Bringing Together Configuration Research: Towards a Common Ground

Paul Gazzillo
University of Central Florida
Orlando, FL, USA
paul.gazzillo@ucf.edu

Myra B. Cohen
Iowa State University
Ames, IA, USA
mcohen@iastate.edu

Abstract

Configurable software makes up most of the software in use today. Configurability, i.e., the ability of software to be customized without additional programming, is pervasive, and due to the criticality of problems caused by misconfiguration, it has been an active topic researched by investigators in multiple, diverse areas. This broad reach of configurability means that much of the literature and latest results are dispersed, and researchers may not be collaborating or be aware of similar problems and solutions in other domains. We argue that this lack of a common ground leads to a missed opportunity for synergy between research domains and the synthesis of efforts to tackle configurability problems. In short, configurability cuts across software as a whole and needs to be treated as a first class programming element. To provide a foundation for addressing these concerns we make suggestions on how to bring the communities together and propose a common model of configurability and a platform, ACCORD, to facilitate collaboration among researchers and practitioners.

CCS Concepts: • **Software and its engineering** → **Language features**.

Keywords: configurability, community building

ACM Reference Format:
Paul Gazzillo and Myra B. Cohen. 2022. Bringing Together Configuration Research: Towards a Common Ground. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '22), December 8–10, 2022, Auckland, New Zealand*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3563835.3568737

1 Introduction

Configurable software makes up most of the software in use today. Developers typically expose sets of preferences (or configuration options) which can be selected and combined in different ways, forming unique instances of the program for each combination of settings. Configurable software can be found in applications for a range of purposes, e.g. business, science, entertainment, system administration, health, and embedded devices [8, 9, 12, 16, 24, 25, 28, 31, 34, 42, 44, 53, 58, 63]. On one end of the spectrum, configuration options may be manifested as user-facing settings such as turning JavaScript on or off in a web browser while, on the other end, it could involve a system builder selecting a driver or protocol for a specific hardware sensor or an administrator defining a set of allowable settings that are known to be secure. Configurations can be defined in an ad-hoc manner or in a more formal language, such as those used in `Kconfig` or `sendmail`. It may also mean choosing specific devices and tuning their settings in a mixed environment such as the Internet of Things (IoT), file systems or even surgical robots [36, 53].

In essence, *configurability is pervasive.* At the same time this penchant for customization can have a large impact on software quality across all stages of development, (e.g. design, coding, testing, deployment, etc.) which means that configurability impacts software reliability, correctness, usability, and security. As a case in point, *security misconfiguration* is listed as number five in the Open Web Application Security Project's (OWASP) 2021 top ten list of the most critical security risks [46], up from sixth place in the last report from 2017. OWASP tracks security incidents across half a million applications, and misconfiguration has one of the highest incidence rates across applications, 4.5% on average, with over two hundred thousand incidents in total.

The criticality of problems due to misconfigurations (a general term we clarify later) along with the many facets of software and its life cycle including requirements, development, security and testing, and the fact that it is used in most software domains, means it has been an active topic researched by investigators in multiple, diverse areas such as systems, networking, software engineering, programming languages, machine learning, scientific computing, human computer interaction, and more.

Figure 1 represents different facets (domain, stakeholder and impact) of configurability. The first facet *domain* indicates the reach of configurability into different application sectors. *Stakeholder* captures the human facet: who is interacting with and being impacted by configurability. And

Onward! '22, December 8–10, 2022, Auckland, New Zealand
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9909-8/22/12.
https://doi.org/10.1145/3563835.3568737

last *impact* shows the effect that configurability can have on software (e.g. variable correctness or safety properties). The combinatorics of combining these facets has meant an explosion in research without a central core.

For instance, the software engineering community has studied configuration faults [1, 4, 7, 20, 38, 39, 48, 50, 60–62] and the need to reduce options for usability [4, 27, 28, 51, 58] or to predict performance [3, 23, 26, 42, 52, 55], the security community has looked at impacts on security [16, 19, 21, 40, 47, 53, 56], the systems community has focused on problems related to overall system configuration [10, 18, 49] and even file storage [34].

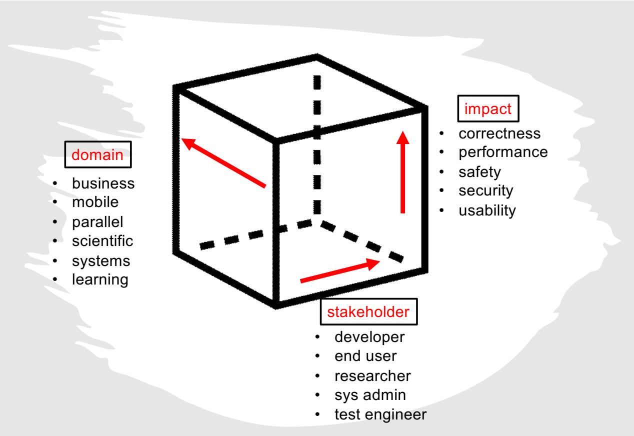

Figure 1. Different facets of configurability. *Domain* includes the application sector, *stakeholders* are those within the software ecosystem who are impacted by configurability and *impact* indicates the effect of varying configurations.

Additionally, there is research from the Human Computer Interaction (or HCI) community [37, 59], and the scientific community which impacts both performance and functionality of their tools [8, 29, 35]. Last, Artificial Intelligence researchers have proposed different ways to achieve optimal parameter tuning [5, 6, 30]. Despite our attempted survey here, we admit we are most likely missing key contributions to this field; it is simply too diverse.

This broad reach of configurability means that much of the literature and latest results for managing its complexity are dispersed, and researchers or practitioners in these different domains may not be collaborating, and/or may not aware of similar problems and solutions in other domains. This can lead both to duplication and missed opportunities to build on state of the art solutions. In short, there is community fragmentation; configurability isn't seen as a *first-class* element, but rather has become an ancillary, yet crosscutting problem.

An exemplar of this fragmentation is its lack of an ACM Content Classification System (CCS) category. CCS contains "Software configuration management and version control systems" and "Software product lines" (one type of configurability) [13], but configuration management has other meanings and product lines encompass one particular paradigm of configurability. Hence, someone trying to understand the wide-ranging effects or research topics on configurability cannot easily find all of this research.

We argue that this lack of a common ground **leads to a missed opportunity for synergy between research domains and the synthesis of efforts to tackle configurability problems**. We need a shared set of solutions.

In short, configurability cuts across software as a whole, going beyond just software engineering. Key challenges we face due to this fragmentation are (1) a lack of common terms and definitions, (2) overlapping research efforts in separate communities, (3) research venues without explicit vehicles for sharing and collaboration on common problems, and (4) solutions that are bespoke for each domain, losing opportunities to build general solutions.

There are several benefits of incorporating configurability into language design. Hence, we view the programming languages community as an important part of this endeavor—the opportunity to build practical and foundational solutions for end-to-end configurability. While this alone will not solve the problem, including configurability in language design will (1) force designers and builders to consider configurability from the beginning, (2) allow modeling languages that are used to reason about the variability space to be leveraged, and (3) add expressiveness that can be used by both static and dynamic analyses to facilitate finding misconfiguration bugs, which have become highly prevalent, yet are notoriously difficult to find since they involve the interaction between the language, build system, and other software infrastructure. Last, (4) performance (or other) metrics can be assigned to code, models and intermediate representations, creating end-to-end traceability.

To provide a foundation for addressing these concerns we propose a common model and suggest a software development paradigm that makes configurability first class (Section 3), along with a platform to facilitate collaboration among researchers and practitioners (Section 4).

2 Motivating Examples

We now illustrate how configurability interacts with software using a security example. In this example, a fault in the configuration code opens the application up to memory leakage. `Optionsbleed` was an exploit in the Apache webserver that "bleeds" arbitrary memory contents to a remote attacker [15, 21]. It demonstrates how unexpected configuration settings can lead to problems. This fault only manifests when an administrator configures a restriction on which HTTP methods are available to a client. The `Limit` directive (Figure 2a) restricts access to the specified HTTP methods, a useful setting for securing a server. In this case, a restriction was meant to be placed on PUT, DELETE, and BIND. However, DELETE is incorrectly spelled as DELTE.

```
<Limit PUT DELTE BIND>
</Limit>
```

(a) The (mispelled) .htaccess file.

```
./configure --enable-dav
```

(b) The build option that compiles the WebDAV module.

```
a2enmod dav
```

(c) The tool that enables the WebDAV module.

Figure 2. The three separate configuration mechanisms involved in Optionsbleed. Unless both (b) and (c) are configured, (a)'s use of BIND will expose Optionsbleed.

But the validity of the HTTP method also depends on what extensions have been configured into the server, e.g., `BIND` is only available if WebDAV support is enabled by the build configuration script (Figure 2b) and run-time configuration tool (Figure 2c).

Each HTTP method can be enabled independently (and combined arbitrarily), therefore we would consider each of these a configuration option with the possible values of on or off. Other configuration options can have multiple values, for instance, if the various HTTP methods were exclusive and only one could be selected at a time. In this system, providing an unknown HTTP method (i.e. using the wrong spelling for the method, e.g., `DELTE`, or if for any other reason the extension is not actually built into the server code), causes a use-after-free bug in the server's HTTP method handling code. Therefore, `BIND` also causes a use-after-free, but only when WebDAV is enabled, since it depends on WebDAV.

This is a classic case of a *misconfiguration* and is due to the fact that the developer did not add code to ensure that the user can only select valid configuration values. The challenge is that the absolute choice of values is dynamic and dependent on other aspects of the system; it is hard for the developer to encode all options and easily validate a user's choice. Ironically, security best practices recommend disabling unneeded features to reduce the attack surface [46], which in this case *increases* the possibility of triggering the leak. This illustrates the need to identify and constrain configuration options just as one would when performing type checking.

Since it triggers a use-after-free, Optionsbleed is in some sense just a traditional software fault. It is sometimes also called a feature interaction, or simply an interaction fault, another confusion in the broader community. This fault only exists in the software under certain configuration settings, hence we use the term *misconfiguration.* Without the misconfiguration, there is no code path that an attacker could ever use to trigger the bug. This distinction is subtle: a traditional fault is a fault because the software violates the specified behavior of the program; in this case, accessing the memory of freed pointers is a typically a violation of correctness. On the other hand, the only reason the fault is feasible is because of a configuration setting that is not supposed to be possible, i.e., referring to an non-existent HTTP method.

To further tease out the distinction between configuration-related faults, we can look at how a developer fixes the problem. Do they just modify the program code? Do they update the installation instructions or modify the build system to make such configuration settings impossible to make? For Optionsbleed, of course, modifying the pointer-handling code to prevent the use of the dangling pointer would close the leak. But the problem of misconfiguration would still remain: how should the software behave when given an unsupported HTTP method? If the build and configuration system enforced explicit specifications of what configuration settings are valid, then the vulnerability would not have been reachable in the first place.

To see this in another context, take the SSH daemon root port privilege escalation vulnerability [14]. This case involves two configurable options, one to run the daemon as root and one to allow forwarding to a local socket. Both features work correctly independently. When used together, however, this leads to a privilege escalation since users can run programs as root, a non-functional program fault. This fault relies on a security policy or an oracle to validate the correct privileges. Without an explicit specification of what configurations are secure, the consequences of their unexpected combination are difficult to reason about. While the developer could have decided that the configuration was permissible and rely on the user to harden their daemon, the developer instead decided that combining these options should not be allowed by any user, i.e., a misconfiguration, because of the security implications citesshbug.

2.1 Configuration Options as Program Values

These misconfigurations might have been framed as traditional program correctness or security properties, where configuration options are just program inputs. But configuration systems are not always implemented in the same language as the software itself (or in a programming language at all): configuration systems may involve compile-time metaprogramming, i.e., macros or conditional compilation. Moreover, configuration options may be persistent across an entire system of separate programs and devices, such as UNIX environment variables.

Therefore, we define *configuration options* as typically long-lived, global values for an entire software system that are set once and either retain the same value during execution (in the case of Optionsbleed) or changed via a well-defined settings menu in interactive programs.

A configuration option is then a special kind of program value that exists outside of any particular program and may affect program behavior by altering what the program *is* versus what the program does during a particular execution. In this way, configuration options can turn an individual program into a family of closely related programs.

This leads us to posit that configuration options and their specifications, are distinct, first-class concepts in software engineering. *While there is overlap between software and configuration specification, we believe there is a benefit to providing developers with methods to make this distinction explicit in their software.*

The activity of defining software configurations is common and distinct enough, the problem space is unique enough, and the potential benefits are strong enough to warrant providing developers these tools for defining and implementing their software configurations. Akin to types, first-class objects, or first-class functions, first-class configuration options provide developer a way to explicitly specify software configurations and to give researchers information about developer intent when designing algorithms, analyses, tools, etc.

3 Towards a Common Ground

To help describe the disparate communities affected by configurability and who are working on related challenges, we present a model that encompasses a wide range of research on configurability and translates across domains. There are multiple views of configurable software, including the layers of configurable software itself, what parts of the software development lifecycle are involved in them, and what kinds of faults emerge from configurable software.

Figure 3 presents the ideas of configurability using this model. In the middle is the configurable software itself, viewed here as taking a set of configuration options (also called features, settings, etc.) and producing a system of software variants, one for each software component in an entire system. For instance, a web service may consist of a configured kernel, webserver, and application, and an IoT system may consist of a number of different smart devices configured for a specific type of home. We identify four main components that group together the many manifestations of configurable software: User and developer specifications, the configuration implementation, and the program code itself. This grouping covers many widely-varying kinds of configurable software. For instance, specifications may be formal feature models, UNIX-style `.conf` files, or just informal README files. We separate user and developer specifications, because users may have their own constraints on what are their acceptable configurations that developers still allow for other users. For instance, an operating system developer may permit any kind of password policy, while one particular system administrator (here a user of the system software) may impose their own password policies.

The configuration implementation component refers to how configuration options are used to select a program variant. These may be implemented with compile-time techniques, such as Makefiles in C software, or run-time techniques such as feature flags in Firefox. Finally, the program code is the part of the software that is oblivious to configurability. It is the set of all possible variants that the configuration implementation chooses from. With compile-time configurability, the program code may be in a different language from the configuration implementation, e.g., C versus Makefiles. But the two might also be blended, as in the case of Firefox, where the distinction is only captured in the control flow of developer-defined configuration options versus program input like the search bar.

Each of these groupings corresponds to some phase(s) of the typical software development lifecycle. The specification components are usually in the realm of requirements engineering, while the configuration implementation and programming are part of development. Maintenance spans specifications and implementation, since fixing bugs or adding features may involve modifying either. Testing spans nearly the entire stack: configuration testing might read the specifications to devise tests, unit testing might only involve program code, and system testing might use the deployed software variant.

3.1 Explaining Misconfigurations using the Model

This model provides a common ground for translating work on configurable software, because each grouping maps to the many research and industry efforts on configurable software. For instance, research on configuration testing algorithms can apply to any configuration specifications, be they compile-time or run-time. Performance testing in the biology domain can benefit from techniques developed in the feature-oriented software domain and vice-versa.

In addition to providing a common ground for translation across domains, our model also provides a more specific explanation of *misconfiguration bugs*. Misconfiguration bugs are currently a hodge-podge of various kinds of software faults [45], for instance, a NULL pointer error in the Linux kernel source that occurs only under x86, when NUMA and PCI are enabled [1, 2] or a build that fails to compile. A misconfiguration is used for any fault, security problem, or performance issue with a wide variety of differing examples, such as a failure to change a default password or a complex interaction between differing configuration options as in SSHD privilege escalation. They can also be ordinary-seeming faults such as a divide by zero, but one that can only be triggered under specific configurations.

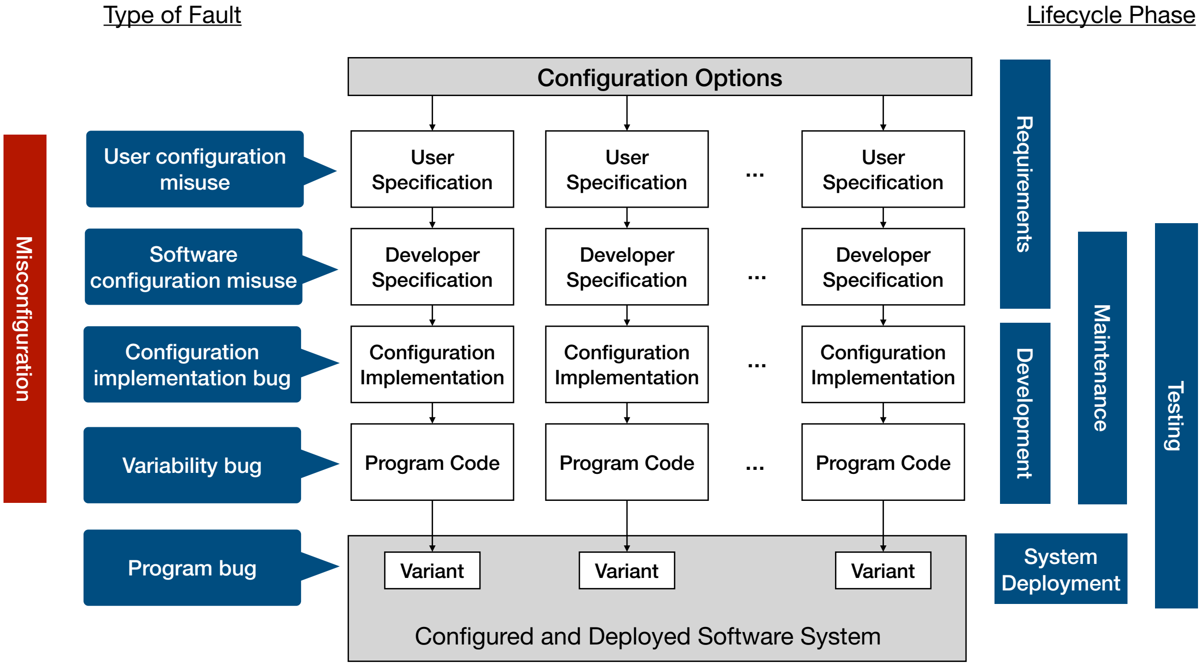

Figure 3. Configurable software components, faults, and lifecycle phases.

Table 1. Examples of configuration-related bugs.

Example	Model Component	Fault Type	Fault Description
Plaintext logging [11, 32, 57]	User specifications	User configuration misuse	Configuration violates user's own policies
sshd privilege escalation [14]	Developer specifications	Software configuration misuse	Configuration should never be permitted
Unimplemented Firefox option [4]	Configuration implementation	Configuration implementation bug	Program code is missing
Kernel panic in a variation [1, 2]	Program code	Variability bug	Correctness of the variant is violated

Table 1 lists different misconfiguration faults, a description, an example, and what component of the Common Model in Figure 3 it corresponds to. A *user configuration* misuse is a violation of the user's own configuration specification. For example, when Facebook, Twitter, and others inadvertently left plaintext logging on their development servers, they violated their own security policies [11, 32, 57]. The fix may not involve removing these options from the web server software, since such logging is still useful for development servers; it is a violation of the software user's specification, not the developer's.

The SSHD privilege fault [14] is an example of a fault in the developer's specification. Since the developer decided that the combination of options should never be permissible by any user, the fix was made in the configuration checking code that all user's of the software are subjected to, rather than advising user's to avoid the options via a hardening guide or configuration checker.

The unimplemented configuration option in Firefox is an example of a bug in the code that implements configuration options, since the program code was available and the configuration option was selectable, but the connection between the two was broken by the implementation; the code was had been deleted [4].

Finally, bugs in the program code are essentially indistinguishable from conventional bugs that occur when there is no configurability. For instance, the kernel panic in this example was caused by a NULL pointer error in the program code [1, 2]. The selection of options was correct, the implementation chose the right program code, but the implementation of the program had a bug. Sometimes, such bugs are called variability bugs, not because they are caused by the configurability of the system, but because they may only appear in some configurations of the software.

4 Bringing the Communities Together

The use of configurable software is only increasing and the gaps between researchers and practitioners affected by configurability will likely only grow. Our goal is to bring together the multiple affected communities to foster collaboration, share solutions, and increase awareness. Collectively, we can advance the state of the art in configurability research and practice, create stronger communities, and maximize the safety and reliability of configurable software.

To facilitate collaboration, we present *ACCORD*, *A Community for Configurability Open Research and Development*. ACCORD is a framework meant to enable shared infrastructure with community contributions and solutions. As a framework it can be instantiated in multiple ways, such as via online repositories, shared communication channels, and increased interaction between research communities. Over time we envision ACCORD cultivating a common lexicon that embodies configurability, providing a way to share tools and best practices, and housing examples of potential faults or pitfalls (*configuration smells*) in different domains, along with solutions to avoid those pitfalls, i.e. *configuration patterns*.

Possible mechanisms to achieve the objectives of ACCORD are to host workshops co-located in conferences on the affected domains, e.g., systems, bioinformatics, etc., to host a shared, online repository for tools, patterns and best practices, and to organize a periodic summit with representatives from the affected domains.

For instance, if IDEs are built with configurability in mind, then documentation and traceability can be easily implemented. And when we denote a variable as a configuration option, why not enforce its relationships and dependencies across the system as we would a type in any other program? Going one step further, as we utilize common unit testing frameworks, they could provide automated parameterization based on those documented configuration options. But these kinds of solutions take a community, as we will need tools that are not bespoke or programming language dependent, but that work across multiple languages and generalize to different domains, each of which has its own notion and way of viewing and using configurations.

Figure 4 illustrates the connection between some of the communities affected by configurability and a proposed collaborative platform, for instance the programming languages community for building domain-specific languages and new high-level typing constructs, the software engineering community for creating develop tools, and other communities such as artificial intelligence, networking, and systems. We also include stakeholders such as educators and industry partners, who are also important for adoption and training. The ACCORD community would host repositories of domain specific programming languages, tools, models, faults, and labeled data for research on configurability which can be widely used. It has the potential to cross the research discipline divide across key communities where research on configurability has been flourishing, e.g. software engineering, systems, and programming languages.

To make ACCORD succeed, we need common conceptual foundations so that researchers from multiple disciplines can more easily work together and exchange ideas. For instance, elevating configuration options to first-class language constructs will provide explicit, machine-readable expressions of configurability. But this effort requires care to not only add configurability features to languages, but to provide tool support for existing configurable software, research the ramifications of the design, and integrate first-class configurability into existing software processes for both computer scientists and researchers in other fields using software, e.g., bioinformatics.

To provide the foundation for common research efforts, we propose research tasks for making configurability a first-class construct, with a shared set of artifacts for use in collaborative research, promotion of configurability, and education. For example, we can extend existing languages by adding formally-defined comments containing configuration option declarations with their dependency constraints, e.g., javadoc, pydoc, etc.

By extending automated documentation engines that support code analysis, this approach provides a basis for building tooling for several phases of the software development lifecycle, such as combinatorial configuration testing, type-checking, etc. This approach provides a platform for exploring configuration as a first-class construct that can be used to inform future language- and tool-design decisions for integrating software configurability into software tooling.

If we focus on existing solutions for configurability, they are simply not enough. For instance `#ifdefs`, while used extensively for configurability, are not part of the actual C language [22]. Variability calculus [17] provides fundamental constructs for variability, but it is not at the right level of abstraction for most working on configurability. Javadoc provides automated documentation but lacks a way to annotate or indicate configurations as other automated documentation systems, e.g. doxygen [54], do for other program symbols. Instead we need novel, high-level annotations and type systems to bring configurability to the forefront.

4.1 Enabling New Research

By building cross-discipline shareable artifacts, we hope ACCORD will lead to novel research directions. For instance, the systems community will be able to leverage results from software testing and validation to improve system reliability. It will allow systems researchers to build more formal approaches based on the software engineering work. At the

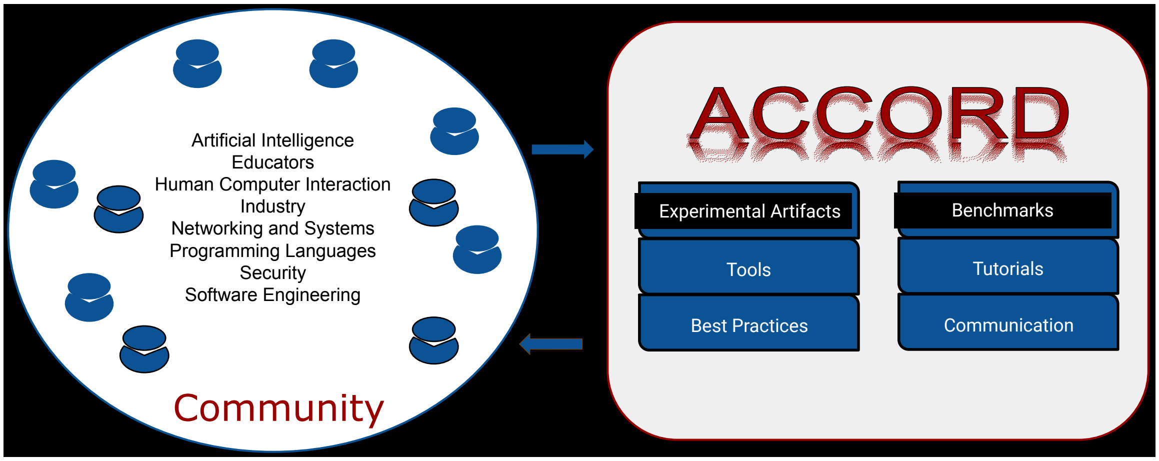

Figure 4. A Community for Configurability Open Research and Development

same time, the software engineering community will have access to larger, more realistic applications with domain knowledge and a better understanding of realistic workloads and use cases. They will also need to consider scalability in ways that they may not have previously considered, as well as issues such as parallelism and threading. By involving the programming languages community, common languages and classifications of types of configuration faults (as well as mitigation by type) will be enabled.

In the security community, research on topics such as debloating to reduce attack surfaces, security analysis (like testing/debugging), and reverse engineering of configuration code will be able to leverage current tools and data sets for experimentation. In general all of the research communities will benefit from reuse and not having to reinvent similar solutions. Take for instance, the area of parameter tuning for machine learning software (one possible community for ACCORD). While hyper-parameters are specific to parameters of the machine learning models, and may not be considered traditional configuration options, the use of several optimization techniques such as grid-search, and the ability to reuse configuration predictions via transfer learning, can benefit researchers in other software domains. At the same time, there are configurable options (not related to hyper-parameters) which behave in a similar way to configuration options in traditional configurable systems and can lead to misconfigurations.

Besides the ability to quickly build on techniques across domains, we plan to build training materials and provide tools that can be used generally for all domains. We provide more concrete examples below of ways that ACCORD can help the larger community.

Programming Language Researchers. Programming language researchers will be able to help make configurations a first-class construct using a common model that can be used to inform future language- and tool-design decisions for integrating software configurability into software tooling.

Systems Builders. This can be a catalyst to improve research from multiple domains; e.g., the systems engineer could utilize predictive models to help pre-screen their system to find areas of concern in a more scalable way. But the tools may not scale to larger systems, driving software engineering researchers to develop better and more scalable predictive approaches. Systems engineers could utilize configuration localization research to help identify the root causes of faults they find, while the software engineers would need to expand these techniques, for instance, to handle dependencies. Having a domain-specific language will allow easier entry into this area and create more cross-fertilization.

Security Engineers. Take an Apache web server misconfiguration which exposes server memory to an attacker that is caused by unchecked configuration files and interactions between software features. By building a community repository for software configuration models, security engineers can pose new configuration problems and leverage configuration testing tools, while software engineering researchers can develop newer and better models and apply them to critical systems software.

Software Engineers. Software engineers who build tools for sampling configuration spaces will have access to real-world models and constraints so that they can validate the scalability and usefulness of their tools. The existence of benchmarks of programs and artifacts will allow them to compare techniques on a wide range of software systems.

4.2 An Example of Collaboration

While ACCORD is still a vision, we present a real use-case with an industrial partner of the first author to motivate the potential benefits for an interdisciplinary approach to solving problems related to configurability. The work began with new techniques for producing more accurate models of the Linux kernel build system. We developed an algorithm and tool to collect a logical model of the configuration specifications from build system code automatically [44]. In order to improve the model, we collaborated with other software engineering researchers to apply the model to a configuration optimization problem [43].

Having a public, reusable modeling artifact enabled research and technology transfer with several other groups across research areas and industry. The first application was to systems software testing. This was a collaboration with testing and analysis researchers, where the tool was repurposed for generating build system configurations to find bugs in multiple configurations [41]. The next collaboration was with Linux kernel maintainers, interested in practical applications of the model to the Linux build system. We observed that the modeling artifact could be used as the basis for finding configuration dependency errors in build system code. This led to research on a bug finder [44] which was integrated into the public repository for the modeling artifact. By demonstrating its capability with accepted patches in the Linux kernel, it developed interest from the maintainers of the Intel 0-day kernel test robot, which runs dozens of analysis and testing tools continuously on the Linux kernel codebase. Our bug finder is now included in their suite of tools, which automatically reports bugs to the Linux kernel mailing list and is responsible for dozens of reports per month [33]. The constant exercise of the tool has led to a stronger artifact, since it identifies issues that we fix in our modeling and bug finding tooling.

There are several key takeaways from this experience that we hope to replicate and expand on with ACCORD. First, collaborating with industry and other stakeholders was synergistic: applying configuration analysis outside of the original research subarea made the modeling algorithms and the resulting artifacts more robust and reusable, since they had to work out-of-the-box for others. Second, having public artifacts enabled research progress in other areas, in this case system development and testing. Using open-source tools for development and issue reporting, enables stakeholders to participate freely. Third, the collaborations led to impact not possible by only publishing in a single research area; for instance, the bug finding tool has led to dozens of patches that fix configuration bugs in real-world code. Additionally, by maintaining a public repository, all users can report issues while the artifact's maintainers can continuously improve the artifacts, which benefits all users and enables smoother future collaborations.

5 Conclusion

Our goal is to promote configurability to a first class element and to provide common ground for all stakeholders, so that we can more easily bring together researchers and practitioners who are typically siloed. We have illustrated what configurability is and provided several examples of how misconfiguration can negatively impact software. We proposed a common software model of configurable software that can help provide a common explanatory framework for the configurable software lifecycle and its impact on software reliability and security. Last, we presented our vision of ACCORD, a way to provide a foundation for collaboration that we hope will continue to grow, providing a platform that accelerates the research and practice related to configurable software.

Acknowledgments

We thank the reviewers for the valuable feedback. This work was supported in part by NSF grants CCF-1941816 and CCF-1909688.

References

[1] Iago Abal, Claus Brabrand, and Andrzej Wasowski. 2014. 42 Variability Bugs in the Linux Kernel: A Qualitative Analysis. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (Vasteras, Sweden) *(ASE '14)*. Association for Computing Machinery, New York, NY, USA, 421–432. https://doi.org/10.1145/2642937.2642990

[2] Iago Abal, Jean Melo, Stefan Stanciulescu, Claus Brabrand, Márcio Ribeiro, and Andrzej Wasowski. 2014. NULL pointer deference due to invalid cast in x86 NUMA. http://vbdb.itu.dk/linux/76baeeb.html. Accessed: 2021-07-08.

[3] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling Effect on Performance Prediction of Configurable Systems: A Case Study. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering* (Edmonton AB, Canada) *(ICPE '20)*. Association for Computing Machinery, New York, NY, USA, 277–288. https://doi.org/10.1145/3358960.3379137

[4] Farnaz Behrang, Myra B. Cohen, and Alessandro Orso. 2015. Users Beware: Preference Inconsistencies Ahead *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 295–306. https://doi.org/10.1145/2786805.2786869

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[6] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13, 1 (feb 2012), 281–305.

[7] Malik Bouchet, Byron Cook, Bryant Cutler, Anna Druzkina, Andrew Gacek, Liana Hadarean, Ranjit Jhala, Brad Marshall, Dan Peebles, Neha Rungta, Cole Schlesinger, Chriss Stephens, Carsten Varming, and Andy Warfield. 2020. Block Public Access: Trust Safety Verification of Access Control Policies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 281–291. https://doi.org/10.1145/3368089.3409728

[8] Mikaela Cashman, Myra B. Cohen, Priya Ranjan, and Robert W. Cottingham. 2018. Navigating the Maze: The Impact of Configurability in Bioinformatics Software. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 757–767. https://doi.org/10.1145/3238147.3240466

[9] Qingrong Chen, Teng Wang, Owolabi Legunsen, Shanshan Li, and Tianyin Xu. 2020. Understanding and Discovering Software Configuration Dependencies in Cloud and Datacenter Systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 362–374. https://doi.org/10.1145/3368089.3409727

[10] Runxiang Cheng, Lingming Zhang, Darko Marinov, and Tianyin Xu. 2021. Test-Case Prioritization for Configuration Testing. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual, Denmark) *(ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 452–465. https://doi.org/10.1145/3460319.3464810

[11] Catalin Cimpanu. 2018. Twitter Admits Recording Plaintext Passwords in Internal Logs, Just Like GitHub. https://krebsonsecurity.com/2018/05/twitter-to-all-users-change-your-password-now/. Accessed: 2019-06-07.

[12] Jane Cleland-Huang, Nitesh Chawla, Myra B. Cohen, Md Nafee Al Islam, Urjoshi Sinha, Lilly Spirkovska, Yihong Ma, Salil Purandare, and Muhammed Tawfiq Chowdhury. 2022. Towards Real-Time Safety Analysis of Small Unmanned Aerial Systems in the National Airspace. In *AIAA AVIATION 2022 Forum*. https://doi.org/10.2514/6.2022-3540

[13] Paul Clements and Linda Northrop. 2002. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA.

[14] CVE. 2016. CVE-2016-10010. https://nvd.nist.gov/vuln/detail/CVE-2016-10010. Accessed: 2021-07-08.

[15] CVE. 2017. CVE-2017-9798. https://nvd.nist.gov/vuln/detail/CVE-2017-9798. Accessed: 2020-06-10.

[16] Constanze Dietrich, Katharina Krombholz, Kevin Borgolte, and Tobias Fiebig. 2018. Investigating System Operators' Perspective on Security Misconfigurations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1272–1289. https://doi.org/10.1145/3243734.3243794

[17] Martin Erwig and Eric Walkingshaw. 2011. The Choice Calculus: A Representation for Software Variation. *ACM Trans. Softw. Eng. Methodol.* 21, 1, Article 6 (dec 2011), 27 pages. https://doi.org/10.1145/2063239.2063245

[18] Nick Feamster and Hari Balakrishnan. 2005. Detecting BGP Configuration Faults with Static Analysis. In *in Proc. Networked Systems Design and Implementation*. 43–56.

[19] Gabriel Ferreira, Momin Malik, Christian Kästner, Jürgen Pfeffer, and Sven Apel. 2016. Do #Ifdefs Influence the Occurrence of Vulnerabilities? An Empirical Study of the Linux Kernel. In *Proceedings of the 20th International Systems and Software Product Line Conference* (Beijing, China) *(SPLC '16)*. ACM, New York, NY, USA, 65–73. https://doi.org/10.1145/2934466.2934467

[20] Brady J. Garvin and Myra B. Cohen. 2011. Feature Interaction Faults Revisited: An Exploratory Study. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*. 90–99. https://doi.org/10.1109/ISSRE.2011.25

[21] Paul Gazzillo. 2020. Inferring and Securing Software Configurations Using Automated Reasoning. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1517–1520. https://doi.org/10.1145/3368089.3417041

[22] Paul Gazzillo and Shiyi Wei. 2019. Conditional Compilation is Dead, Long Live Conditional Compilation!. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results* (Montreal, Quebec, Canada) *(ICSE-NIER '19)*. IEEE Press, 105–108. https://doi.org/10.1109/ICSE-NIER.2019.00035

[23] Alexander Grebhahn, Norbert Siegmund, and Sven Apel. 2019. Predicting Performance of Software Configurations: There is no Silver Bullet. arXiv:1911.12643

[24] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control Parameters Considered Harmful: Detecting Range Specification Bugs in Drone Configuration Modules via Learning-Guided Search. In *Proceedings of the International Conference on Software Engineering*. ACM. https://doi.org/10.1145/3510003.3510084

[25] Karam Ignaim and João M. Fernandes. 2019. An Industrial Case Study for Adopting Software Product Lines in Automotive Industry: An Evolution-Based Approach for Software Product Lines (EVOA-SPL). In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B* (Paris, France) *(SPLC '19)*. Association for Computing Machinery, New York, NY, USA, 183–190. https://doi.org/10.1145/3307630.3342409

[26] Muhammad Adil Inam, Wajih Ul Hassan, Ali Ahad, Adam Bates, Rashid Tahir, Tianyin Xu, and Fareed Zaffar. 2022. Forensic Analysis of Configuration-based Attacks. In *Proceedings of the 29th Network and Distributed System Security Symposium (NDSS'22)*.

[27] Dongpu Jin, Myra B. Cohen, Xiao Qu, and Brian Robinson. 2014. PrefFinder: getting the right preference in configurable software systems. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (Vasteras, Sweden) *(ASE '14)*. Association for Computing Machinery, New York, NY, USA, 151–162. https://doi.org/10.1145/2642937.2643009

[28] D. Jin, X. Qu, M.B. Cohen, and B. Robinson. 2014. Configurations Everywhere: Implications for Testing and Debugging in Practice. In *Companion Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE Companion 2014)*. Association for Computing Machinery, New York, NY, USA, 215–224. https://doi.org/10.1145/2591062.2591191

[29] JCS Kadupitiya, Geoffrey C Fox, and Vikram Jadhao. 2020. Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles. *The International Journal of High Performance Computing Applications* 34, 3 (2020), 357–374. https://doi.org/10.1177/1094342019899457 arXiv:https://doi.org/10.1177/1094342019899457

[30] Patrick Koch, Oleg Golovidov, Steven Gardner, Brett Wujek, Joshua Griffin, and Yan Xu. 2018. Autotune: A Derivative-Free Optimization Framework for Hyperparameter Tuning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 443–452. https://doi.org/10.1145/3219819.3219837

[31] Emily Kowalczyk, Myra B. Cohen, and Atif M. Memon. 2018. Configurations in Android Testing: They Matter. In *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis* (Montpellier, France) *(A-Mobile 2018)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3243218.3243219

[32] Brian Krebs. 2019. Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years. https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/. Accessed: 2019-06-07.

[33] lore.kernel.org 2022. Linux Kernel Mailing list – kismet search results. https://lore.kernel.org/all/?q=kismet. Accessed: 2022-07-13.

[34] Tabassum Mahmud, Duo Zhang, Om Rameshwar Gatla, and Mai Zheng. 2022. Understanding Configuration Dependencies of File Systems. In

Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems (Virtual Event) *(HotStorage '22)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3538643.3539756

[35] Serghei Mangul, Thiago Mosqueiro, Richard J. Abdill, Dat Duong, Keith Mitchell, Varuni Sarwal, Brian Hill, Jaqueline Brito, Russell Jared Littman, Benjamin Statz, Angela Ka-Mei Lam, Gargi Dayama, Laura Grieneisen, Lana S. Martin, Jonathan Flint, Eleazar Eskin, and Ran Blekhman. 2019. Challenges and recommendations to improve the installability and archival stability of omics computational tools. *PLOS Biology* 17, 6 (06 2019), 1–16. https://doi.org/10.1371/journal.pbio.3000333

[36] Niloofar Mansoor, Jonathan A. Saddler, Bruno Silva, Hamid Bagheri, Myra B. Cohen, and Shane Farritor. 2018. Modeling and Testing a Family of Surgical Robots: An Experience Report *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 785–790. https://doi.org/10.1145/3236024.3275534

[37] Sampada Marathe and S. Shyam Sundar. 2011. What Drives Customization? Control or Identity?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 781–790. https://doi.org/10.1145/1978942.1979056

[38] Flávio Medeiros, Iran Rodrigues, Márcio Ribeiro, Leopoldo Teixeira, and Rohit Gheyi. 2015. An Empirical Study on Configuration-Related Issues: Investigating Undeclared and Unused Identifiers. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (Pittsburgh, PA, USA) *(GPCE 2015)*. Association for Computing Machinery, New York, NY, USA, 35–44. https://doi.org/10.1145/2814204.2814206

[39] Jens Meinicke, Chu-Pan Wong, Christian Kästner, Thomas Thüm, and Gunter Saake. 2016. On Essential Configuration Complexity: Measuring Interactions in Highly-Configurable Systems. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (Singapore, Singapore) *(ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 483–494. https://doi.org/10.1145/2970276.2970322

[40] Mainack Mondal, Günce Su Yilmaz, Noah Hirsch, Mohammad Taha Khan, Michael Tang, Christopher Tran, Chris Kanich, Blase Ur, and Elena Zheleva. 2019. Moving Beyond Set-It-And-Forget-It Privacy Settings on Social Media. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 991–1008. https://doi.org/10.1145/3319535.3354202

[41] Austin Mordahl, Jeho Oh, Ugur Koc, Shiyi Wei, and Paul Gazzillo. 2019. An Empirical Study of Real-World Variability Bugs Detected by Variability-Oblivious Tools. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Tallinn, Estonia) *(ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 50–61. https://doi.org/10.1145/3338906.3338967

[42] Vivek Nair, Tim Menzies, Norbert Siegmund, and Sven Apel. 2017. Using Bad Learners to Find Good Configurations. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 257–267. https://doi.org/10.1145/3106237.3106238

[43] Jeho Oh, Don S. Batory, Marijn J. H. Heule, Margaret Myers, and Paul Gazzillo. 2019. Uniform Sampling from Kconfig Feature Models.

[44] Jeho Oh, Necip Fazıl Yıldıran, Julian Braha, and Paul Gazzillo. 2021. Finding Broken Linux Configuration Specifications by Statically Analyzing the Kconfig Language. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 893–905. https://doi.org/10.1145/3468264.3468578

[45] OWASP. 2022. A05:2021 – Security Misconfiguration. https://owasp.org/Top10/A05_2021-Security_Misconfiguration/. Accessed: 2022-09-04.

[46] OWASP. 2022. Top 10 - 2021: The Ten Most Critical Web Application Security Risks. https://owasp.org/Top10/. "Accessed: 2022-06-29".

[47] Stijn Pletinckx, Kevin Borgolte, and Tobias Fiebig. 2021. Out of Sight, Out of Mind: Detecting Orphaned Web Pages at Internet-Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 21–35. https://doi.org/10.1145/3460120.3485367

[48] Xiao Qu, Myra B. Cohen, and Gregg Rothermel. 2008. Configuration-aware Regression Testing: An Empirical Study of Sampling and Prioritization. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis* (Seattle, WA, USA) *(ISSTA '08)*. Association for Computing Machinery, New York, NY, USA, 75–86. https://doi.org/10.1145/1390630.1390641

[49] Ariel Rabkin and Randy Katz. 2011. Precomputing possible configuration error diagnoses. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 193–202. https://doi.org/10.1109/ASE.2011.6100053

[50] Brian Robinson, Mithun Acharya, and Xiao Qu. 2012. Configuration Selection Using Code Change Impact Analysis for Regression Testing. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*. IEEE Computer Society, 129–138. https://doi.org/10.1109/ICSM.2012.6405263

[51] Mohammed Sayagh, Noureddine Kerzazi, Fabio Petrillo, Khalil Bennani, and Bram Adams. 2020. What should your run-time configuration framework do to help developers? *Empirical Software Engineering* (2020). https://doi.org/10.1007/s10664-019-09790-x

[52] Norbert Siegmund, Alexander Grebhahn, Christian Kästner, and Sven Apel. 2015. Performance-Influence Models for Highly Configurable Systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 284–294. https://doi.org/10.1145/2786805.2786845

[53] Oleksandr Tomashchuk, Dimitri Van Landuyt, and Wouter Joosen. 2021. *The Architectural Divergence Problem in Security and Privacy of EHealth IoT Product Lines.* Association for Computing Machinery, New York, NY, USA, 114–119. https://doi.org/10.1145/3461001.3473061

[54] Dimitri van Heesch. 2022. Doxygen. https://doxygen.nl/. Accessed: 2022-09-04.

[55] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. 2021. White-Box Analysis over Machine Learning: Modeling Performance of Configurable Systems. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1072–1084. https://doi.org/10.1109/ICSE43902.2021.00100

[56] Helen J. Wang, John C. Platt, Yu Chen, Ruyun Zhang, and Yi min Wang. 2004. Automatic Misconfiguration Troubleshooting with Peerpressure. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6* (San Francisco, CA) *(OSDI'04)*. USENIX Association, USA, 17.

[57] Zack Whittaker. 2018. GitHub says bug exposed some plaintext passwords. https://www.zdnet.com/article/github-says-bug-exposed-account-passwords/. Accessed: 2019-06-07.

[58] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. 2015. Hey, You Have given Me Too Many Knobs!: Understanding and Dealing with over-Designed Configuration in System Software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 307–319. https://doi.org/10.1145/2786805.2786852

[59] Tianyin Xu, Vineet Pandey, and Scott Klemmer. 2016. An HCI View of Configuration Problems. arXiv.

[60] C. Yilmaz, M. B. Cohen, and A. Porter. 2006. Covering arrays for efficient fault characterization in complex configuration spaces. *IEEE Transactions on Software Engineering* 31, 1 (2006), 20–34. https://doi.org/10.1109/TSE.2006.8

[61] Jialu Zhang, Ruzica Piskac, Ennan Zhai, and Tianyin Xu. 2021. Static Detection of Silent Misconfigurations with Deep Interaction Analysis. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 140, 30 pages. https://doi.org/10.1145/3485517

[62] Sai Zhang and Michael D. Ernst. 2014. Which Configuration Option Should I Change?. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 152–163. https://doi.org/10.1145/2568225.2568251

[63] Yuanliang Zhang, Haochen He, Owolabi Legunsen, Shanshan Li, Wei Dong, and Tianyin Xu. 2021. An Evolutionary Study of Configuration Design and Implementation in Cloud Systems. In *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings* (Virtual Event, Spain) *(ICSE ’21)*. IEEE Press, 175–176. https://doi.org/10.1109/ICSE-Companion52605.2021.00075