments

This work has been supported by the U.S. National Science Foundation under award numbers 1117593, 1218115, and 1321141, and by Infosys Limited.

References

[1] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[2] P. Kruchten, "The Software Architect," in *Software Architecture*. Springer, 1999, pp. 565–583.

[3] M. Cataldo, C. Shelton, Y. Choi, Y.-Y. Huang, V. Ramesh, D. Saini, and L.-Y. Wang, "Camel: A Tool for Collaborative Distributed Software Design," in *Proceedings of the 4th International Conference on Global Software Engineering (ICGSE)*. IEEE, 2009, pp. 83–92.

[4] D. Loksa, N. Mangano, T. D. LaToza, and A. van der Hoek, "Enabling a Classroom Design Studio with a Collaborative Sketch Design Tool," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Press, 2013, pp. 1073–1082.

[5] M. Basheri and L. Burd, "Exploring the Significance of Multitouch Tables in Enhancing Collaborative Software Design Using UML," in *Proceedings of the Frontiers in Education Conference*. IEEE, 2012.

[6] J. Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. M. Rama, and S. Padmanabhuni, "CoDesign: A Highly Extensible Collaborative Software Modeling Framework," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, vol. 2. ACM, May 2010, pp. 243–246.

[7] K. Altmanninger, M. Seidl, and M. Wimmer, "A Survey on Model Versioning Approaches," *International Journal of Web Information Systems (IJWIS)*, vol. 5, no. 3, pp. 271–304, 2009.

[8] J. Bang, I. Krka, N. Medvidovic, N. Kulkarni, and S. Padmanabhuni, "How Software Architects Collaborate: Insights from Collaborative Software Design in Practice," in *Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, May 2013, pp. 41–48.

[9] A. Sarma, D. Redmiles, and A. van der Hoek, "Categorizing the Spectrum of Coordination Technology," *IEEE Computer*, 2010.

[10] T. N. Nguyen, E. V. Munson, J. T. Boyland, and C. Thao, "An Infrastructure for Development of Object-Oriented, Multi-Level Configuration Management Services," in *Proceedings of the 27th International Conference on Software Engineering (ICSE)*. ACM, 2005.

[11] K. Altmanninger, G. Kappel, A. Kusel, W. Retschitzegger, M. Seidl, W. Schwinger, and M. Wimmer, "AMOR – Towards Adaptable Model Versioning," in *Proceedings of the International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS*, vol. 8, 2008, pp. 4–50.

[12] M. Koegel, J. Helming, and S. Seyboth, "Operation-based Conflict Detection and Resolution," in *Proceedings of the ICSE Workshop on Comparison and Versioning of Software Models*. IEEE Computer Society, 2009, pp. 43–48.

[13] A. Mehra, J. Grundy, and J. Hosking, "A Generic Approach to Supporting Diagram Differencing and Merging for Collaborative Design," in *Proceedings of the 20th International Conference on Automated Software Engineering (ASE)*. ACM, 2005, pp. 204–213.

[14] J. Bang, D. Popescu, and N. Medvidovic, "Enabling Workspace Awareness for Collaborative Software Modeling," *The Future of Collaborative Software Development at the ACM Conference on Computer Supported Cooperative Work (FutureCSD)*, February 2012.

[15] Y. Brun, R. Holmes, M. Ernst, and D. Notkin, "Early Detection of Collaboration Conflicts and Risks," *IEEE Transactions on Software Engineering (TSE)*, vol. 39, pp. 1358–1375, October 2013.

[16] A. Sarma, D. Redmiles, and A. van der Hoek, "Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2008.

[17] X. Blanc, I. Mounier, A. Mougenot, and T. Mens, "Detecting Model Inconsistency through Operation-based Model Construction," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*. IEEE, 2008, pp. 511–520.

[18] C. A. Mattmann, C. S. Lynnes, L. Cinquini, P. M. Ramirez, A. F. Hart, D. Williams, D. Waliser, and P. Rinsland, "Next Generation Cyberinfrastructure to Support Comparison of Satellite Observations with Climate Models," in *Proceedings of European Space Agency Conference on Big Data from Space (BiDS)*, November 2014.

[19] B. Westfechtel, "Merging of EMF Models," *Software & Systems Modeling*, vol. 13, no. 2, pp. 757–788, 2014.

[20] A. Sarma, D. F. Redmiles, and A. van der Hoek, "Palantír: Early Detection of Development Conflicts Arising from Parallel Code Changes," *IEEE Transactions on Software Engineering (TSE)*, 2012.

[21] M. L. Guimarães and A. R. Silva, "Improving Early Detection of Software Merge Conflicts," in *Proceedings of the 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012.

[22] J. Wloka, B. Ryder, F. Tip, and X. Ren, "Safe-Commit Analysis to Facilitate Team Software Development," in *Proceedings of the 31st International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2009, pp. 507–517.

[23] P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development," in *Proceedings of the 10th European Conference on Computer Supported Cooperative Work (ECSCW)*. Springer, 2007, pp. 159–178.

[24] R. Balzer, "Tolerating Inconsistency," in *Proceedings of the 13th International Conference on Software Engineering (ICSE)*. IEEE, 1991.

[25] M. Goedicke, T. Meyer, and G. Taentzer, "Viewpoint-Oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies," in *Proceedings of the International Symposium on Requirements Engineering (RE)*. IEEE, 1999.

[26] A. Egyed, "Automatically Detecting and Tracking Inconsistencies in Software Design Models," *IEEE Transactions on Software Engineering (TSE)*, vol. 37, no. 2, pp. 188–204, 2011.

[27] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: A Consistency Checking and Smart Link Generation Service," *ACM Transactions on Internet Technology (TOIT)*, 2002.

[28] T. J. Schriber and D. T. Brunner, "Inside Discrete-Event Simulation Software: How It Works and Why It Matters," in *Proceedings of the Winter Simulation Conference*. IEEE, 2005.

[29] J. A. Whittaker and M. Thomason, "A Markov Chain Model for Statistical Software Testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994.

[30] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-Based Performance Prediction in Software Development: A Survey," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, 2004.

[31] G. Edwards, C. Seo, and N. Medvidovic, "Model Interpreter Frameworks: A Foundation for the Analysis of Domain-Specific Software Architectures," *Journal of Universal Computer Science*, 2008.

[32] Institute for Software Integrated Systems, Vanderbilt University, "Generic Modeling Environment," http://isis.vanderbilt.edu/projects/gme/, 2014, [Online; accessed March 2, 2015].

[33] S. Malek, M. Mikic-Rakic, and N. Medvidovic, "A Style-aware Architectural Middleware for Resource-constrained, Distributed Systems," *IEEE Transactions on Software Engineering (TSE)*, 2005.

[34] C. Seo, S. Malek, and N. Medvidovic, "An Energy Consumption Framework for Distributed Java-Based Systems," in *Proceedings of ASE*. ACM, 2007, pp. 421–424.

[35] M. Woodside, "Tutorial Introduction to Layered Modeling of Software Performance. Carleton University," http://sce.carleton.ca/rads/, 2005, [Online; accessed March 2, 2015].

[36] D. Damian, R. Helms, I. Kwan, S. Marczak, and B. Koelewijn, "The Role of Domain Knowledge and Cross-Functional Communication in Socio-Technical Coordination," in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013.

[37] C. Gutwin and S. Greenberg, "Workspace Awareness for Groupware," in *Conference Companion on Human Factors in Computing Systems (CHI)*. ACM, 1996, pp. 208–209.

[38] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 2007, pp. 1313–1322.

[39] L. Hattori and M. Lanza, "Syde: A Tool for Collaborative Software Development," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, vol. 2. ACM, May 2010, pp. 235–238.

[40] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Crystal: Precise and Unobtrusive Conflict Warnings," in *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2011, pp. 444–447.