Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems

An Industrial Experience Report

Heng Li
Queen's University, Canada
hengli@cs.queensu.ca

Tse-Hsun (Peter) Chen
Concordia University, Canada
peterc@encs.concordia.ca

Ahmed E. Hassan
Queen's University, Canada
ahmed@cs.queensu.ca

Mohamed Nasser
BlackBerry, Canada

Parminder Flora
BlackBerry, Canada

ABSTRACT

In current DevOps practice, developers are responsible for the operation and maintenance of software systems. However, the human costs for the operation and maintenance grow fast along with the increasing functionality and complexity of software systems. Autonomic computing aims to reduce or eliminate such human intervention. However, there are many existing large systems that did not consider autonomic computing capabilities in their design. Adding autonomic computing capabilities to these existing systems is particularly challenging, because of 1) the significant amount of efforts that are required for investigating and refactoring the existing code base, 2) the risk of adding additional complexity, and 3) the difficulties for allocating resources while developers are busy adding core features to the system. In this paper, we share our industrial experience of re-engineering autonomic computing capabilities to an existing large-scale software system. Our autonomic computing capabilities effectively reduce human intervention on performance configuration tuning and significantly improve system performance. In particular, we discuss the challenges that we encountered and the lessons that we learned during this re-engineering process. For example, in order to minimize the change impact to the original system, we use a variety of approaches (e.g., aspect-oriented programming) to separate the concerns of autonomic computing from the original behaviour of the system. We also share how we tested such autonomic computing capabilities under different conditions, which has never been discussed in prior work. As there are numerous large-scale software systems that still require expensive human intervention, we believe our experience provides valuable insights to software practitioners who wish to add autonomic computing capabilities to these existing large-scale software systems.

CCS CONCEPTS

• **Software and its engineering** → *Software development process management*; • **Computer systems organization** → *Self-organizing autonomic computing*;

KEYWORDS

Autonomic computing, software re-engineering, performance engineering, software testing

ACM Reference Format:
Heng Li, Tse-Hsun (Peter) Chen, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2018. Adopting Autonomic Computing Capabilities in Existing Large-Scale Systems: An Industrial Experience Report. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183544

1 INTRODUCTION

Due to advances in software technologies, software systems are becoming increasingly complex and powerful. For example, VISA's card processing systems and BlackBerry's enterprise systems are able to handle millions of concurrent users. To achieve their optimal performance, these complex software systems require highly skilled developers and testers for operation and maintenance of the software [8]. Such manual efforts are one of the main costs of operating and maintaining modern large-scale software systems [15]. In particular, manual interventions are constantly required for such systems given the constantly changing environment (e.g., network bandwidth) and workloads, thereby driving up the overall operational costs [8].

Researchers have proposed the concept of autonomic computing to reduce the necessity for human monitoring and intervention in large-scale systems. Autonomic computing was first introduced by IBM. Autonomic computing defines a computing framework that is capable of managing itself, and rapidly and continuously adapting to changing operating environments [8]. For example, the IBM DB2 system[1] contains many autonomic computing features, among which is the feature of self-tuning memory[2]. The self-tuning memory feature automatically adjusts the values of several memory configuration parameters in accordance with workload changes;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE-SEIP '18, May 27-June 3 2018, Gothenburg, Sweden

© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.

ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183544

[1] https://www.ibm.com/analytics/us/en/db2

[2] https://www.ibm.com/support/knowledgecenter/en/SSEPGG_9.7.0/com.ibm.db2.luw.admin.dbobj.doc/doc/r0051462.html

thus, simplifying memory configuration tasks and optimizing performance.

In recent years, many real-time data analysis frameworks (e.g., Apache Spark and Apache Storm) have become easily accessible, significantly reducing the challenge of designing and implementing new autonomic computing systems. Developers can easily leverage such frameworks to implement autonomic computing systems that require the analysis of real-time system data. However, many existing large-scale systems that were not designed with autonomic computing capabilities in mind still suffer from the growing and expensive costs of human involvement. Adding autonomic computing capabilities to these existing large-scale systems is a particularly challenging software engineering task. First, adopting autonomic computing capabilities to existing systems requires tremendous investigation, code changes, and refactoring of the existing code, since the systems were designed and built without an autonomic computing mindset. Second, companies are not likely to risk fully redesigning the system or performing significant code changes, since such changes may lead to unexpected consequences (e.g., introduce new functional or non-functional bugs). Finally, in modern fast-paced agile software development, companies may have difficulties in allocating and scheduling additional resources for implementing autonomic computing capabilities, as developers are constantly working on delivering customers' needs in parallel.

In this paper, we share our industrial experience of adopting autonomic computing capabilities to an existing non-autonomic large-scale software system. Our autonomic computing capabilities aim to autonomically tune the system configurations to optimize system performance and reduce human intervention. We encountered many engineering and design challenges when adding autonomic computing capabilities to the system. We needed to understand the system runtime behaviour from scratch, minimize code changes to the original system, and thoroughly test the autonomic computing capabilities. In the end, we conquered the challenges and successfully integrated autonomic computing capabilities to the system with minimal code changes (i.e., only a few hundred lines of code) to the original system. We believe that our experience in transforming existing systems to become autonomic is invaluable and can help software practitioners and researchers who want to adopt autonomic computing to existing systems.

The main contributions of this paper are:

- We offer an industrial experience report of the software engineering challenges that are encountered when adopting autonomic computing capabilities to existing large-scale non-autonomic commercial systems.
- We provide an overview of our approach on how to minimize code changes and performance overhead to existing systems when adopting autonomic computing capabilities.
- We are the first to discuss our experience in testing the autonomic computing capabilities that were integrated with existing large-scale systems.

Paper organization. The rest of the paper is organized as follows. Section 2 surveys related work on software engineering of autonomic computing. Section 3 describes our subject system and explains the background for adding autonomic computing capabilities to the system. Section 4 discusses the challenges that we encountered in the project and our solutions to these challenges. Section 5 describes our implementation of the autonomic computing capabilities. Section 6 discusses how we tested our autonomic computing capabilities. Finally, Section 7 concludes the paper.

2 RELATED WORK

In this section, we discuss related work to our study. We divide the related work into two categories: 1) background of autonomic computing and 2) adopting autonomic computing capabilities to existing systems.

2.1 Autonomic computing

Many existing studies focus on the methodologies for designing a software system with autonomic computing capabilities. However, our paper shares our experience of adding autonomic computing capabilities to an existing large-scale software system that did not consider autonomic computing in its design. IBM is one of the biggest initializers for autonomic computing. They first defined the term *autonomic computing* as computing systems that can manage themselves given high-level objectives from developers [12]. IBM later proposed an architectural approach for creating autonomic components and composing these autonomic components into autonomic systems [8, 20]. They also validated many of their ideas in two prototype autonomic computing systems. Dobson et al. [5] discussed the progress and the state-of-the-art approaches of autonomic computing in the first 10 years since it was proposed. They argue that the original vision of autonomic computing remains unfulfilled, and that researchers must develop a comprehensive engineering approach (i.e., autonomic systems engineering) to create effective solutions for the next-generation of software systems.

2.2 Adding autonomic computing capabilities into existing systems

There are a few studies on adding autonomic computing capabilities to existing systems. Mulcahy et al. [15] described a commercial software project in which they added autonomic computing capabilities into a legacy order-placing system. The resulting system has the ability of self-monitoring, self-configuration and self-healing. Their autonomic computing capabilities significantly reduced the human efforts that were involved in placing orders, which also reduced the costs associated with human error in system operation activities. Mulcahy et al. [16] also proposed an approach to add autonomic computing capabilities into a legacy order-fulfillment system. They used the service-oriented architecture to add a new autonomic component to the original system. Their approach reduced the cost of human labor for screening orders and also reduced errors in the screening process. Amoui [1] proposed an approach to add autonomic computing capabilities to an existing software in an evolution process. The approach tried to implement self-adaptive requirements by using a co-evolution model in which the autonomic requirements are added to the original software system by a set of transformations.

Our work is unique and different from the above-mentioned studies. While these existing studies focus on the architecture design and the algorithms for parameter tuning, our work focuses on the software engineering challenges (e.g., testing) that one might face when adding autonomic computing capabilities to existing large-scale non-autonomic software systems. In particular, we focus on the challenges that we encountered during the design (e.g., how can we bring minimal impact to the original system), implementation (e.g., how to avoid introducing too many code changes), and testing (e.g., testing of robustness) of the added autonomic computing capabilities.

3 BACKGROUND

In this section, we briefly discuss our subject system and the background story for adding automatic computing capabilities to the system.

Our subject system is a large-scale enterprise system. Due to competitive and critical nature of the system, we cannot give the exact details of the subject system. However, the system is very large in size (millions of lines of code), under active development, and maintained by a large number of software developers. Thousands of organizations around the world depend on the subject system for their mission-critical operations.

Similar to other non-autonomic large-scale systems, our subject system requires expensive manual efforts for operation and maintenance. Due to the increasing popularity of DevOps (development and operations), such manual efforts are usually handled by developers who have an in-depth knowledge about the system. However, although developers are very familiar with the system, in our experience, they still have difficulties finding the optimal values for performance-related configuration parameters. There are several reasons. First, in existing systems that were not built with autonomic computing capabilities in mind, the values of the configuration parameters are usually read offline (e.g., from static configuration files), which increases the challenge of real-time configuration tuning. Second, performance-related configuration parameters are sensitive to the working environment (e.g., network bandwidth or latency) and the workload (e.g., the amount of user requests, which is constantly changing and shows high variations). Hence, finding the configuration values that can lead to the optimal system performance is extremely expensive and requires developers' constant attention.

In order to reduce the needed manual costs to constantly tune the parameters, our industrial partner wanted to add autonomic computing capabilities to the enterprise system so that the system can autonomically tune its parameters without human intervention. However, the subject system was designed without the autonomic computing mindset. Significant investigation, code changes, and refactoring are needed to add such autonomic computing capabilities. Therefore, our industrial partner cooperated with us to explore the feasibility of adding the autonomic computing capabilities to the existing system with minimal code changes, as a research project that is independent from the development and maintenance of the system. As embedded researchers, we had limited support from the development teams, since developers are actively working on delivering customers' needs in a fast-paced agile software development setting.

In order to minimize the risk of changing too much code, we proposed a set of solutions that successfully added autonomic computing capabilities to the system with minimal code changes (i.e., only a few hundred lines of code) to the original system. As a result of the project, the system gets the abilities of self-monitoring its performance measures and workloads, self-optimizing its parameter values, and self-configuring the optimal parameter values. Moreover, we designed our autonomic computing capabilities in a way that would not bring too much side effect to the normal behaviour of the system, which significantly improves the adoption of our research results.

4 CHALLENGES AND LESSONS LEARNED

Figure 1 shows our overall engineering process of adding autonomic computing capabilities to our subject system which was designed without an autonomic computing mindset. Overall, our engineering process contains four parts: understanding system behaviour, adding autonomic computing capabilities, adding the capability to monitor and control the autonomic computing capabilities, and testing the autonomic computing capabilities.

In this section, we provide detailed discussions on the challenges that we encountered when adding autonomic computing capabilities, our solutions to these challenges, and the lessons that we learned. We also describe the details of our engineering process in our discussions. Our methodology for testing the autonomic computing capabilities is discussed in detail in Section 6.

4.1 Understanding system runtime behaviour from scratch

Challenge. An autonomic system should be able to adapt itself according to the changing environment and workloads. In order to add autonomic computing capabilities to our subject system, we first need to understand how the system behaves under different environment and workloads, and how the configuration parameters impact its behaviour. However, compared to autonomic systems, for which the system behaviour is carefully examined to support autonomic computing capabilities, the runtime behaviour of a large-scale software system without autonomic considerations (e.g., our subject system) is usually never well understood. In such systems without autonomic considerations, even experienced developers may not clearly understand the runtime performance impact of the configurable parameters. Namely, developers can only change the values of these configuration parameters offline (e.g., read once at startup), without the ability to know the impact of changing the parameters under evolving environment and workloads. Therefore, our first challenge is to understand the system runtime behaviour, and in particular, how the configuration parameters impact the runtime performance of the system.

Solution. In order to understand how the configuration parameters impact the runtime performance of the system, we first discussed with senior developers about which parameters might impact the system performance, and how developers typically set the parameter values during their own testing. We then ran testing experiments with different parameter configurations to understand runtime system behaviour and how different configurations actively impact the performance of the system. The senior developers' advice and our exploratory experiments provided us a rough idea about the configuration parameters' impact on the system performance.

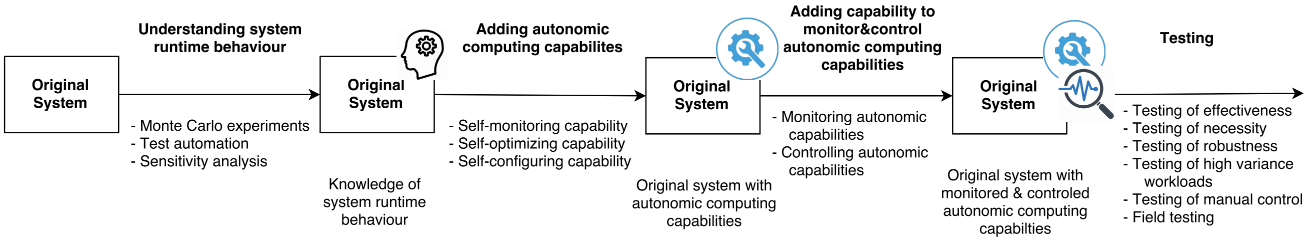
Original System
Understanding system runtime behaviour
- Monte Carlo experiments
- Test automation
- Sensitivity analysis
Original System
Knowledge of system runtime behaviour
Adding autonomic computing capabilites
- Self-monitoring capability
- Self-optimizing capability
- Self-configuring capability
Original System
Original system with autonomic computing capabilities
Adding capability to monitor&control autonomic computing capabilities
- Monitoring autonomic capabilities
- Controlling autonomic capabilities
Original System
Original system with monitored & controled autonomic computing capabilties
Testing
- Testing of effectiveness
- Testing of necessity
- Testing of robustness
- Testing of high variance workloads
- Testing of manual control
- Field testing

Figure 1: Our overall engineering process of adding autonomic computing capabilities to an existing large-scale system.

However, before adding autonomic computing capabilities to the system, we need a better quantitative understanding of the relationship between the parameters and the system performance. Therefore, we want to collect enough test data that describes the setting of parameter values and the corresponding performance measures. One way to fully understand the effect of the configuration parameters on system performance is to exhaustively run different combinations of the parameter values. However, such an approach is unfeasible. Supposing that we have five parameters and we want to test ten values for each parameter, we would need to run the system 100,000 (i.e., 10^5) times. Each test takes hours to finish so we would need tens of years to collect the data.

As an alternative, we use a Monte Carlo method [14] to generate repeated random samples of the parameter values. The advantage of the Monte Carlo method is that the number of random samples does not increase with the number of independent parameters. Specifically, in each sample, the Monte Carlo method randomly chooses the values for the parameters within given ranges. In total, we generate 60 randomly sampled sets of parameter values, and we measure the performance of the system under these parameter settings. Running tests using these 60 randomly sampled sets of parameter values took only a few days. Even though we only sampled 60 sets of parameter values in total, we actually evaluated 60 random values for each parameter. These experiments were fully automated and did not need any additional human resources. For each test, we recorded the values of the configuration parameters and the resulting performance measures.

After we collected the performance data for the 60 tests (i.e., 60 parameter settings), we performed a sensitivity analysis to understand the configuration parameters' impact on the system performance. Specifically, we built Multivariate Adaptive Regression Splines (MARS) [6] models to understand the relationship between the parameters and the performance measures. The MARS models can automatically model the nonlinear relationship between the response variable (e.g., our performance measures) and the explanatory variables (e.g., our parameters), as well as the interactions between the explanatory variables. We found that only some of the parameters have a statistically significant impact on the performance of the system. We removed those statistically *insignificant* parameters from further consideration in our work. The MARS models also describe, for each performance measure, which parameters have a positive (or negative) impact.

Our approach significantly reduces the search space (i.e., by reducing the number of configuration parameters) and helps find the optimal parameter values faster. The knowledge that we get from the statistical analysis (e.g., which parameters have a positive or negative impact on performance measures) is also crucial for us to design our optimization engine to autonomically tune the parameter values for optimal performance.

Lessons learned. **Leveraging statistical techniques to understand system behaviour.** In this work, we use a Monte Carlo method to design the experiments to collect the data that is later used to understand the relationship between the configuration parameters and the system performance. We also use the MARS models to model the relationship between the parameters and the performance of the system. However, our goal is not to fully understand the relationship between the configuration parameters and the performance measures, and our models are not targeted to predict the performance measures under certain parameter values. Even with test automation, testing resources (e.g., hardware) are still a huge bottleneck for running a large number of tests to fully understand the relationship. Instead, our goal is to find the configuration parameters that have the most significant impact on the performance measures (i.e., performance critical parameters), and how changing these configuration parameters impact the performance measures (i.e., positive or negative impact).

Even though there might be other configuration parameters that have less significant impact on the performance measures, we only consider these performance critical parameters for performance tuning. We do so for several reasons: 1) tuning these parameters can more effectively improve the performance of the system; 2) tuning a smaller number of parameters improves the speed of searching for the optimal configuration; and 3) we introduce fewer code changes to the system.

Using test automation to minimize experimental overhead. In this work, we need to repeatedly test the performance of the system under different parameter settings to collect the data for analyzing the relationship between the configuration parameters and the system performance. As the system does not support updating parameters dynamically, for each parameter setting, we need to stop the previous test, cleanup the system to remove the impact of the previous test (e.g., restoring the database), change the parameter setting, and restart the system, etc. Manually repeating these steps is cumbersome and has the risk of introducing human errors (e.g., forgetting to cleanup the system before restarting a new test). Therefore, we implemented test automation tools to automate the entire process.

Nevertheless, even after we use the Monte Carlo method to reduce the number of tests, it is still challenging to schedule testing resources for these tests, as these testing resources are usually demanded by other higher-priority tests (e.g., continuous integration tests). Therefore, test automation also allowed us to effectively leverage the testing resources by running tests in machine-spare time (e.g., during holidays or weekends).

We leverage both domain experts' experience and statistical techniques, e.g., Monte Carlo experiments and sensitivity analysis, to understand the relationship between the configuration parameters and system performance. Test automation is also critical for minimizing manual effort during the experiments and leveraging testing resources during machine-spare time.

4.2 Adding autonomic computing capabilities to a non-autonomic system with minimal code changes

Challenge. Non-autonomic systems often read configurations statically (e.g., at startup and usually from files), and developers cannot easily change the values of the configuration parameters when the system is running. In addition, many existing systems do not continuously monitor their production behaviour for information such as performance (e.g., CPU and memory) and workloads (e.g., user requests).

Hence, in order to add autonomic computing capabilities to existing non-autonomic systems, we need to add three main capabilities to the systems: a self-monitoring capability that monitors the performance measures, a self-optimizing capability that searches for optimal parameters based on the monitored performance measures, and a self-configuring capability that dynamically updates the values of the configuration parameters. However, adding these autonomic computing capabilities to existing systems requires changing the software requirements, which may cause severe risks [1]. Even if we decided to change the requirements, we would need to perform significant code changes and affect the overall design of the system, which may cause additional bugs, development overheads, and maintenance problems.

Solution. We propose an autonomic computing solution that introduces minimal changes to the system and comes with negligible system overhead. We separate the autonomic computing capabilities from the original system. The separated autonomic computing capabilities are able to: 1) collect and compute performance measures from the system (self-monitoring); 2) search for the optimal configuration parameters based on the performance measures (self-optimizing); and 3) updating the system configuration parameters (self-configuring). We found that separating the autonomic computing capabilities significantly reduced required code changes to the original system and minimized the performance overhead. Below, we discuss how we refactored the system to become autonomic.

Self-monitoring. To minimize performance impact and the required code changes when monitoring the system, we leverage existing system logs (e.g., web server access logs) and collect performance counters (e.g., sampled CPU and memory usage). The readily-available logs and the performance counters allow us to monitor many system-specific performance measures (e.g., throughput) and other general performance measures (e.g., CPU usage), respectively. By taking advantage of the existing logs, we do not need to make code changes to the original system for collecting system-specific performance measures. On the other hand, the performance overhead of collecting performance counters is very low (under 1% CPU overhead) and does not require any code changes (e.g., can be handled by the underlying OS).

Self-optimizing. To minimize code changes and the analysis overhead on the original system, we separated the optimization component from the original system. We analyze collected performance counters and logs on separate machines so that we would not affect the original system.

Self-configuring. To minimize the code changes to the system, we use an aspect-oriented programming approach based on reflection [18] to dynamically configure the values of the parameters in the system. Using an aspect-oriented programming approach separates the self-configuring concern and significantly simplifies dynamic parameter tuning. Aspect-oriented programming is a general programming paradigm that aims to increase modularity by separating cross-cutting concerns [13]. Aspect-oriented programming is particularly useful for adding behaviour that is not central to the business logic of a program (e.g., the autonomic computing capabilities) without bringing much impact to the core business logic of the system. In the end, our approach only required changing a few hundreds lines of code to the original system.

Lessons learned. **Avoiding code merging conflicts in parallel development.** In our project, we are developing our autonomic computing capabilities while developers are focusing on developing the next release of the original system. Therefore, many conflicts may occur when we are merging the code. However, we find that by adopting an aspect-oriented programming approach, we minimize the dependencies between the autonomic computing capabilities and the original system. Thus, the development of the autonomic computing capabilities and the active development of the original system can be done in parallel, while minimizing potential code merging conflicts.

Maintaining the performance profile of the original system. Software systems usually need to maintain a certain level of service-level agreement, such as the number of machines required for handing X number of users and the hardware requirements. By separating the autonomic computing capabilities, we ensure that we would not negatively impact the performance of the original system and change the known performance profiles and service-level agreements.

Leveraging existing logs to monitor system runtime behaviour. System logs contain rich information about the runtime behaviour of a system [2, 22, 23]. These logs are usually leveraged by developers and testers to detect abnormal conditions or debug system failures [3, 11]. In this work, we calculate performance measures based on the readily available logs. For example, we measure the occurrence of some specific events (e.g., requests) by counting the number of corresponding log lines. We then calculate some goal measures based on the occurrences of the specific events. Many existing systems already use tools such as syslog [21] to collect their runtime logs to external machines. In such cases, analyzing the logs would not cause any additional overhead to the deployed system.

An alternative approach for monitoring system runtime behaviour would be applying the aspect-oriented programming paradigm to instrument probes in the source code of the original system. However, the needed information that we derive from logs comes from several different computing components, which are written in different programming languages and may be developed by other organizations. Therefore, it is very challenging to use aspect-oriented programming approaches to instrument probes in these components.

We minimize code changes to the original system by separating the autonomic computing concerns from the original business logic, as well as leveraging readily available system logs to monitor system runtime behaviour.

4.3 Working with domain experts with limited knowledge of autonomic computing

Challenge. Adding autonomic computing capabilities to an existing large-scale software system involves all aspects of a software system's life cycle, including requirement collection, design, implementation, testing, and operation. In order to accomplish the project with limited resources and tight schedules, we need to drive the domain experts (e.g., developers, testers, and operators) to make their contributions to the project through different ways. For example, we need to drive testers to set up the system's running environment and fix problems in the environment; we need developers' buy-in to change the code of the system, even though we minimized the code changes to the original system, to support the autonomic computing capabilities; and we need operators' help to deploy and test the autonomic computing capabilities in a real working environment.

However, domain experts are usually busy with their day-to-day responsibilities. For example, developers may follow fast-paced agile software development cycles to meet customers' growing functional needs. Hence, domain experts may not be able to afford much extra time to contribute to this research project. More importantly, since the system was initially designed without autonomic computing mindsets, domain experts only have limited knowledge about the concept of autonomic computing. Thus, it is challenging to drive the domain experts to make contributions to this research project. Even worse, these domain experts often come from different business groups, which poses additional challenges when dealing with cross-team and cross-organization cooperation issues.

Solution. In order to build effective autonomic computing capabilities for the system, we worked closely with the domain experts of the system. These domain experts have years of experience in developing, testing, and operating the system. As an effective way of communication, we had regular meetings with these domain experts. In these regular meetings, we presented our approaches and progress, as well as the challenges that we encountered. In this way, the domain experts were continuously kept on the same page as us. Therefore, they were able to effectively share their expertise and provide us with valuable suggestions. For example, we discussed the relationship between the parameters and the system performance in many meetings, and the domain experts provided very helpful suggestions for us to determine the initial set of parameters that may impact the system performance. We also continuously demonstrated the value of the autonomic computing capabilities, to motivate the domain experts to make their contributions to the research project from different perspectives (e.g., changing the source code or testing the autonomic computing capabilities in a real production environment).

The domain experts usually have limited knowledge of autonomic computing. In order to drive them to make contributions to the research project, we used evidence to illustrate the effectiveness of the autonomic computing capabilities. The first thing that we had to do is to show the value and feasibility of adding autonomic computing capabilities to the system. Hence, before directly asking domain experts to contribute to the project, we conducted initial experiments to demonstrate that the autonomic computing capabilities can significantly improve system performance and reduce human intervention (i.e., proof of concept). Initially, we demonstrated the concept of autonomic computing without changing the source code of the system, i.e., without the ability to dynamically update the parameter values. Specifically, when our self-optimizing component determines new values of configuration parameters, we used a script to automatically stop the system, make changes to the parameter values, and restart the system. Although the search process would take a very long period of time (each search step requires rerunning the test), through this initial experiment, we demonstrated that the autonomic computing capabilities can achieve much better performance than using a default parameter setting with some manual tuning.

With the proof of concept, we worked with the developers to change the system code to first support dynamic configuration of several important configuration parameters (i.e., ones with a high influence on system performance). Then, we integrated developers' code changes to our autonomic computing capabilities so that we can dynamically change these configuration parameters. We further conducted experiments to illustrate that, with the dynamic configuration, our autonomic computing capabilities can efficiently improve the system performance and reduce human intervention. Finally, we convinced the developers and worked with them to provide dynamic configuration for the rest of the parameters. In short, to minimize development risks and engage other domain experts, we find that it is important to work incrementally to uncover the benefits of adding autonomic computing capabilities.

Lessons learned. **Minimizing the risks of adding autonomic computing capabilities.** While working with developers and testers across the company, we found one major obstacle that discouraged domain experts from contributing to the project: "Domain experts are particularly concerned with the additional risks that the autonomic computing capabilities may bring to the original system, since their top priority is to ensure the correct functionality and scalable behaviour of the current system". Therefore, to minimize the risks, we designed our autonomic computing capabilities as a separate component to the original system. Therefore, the autonomic computing capabilities can be disabled whenever a problem occurs, without affecting the normal system behaviour. Besides, we always recover the system back to the default state in case of failures (e.g., crashes) of the autonomic computing capabilities, so that the failures of the autonomic computing capabilities will not interrupt the normal execution of the original system. We also recover the system to the default state after completing a workload, so that the autonomic computing capabilities does not impact the execution of future workloads. Such design significantly attracted developers' attention on the research project and motivated their participation.

Providing prompt feedback to the domain experts. Providing prompt feedback to the domain experts can keep their enthusiasm in the research project, thus they are more likely to provide further contribution on the project. For example, once developers made code changes to support dynamic configuration for the first set of impactful parameters, we quickly integrated such changes into our autonomic computing capabilities. We then provided prompt feedback to the developers and showed them the benefits of the code changes on the autonomic computing capabilities. Developers were encouraged by the results and were willing to support the dynamic configuration of the rest of the parameters. We think that providing prompt feedback is another effective way of communication and it can keep the domain experts more passionate about the autonomic computing capabilities.

We use effective communication and experimental evidence to motivate the domain experts to make contributions from different perspectives (e.g., development, testing, and operation). We also minimize the risk of adding autonomic computing capabilities, which further motivates domain experts' contributions.

4.4 Increasing the adoption of the autonomic computing capabilities

Challenge. We find that, due to an incomplete understanding of the added autonomic computing capabilities, developers may not always want to adopt the autonomic computing capabilities *in production*. More specifically, some developers are concerned that our autonomic computing capabilities were not designed as part of the original system, which may cause unexpected consequences to the deployed system, especially given the wide number of deployment sites. Although we minimized the risks of adopting autonomic capabilities by implementing them as a separate component to the original system (Section 4.2), developers may still have concerns regarding the additional capabilities.

Solution. To solve the challenge, we integrated more manual control and debugging information in our autonomic computing version. The autonomic computing version also allows developers to enable/disable the autonomic computing capabilities in real-time, providing full control to the developers. We also made our autonomic computing capabilities easy to use for people who do not understand autonomic computing. For example, we designed a simple UI so that developers can simply click a button to enable or disable the autonomic computing capabilities (e.g., in case unexpected errors occur). Enabling or disabling the autonomic computing capabilities does not interrupt the normal execution of the system. Developers can also observe the dynamic parameter settings and optimal parameter searching, as well as the performance measures of the system through the UI. We find that by providing full control to the developers, they can slowly start trusting the autonomic computing capabilities (i.e., it is not completely autonomic in a way that developers cannot monitor or control), which significantly helps in the adoption of the capabilities.

Lessons learned. **Providing real-time debugging support to developers.** Most of the existing research in autonomic computing focuses on the architecture design or parameter tuning algorithms [4, 5]. However, we find that it is important to make the effect of such autonomic computing capabilities transparent to developers. Our real-time monitoring features allow developers to easily debug problems in the system (i.e., problems that are related to either the autonomic computing capabilities, or the original system in general) early in the software development process, and hence reduces costs. In addition, through the debugging process, developers slowly gain more confidence in the overall system and are more willing to adopt our autonomic computing capabilities in the production environment.

Stress testing of the autonomic computing capabilities. To provide more confidence to the development teams, we found that it is important to stress test the autonomic computing capabilities. Unlike regular performance tests, where we aim to simulate real-world workloads and observe any abnormal system behaviour [3], we want to ensure that our autonomic computing capabilities would still work expectedly under extreme workloads. We want to report the overload profiles of the autonomic computing capabilities, so we can illustrate that under extreme workloads: 1) whether it is possible for the autonomic computing capabilities to recover in case of failures; 2) whether the autonomic computing capabilities would misbehave (e.g., finding worse parameters); and 3) whether the feature for enabling/disabling the autonomic computing capabilities can still work. By showing a complete overload profile, developers gained significantly more trust in the reliability of the autonomic computing capabilities in a production environment.

In order to increase the adoption of the autonomic computing capabilities, we provide full control to developers on debugging, monitoring, and enabling/disabling the autonomic computing capabilities. We also stress test the autonomic computing capabilities to ensure that the system can still work expectedly under extreme workloads.

5 IMPLEMENTATION

In this section, we briefly discuss the overall structure and the implementation of our autonomic computing capabilities.

Figure 2 illustrates the overall structure of our autonomic computing capabilities. Our autonomic computing capabilities mainly contain four components: the self-monitoring component, the self-optimizing component, the self-configuring component, and the monitoring and controlling component. The original system takes an initial configuration of the parameters and executes a given workload. The self-monitoring component collects performance related log data and calculates corresponding performance measures. The self-optimizing component searches for a better set of parameter values based on the monitored performance measures. The self-configuring component updates the parameter values of the system on-the-fly. These self-monitoring, self-optimization, and

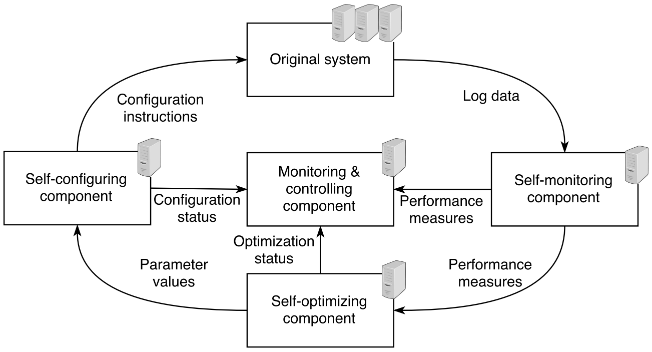
Original system
Configuration instructions
Log data
Self-configuring component
Monitoring & controlling component
Self-monitoring component
Configuration status
Performance measures
Optimization status
Parameter values
Performance measures
Self-optimizing component

Figure 2: The overview structure of our autonomic computing capabilities.

self-configuring processes repeat until an optimal performance is achieved. Finally, the monitoring and controlling component monitors the status of the entire autonomic computing capabilities and supports external control of the autonomic computing capabilities.

The original system. The performance of the original system is determined by the working environment, the workload, and a set of performance-related configuration parameters. As the working environment and the workload are constantly changing, we need to update the values of the configuration parameters to adapt to the changing environment and workload.

The self-monitoring component. The self-monitoring component contains two parts: a data stream producer and a data stream consumer. The producer uses Kafka[3] to collect log data and generate log data stream in real-time. The consumer uses Spark[4] streaming to process the log data stream and calculate performance measures. The self-monitoring component also provides performance measures to other components (i.e., the self-optimizing component and the monitoring and controlling component) in real-time. The performance measures are written into a document-based database (e.g., MongoDB) for analysis and storage purposes. The producer can be located on a separate machine that has reading access to the system logs. The consumer is located on a separate machine.

The self-optimization component. The self-optimization component uses a local search algorithm to search for the optimal configuration point. We use a local search instead of global search algorithms or genetic algorithms because the local search can response faster to the changing environment and workload [7]. The local search algorithm continually evaluates its neighbour points (i.e., configuration points with the closest parameter values) and moves in the direction that improves system performance, and terminates when it reaches a peak (no neighbour point has a better performance). As discussed in Section 4.1, we use Monte Carlo experiments and MARS models to find the performance critical parameters and their impact on the performance of the system. We only consider the performance critical parameters, thus we significantly reduce our search space. Furthermore, we implement a guided search algorithm, based on the known relationship between the parameters and performance measures, to evaluate only the neighbour points that may improve system performance. Our search algorithm ensures that we can reach an optimization point faster (e.g., within minutes) and can react quickly to changes in the workload. The self-optimization component is located on a separate machine from the original system.

The self-configuring component. The self-configuration component updates the parameter values of the system while the system is still running. Many locations (i.e., reference locations) of the system's code use these parameter values in their business logic. Changing all the reference locations to support the dynamic update of the parameter values is cumbersome and risky. Instead, we use an aspect-oriented programming paradigm to separate the concern of dynamically changing the parameter values from the original business logic of the system. Specifically, we define the cross-cutting concerns of dynamically changing parameter values as aspects, and automatically apply these aspects at the code locations where the parameter values are referenced (i.e., join points). We reduce the impact of code changes to the original system by separating the concerns of dynamically updating the parameter values from the original business logic of the system.

The monitoring and controlling component. The monitoring and controlling component provides a web UI for developers to monitor the status of our autonomic computing capabilities in real-time. It monitors the performance measures that are calculated by the self-monitoring component; it monitors the searching process of the self-optimizing component; it also monitors the values of the configured parameters from the self-configuring component. The monitoring and controlling component also provides options for developers to enable or disable the autonomic computing capabilities at any time (i.e., before or while the original system is running), or manually change the values of the configuration parameters while the system is running. In summary, the monitoring and controlling component makes our autonomic computing capabilities transparent to developers.

6 TESTING

As shown in Figure 1, the last step in our engineering process is to test the autonomic computing capabilities. Testing of re-engineered autonomic computing capabilities needs to consider additional perspectives other than traditional functional testing and load testing. However, prior studies [1, 15, 16] never discussed the testing of re-engineered autonomic computing capabilities.

Testing autonomic computing capabilities is a long and complicated software engineering process. We need to test not only the autonomic computing capabilities, but also the original system while the autonomic computing capabilities are enabled. In addition to the testing of the effectiveness and robustness of the autonomic computing capabilities, we also need to test whether the autonomic computing capabilities are needed at all (i.e., whether we can find a universally optimal configuration and apply it in all the system deployment). Finally, we need to test whether the autonomic computing capabilities can be effectively monitored and controlled by developers.

In this section, we briefly discuss our testing strategies for ensuring that the autonomic computing capabilities meet our requirements (e.g., effectiveness and robustness). We also briefly discuss

[3] https://kafka.apache.org
[4] https://spark.apache.org

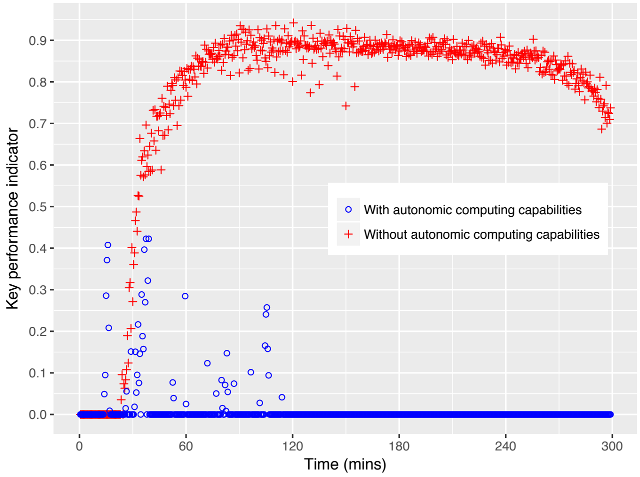

Figure 3: Comparing the performance of the system with and without autonomic computing capabilities. The key performance indicator ranges from 0 to 1, the lower the better.

how the autonomic computing capabilities are adopted in production.

6.1 Testing of effectiveness

In our testing environment, we use a performance testbed that involves multiple machines to verify the effectiveness of our autonomic computing capabilities. We tested the autonomic computing capabilities in a wide range of workloads. We varied our workloads to consider different combinations of total users, concurrent users, and types of requests. We also tested the autonomic computing capabilities in a wide range of environmental condition. For example, we used different numbers of servers to host the original system under different constraints (e.g., network latencies or bandwidth). In each test, we compare the performance of the system with and without enabling the autonomic computing capabilities. We applied our prior data-driven performance test analysis techniques [3, 9–11, 17, 19] aiming to identify how and where the performance is different.

Enabling the autonomic computing capabilities achieves a much better performance than using the default configuration. Figure 3 shows the results of an example test in which we execute a very large workload to the system (similar workloads are expected in some real production environment). The key performance indicator ranges from 0 to 1, the lower the better. The figure shows the evolution of the key performance indicator over time, with or without enabling the autonomic computing capabilities. Figure 3 shows that after enabling the autonomic computing capabilities, we significantly improve the performance of the system (i.e., the key performance indicator improved from over 0.8 to nearly zero). Our autonomic computing capabilities also adapt to the changing workload very fast (i.e., in several minutes). Our tests under different environmental conditions and workloads show similar results. Note that the label and values on the figures are normalized due to the competitive nature of the system.

6.2 Testing of necessity

A potential question regarding our autonomic computing capabilities may be whether our framework is able to find a *single optimal configuration* that may work for all workloads. If that is the case, then the autonomic computing capabilities that we developed using our approach would not be valuable.

There does not exist a universally optimal configuration. In order to find out whether there is a universally optimal configuration that can be applied to all the environment and workloads, we designed experiments to apply previously-found optimal configurations to new tests. We first ran some tests with the autonomic computing capabilities enabled, under different environment and workloads. We collected the optimal configurations from these tests (the autonomic computing capabilities reach an optimal configuration for a certain working environment and a workload, and remain stable at the optimal configuration point). Then, we statically applied these optimal configurations to the same tests (i.e., under the same environment and workloads) *without* enabling the autonomic computing capabilities. We found that using the autonomic computing capabilities always achieves better performance than statically using a previously-found optimal configuration. We analyzed the reasons and we found that as the environmental condition and the workloads are highly dynamic, a static configuration cannot always adapt to the changing environment and workloads. Namely, the configurations are constantly tuned and improved until they reach a stable and optimal point. Hence, there is no single optimal configuration that will work for the system.

6.3 Testing of manual control

A key feature of our autonomic computing capabilities is that they are transparent to developers. Developers have full control of the autonomic computing capabilities. We tested various cases where manual control might be needed: 1) enabling/disabling the autonomic computing capabilities before the system services a workload; 2) enabling/disabling the autonomic computing capabilities while the system is servicing a workload; 3) disabling and re-enabling the autonomic computing capabilities when the system is servicing a workload; 4) manually changing the values of the configuration parameters when the autonomic computing capabilities are disabled; 5) manually changing the values of the configuration parameters when the autonomic computing capabilities are enabled. In all cases, we found that developers were able to use the manual control without any problems.

6.4 Testing of robustness

We want to ensure that our autonomic computing capabilities work expectedly under extreme workloads. Therefore, we tested, under extreme workloads: 1) whether the autonomic computing capabilities would fail and what is the impact of a failure; 2) whether the autonomic computing capabilities would bring negative impact (e.g., finding worse configurations); 3) whether we can still manually control the autonomic computing capabilities; 4) what would happen if we enable/disable the autonomic computing capabilities in the middle of a workload. Our test results show that our autonomic computing capabilities are robust under extreme workloads.

6.5 Testing of high variance workloads

In a real production environment, the workload is constantly changing. Therefore, we want to test how the autonomic computing capabilities behave under a high variance workload. We introduced a significant amount of variance in the workloads when the autonomic computing capabilities are enabled. Our testing results show that our autonomic computing capabilities can quickly adapt to the changing workloads (i.e., in several minutes) and optimize the performance to desired ranges.

6.6 Field testing

We tested our autonomic computing capabilities in a real production environment. Our testing results show that, under production workloads, our autonomic computing capabilities achieve similar performance improvement as in the testing environment.

6.7 Adoption of our autonomic computing capabilities

We presented our approaches and experimental results to the production solution team, who makes plans for new features or enhancing current production deployment. Our autonomic computing capabilities are very well-received by the developers and are integrated into the production environment.

> *Through testing of our autonomic computing capabilities, we show that they can effectively improve the performance of the system while freeing developers from manually tuning configuration parameters. Our autonomic computing capabilities are also tested to be robust under extreme or high variance workloads.*

7 CONCLUSION

As software systems become increasingly complex, human costs for system operation and maintenance play a more and more significant role in the overall cost of software systems. Autonomic computing was proposed to reduce the necessity for human intervention in large software systems. However, it is challenging to add autonomic computing capabilities to the existing systems that were not designed with the autonomic computing mindset. In this paper, we document our experience of successfully adding autonomic computing capabilities into an existing large-scale non-autonomic system with minimal changes to the original system. In particular, we focus on the software engineering challenges that we encountered during the process, how we overcame these challenges, and the lessons that we learned. We also share our high-level implementation of the autonomic computing capabilities and how we test such capabilities under different conditions. We believe that our experience on transforming existing systems to become autonomic can help software practitioners who also want to adopt autonomic computing capabilities to existing non-autonomic large-scale systems. As there are many large-scale software systems that still suffer from expensive human costs, we encourage future work on investigating research methodologies that can help developers add autonomic computing capabilities to these existing software systems in a cost-effective manner.

REFERENCES

[1] Mehdi Amoui. 2009. Evolving software systems towards adaptability. In *Proceedings of the 16th Working Conference on Reverse Engineering (WCRE '09)*. IEEE, 299–302.

[2] Tse-Hsun Chen, Weiyi Shang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2016. CacheOptimizer: Helping Developers Configure Caching Frameworks for Hibernate-based Database-centric Web Applications. In *Proceedings of the 24th International Symposium on the Foundations of Software Engineering (FSE '16)*. 666–677.

[3] Tse-Hsun Chen, Mark D. Syer, Weiyi Shang, Zhen Ming Jiang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2017. Analytics-driven Load Testing: An Industrial Experience Report on Load Testing of Large-scale Systems. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '17)*. 243–252.

[4] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, Berlin, Heidelberg, 1–32.

[5] Simon Dobson, Roy Sterritt, Paddy Nixon, and Mike Hinchey. 2010. Fulfilling the vision of autonomic computing. *Computer* 43, 1 (2010), 35–41.

[6] Jerome H Friedman. 1991. Multivariate adaptive regression splines. *The Annals of Statistics* 19, 1 (1991), 1–67.

[7] Pascal Van Hentenryck and Laurent Michel. 2009. *Constraint-based local search*. The MIT press.

[8] IBM. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31 (2006).

[9] Zhen Ming Jiang, Ahmed E. Hassan, Gilbert Hamann, and Parminder Flora. 2008. An automated approach for abstracting execution logs to execution events. *J. Softw. Maint. Evol.* 20, 4 (2008), 249–267.

[10] Zhen Ming Jiang, Ahmed E. Hassan, Gilbert Hamann, and Parminder Flora. 2008. Automatic identification of load testing problems. In *Proceedings of the 24th International Conference on Software Maintenance (ICSM '08)*. 307–316.

[11] Zhen Ming Jiang, Ahmed E. Hassan, Gilbert Hamann, and Parminder Flora. 2009. Automated performance analysis of load tests. In *Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM '09)*. 125–134.

[12] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[13] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*. 220–242.

[14] Nicholas Metropolis and Stanislaw Ulam. 1949. The Monte Carlo method. *Journal of the American statistical association* 44, 247 (1949), 335–341.

[15] James J Mulcahy and Shihong Huang. 2014. Autonomic Software Systems: Developing for Self-Managing Legacy Systems. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME '14)*. 549–552.

[16] James J Mulcahy and Shihong Huang. 2015. An autonomic approach to extend the business value of a legacy order fulfillment system. In *Proceedings of the 9th Annual IEEE Systems Conference (SysCon '15)*. 595–600.

[17] Weiyi Shang, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2015. Automated Detection of Performance Regressions Using Regression Models on Clustered Performance Counters. In *Proceedings of the 6th International Conference on Performance Engineering (ICPE '15)*. 15–26.

[18] Gregory T Sullivan. 2001. Aspect-oriented programming using reflection and metaobject protocols. *Commun. ACM* 44, 10 (2001), 95–97.

[19] Mark D. Syer, Zhen Ming Jiang, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. 2014. Continuous Validation of Load Test Suites. In *Proceedings of the 5th International Conference on Performance Engineering (ICPE '14)*. 259–270.

[20] Steve R White, James E Hanson, Ian Whalley, David M Chess, and Jeffrey O Kephart. 2004. An architectural approach to autonomic computing. In *Proceedings of the 1st IEEE International Conference on Autonomic Computing (ICAC '04)*. 2–9.

[21] Kenji Yamanishi and Yuko Maruyama. 2005. Dynamic Syslog Mining for Network Failure Monitoring. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '05)*. 499–508.

[22] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2011. Improving software diagnosability via log enhancement. In *Proceedings of the 16th International Conference on Architectural support for programming languages and operating systems (ASPLOS '11)*. 3–14.

[23] Xu Zhao, Kirk Rodrigues, Yu Luo, Ding Yuan, and Michael Stumm. 2016. Non-intrusive Performance Profiling for Entire Software Stacks Based on the Flow Reconstruction Principle. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI '16)*. 603–618.