Do External Feedback Loops Improve the Design of Self-Adaptive Systems? A Controlled Experiment

Danny Weyns, M. Usman Iftikhar, and Joakim Söderlund
Department of Computer Science, Linnaeus University, Växjö, Sweden
{danny.weyns, usman.iftikhar}@lnu.se, jsooa08@student.lnu.se

***Abstract*—Providing high-quality software in the face of uncertainties, such as dealing with new user needs, changing availability of resources, and faults that are difficult to predict, raises fundamental challenges to software engineers. These challenges have motivated the need for self-adaptive systems. One of the primary claimed benefits of self-adaptation is that a design with external feedback loops provide a more effective engineering solution for self-adaptation compared to a design with internal mechanisms. While many efforts indicate the validity of this claim, to the best of our knowledge, no controlled experiments have been performed that provide scientifically founded evidence for it. Such experiments are crucial for researchers and engineers to underpin their claims and improve research. In this paper, we report the results of a controlled experiment performed with 24 final-year students of a Master in Software Engineering program in which designs based on external feedback loops are compared with designs based on internal mechanisms. The results show that applying external feedback loops can reduce control flow complexity and fault density, and improve productivity. We found no evidence for a reduction of activity complexity.**

I. Introduction

The upcoming generation of software systems will increasingly consist of loosely coupled interacting subsystems. Examples are service-based systems to support business collaborations, large-scale mobile applications, smart homes, and multi-robot systems. Engineering these systems and guaranteeing the required qualities (performance, robustness, etc.) is complex due to the inherent uncertainty resulting from incomplete knowledge at design time. Among the uncertainties are new user needs, subsystems that come and go at will, dynamically changing availability of resources, and faults and intrusions that are difficult to predict. The challenges of the next generation software systems have motivated the development of self-adaptive software systems.

Self-adaptation endows a system with the capability to adapt itself autonomously to internal changes and dynamics in the environment in order to achieve particular quality goals in the face of uncertainty. A self-adaptive system consists of two parts: a managed system and a managing system [1], [2], [3]. The managed system is situated in an environment and provides some functionality to users. The managing system realizes a feedback loop that adapts the managed system when needed, according to some goals. In particular, the managing system comprises the software to monitor and reason about the managed system and its environment, and perform adaptations of the managed system when needed. In practice, the software of the managed system and the managing system may be clearly partitioned in separate modules or layers, or the software of the two parts may be (partially) interwoven.

Over the past fifteen years, researchers have developed a vast body of work on engineering self-adaptive systems [4], [5], [6]. Two prominent lines of research in this field are architecture-based self-adaptation and control-based self-adaptation. Both lines of research recognize the crucial role of feedback loops in realizing self-adaptation of software systems, but from different viewpoints. Architecture-based self-adaptation [7], [2], [8], [9] emphasizes software components for feedback loops, runtime models and mechanisms, and the interaction with the managed system. Control-based self-adaptation [10] applies control theory to synthesize and analyze feedback control loops for computing systems. Our focus here is on architecture-based self-adaptation, in particular, the design of self-adaptation using software components that realize feedback loops for particular quality properties.[1]

One of the primary claimed benefits of self-adaptation is that external feedback loops provide a more effective engineering solution for self-adaptation. In one of the most influential papers of the field [2], the authors state:

> *External control mechanisms provide a more effective engineering solution than internal mechanisms for self-adaptation because they localize the concerns of problem detection and resolution in separable modules.*

While many efforts indicate the validity of this statement, to obtain objective and statistically relevant underpinning of it, we need formal, rigorous investigations. Empirical experiments provide the means for this, but to the best of our knowledge, no controlled experiments have been performed that provide evidence for the benefit of external feedback loops. This paper contributes with a controlled experiment that compares the design of self-adaptive systems using explicit monitor-analysis-plan-execute (MAPE) loops with designs that use internal adaptation mechanisms. The concrete goal of the empirical experiment is to address the following research question:

> **Analyze** the design of self-adaptive systems
> **for the purpose** of evaluating internal adaptation mechanisms and external MAPE loops

[1] We do not consider control-based approaches as the design of these solutions are based on different techniques, such as target system identification, control algorithm design, mathematical analysis of response, etc.

with respect to their design complexity, fault density, and productivity,
from the viewpoint of researchers,
in the context of final-year master students in software engineering that add different self-adaptation properties to the design of a distributed software application.

With internal adaptation mechanisms, we refer to a design that realizes adaptation using tactics and techniques that are integrated within the components of the managing system. With external MAPE loops, we refer to a design that realizes adaptation using a monitor-analysis-plan-execute loop that is added to (and thus external to) the managed system. We focus on three important measures of effectiveness of engineering: design complexity, fault density, and productivity. Design complexity allows comparing the difficulty in producing and understanding a design. It is a key factor with respect to comprehensibility, testing, and maintainability [11]. Fault density allows comparing the relative number of faults in software components, which is an important criteria for the quality of a design [12]. It can be used for example to define the effort and time that is required for testing. Productivity allows comparing the amount of functionality that can be produced per time unit. Productivity directly relates to size, cost, and work effort required to engineer a software system [13].

The experiment is performed in the context of a course in a Software Engineering Master program at Linnaeus University, where final-year students had to create different designs for a distributed traffic monitoring application. All the material of the experiment, including course material, the design and implementation of the traffic monitoring application, the assignments of the tests, and the experiment results are available at the experiment website.[2]

The remainder of this paper is structured as follows. Section II gives an overview of the experiment setting. In Section III, we explain planning of the experiment. Section IV presents the analysis and discussion of the experiment results. Threats to validity are explained in Section V. We discuss related work in Section VI, and draw conclusions in Section VII.

II. Experiment Setting

The experiment took place as part of a nine-week course on Software Architectures for Adaptive Systems. The subjects of the experiment are 24 final-year master students in software engineering. The objects are the *reference approach* and the *MAPE approach*. With the reference approach subjects are instructed to use internal mechanisms to extend a given design for a requested adaptation property, that is, the functionality for the property has to be integrated within the components of the design. With the MAPE approach subjects are instructed to use an external MAPE loop to extend the design for a requested adaptation property. Fig. 1 shows a scenario of the application.

The application is a traffic monitoring system that consists of a set of intelligent cameras, which are distributed along the road. Each camera has a limited viewing range and cameras

[2]http://homepage.lnu.se/staff/daweaa/experiments/FeedbackLoops/index.htm

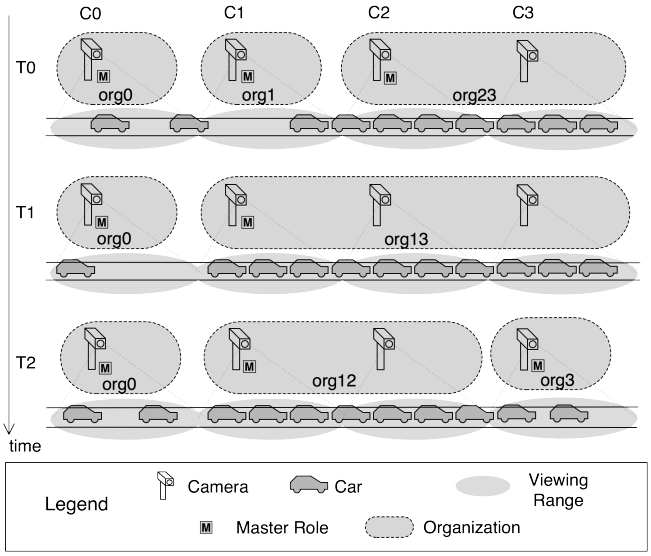


Figure 1. Scenario of the application used in the experiment

are placed to get an optimal coverage of the highway with a minimum overlap. The system provides information about traffic jams to clients, which can be traffic light controllers, driver assistance systems, etc. To realize this functionality, the cameras collaborate in organizations: if a traffic jam spans the viewing range of multiple cameras, they form an organization (structured as master-slave) that provides information to clients with an interest in traffic jams. When traffic conditions change, the organizations dynamically adapt as illustrated in Fig. 1.

Fig. 2 shows the primary components deployed on a camera.

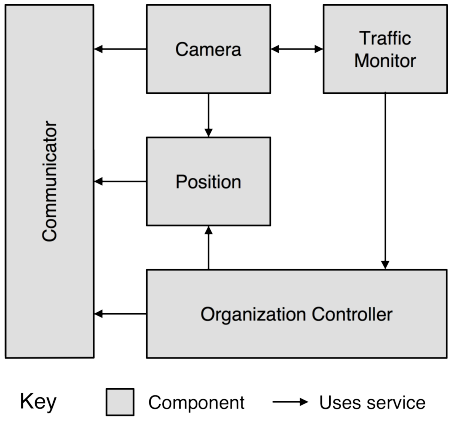


Figure 2. Primary components deployed on a camera

The main responsibilities of the components are:

- Camera provides a service for retrieving the current traffic conditions on the monitored road segment.
- Traffic Monitor provides a service for determining whether or not the locally monitored traffic is congested.
- Organization Controller provides a service for merging and splitting organizations. Organization management is the responsibility of the cameras in the role of master.

- Position provides a service to obtain information about the physical positions of cameras.
- Communicator provides a service for inter-node communication.

In the experiment, we considered two types of self-adaptation properties: i) enable the system to deal with silent node failures[3] (robustness), and ii) enable the system to add new cameras dynamically (openness).

For the MAPE approach, a new node is added to the system where components can be defined that realize the MAPE functions. These components can interact with the cameras via additional proxy components that are added to each camera. The proxy components can access the regular components of the cameras via get and set methods.

Fig. 3 gives an overview of the course. The course was split in two parts. Each part consisted of two or three weeks lectures and a home task, followed by two weeks for evaluation (when students did the tests of the experiment).

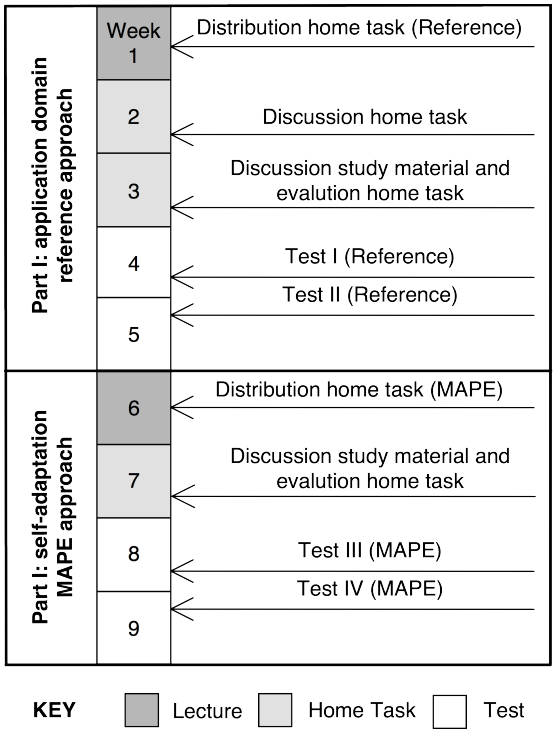


Figure 3. Overview of 9-weeks course in which the experiment took place

In part I, students are educated on the traffic monitoring application and on tactics to realize adaptation for different types of quality properties, which fully covers the reference approach. In part II, students are educated on self-adaptation, with particular attention for MAPE loops, which covers the MAPE approach. For the home task in part I, students were instructed to study: i) the design of the application using the architecture documentation and the implementation, and ii) tactics to realize adaptation, using book chapters and articles.

[3]A camera becomes unresponsive without sending any incorrect data.

For the home task of part II, students had to study a set of key papers in the field of self-adaptation, and hand in a summary with a set of questions and/or critical remarks. For each home task, an evaluation session was organized where students and instructors discussed the study material and evaluated the task results. For the first home task, an extra intermediate discussion session was organized where students could discuss questions with the instructors.

The experiment took place in the last two weeks of both parts. In each session (of three and half hours), subjects received an assignment to extend the design of the application for an adaptation property (with pen and paper), using a specific approach (i.e., the reference approach or the MAPE approach). For the course, students are graded on all tests.

III. Experiment Planning

Subsequently, we discuss assignments, experiment design, hypotheses, and independent and dependent variables.

A. Assignments

We used four different assignments of the traffic monitoring application, allowing to have subjects solving different tasks in different experiment sessions. The difficulty to solve the problem was of the same level for all assignments. The following table summarizes the assignments:

Assignment	Task
A1	Robustness scenario 1
A2	Openness scenario 1
A3	Robustness scenario 2
A4	Openness scenario 2

We used assignments for two adaptation properties: robustness and openness. For each property we used two different scenarios. In assignments A1 and A3, subjects have to add support to deal with silent node failures. In A1 subjects have to extend the given design of the traffic monitoring system to deal with such a failure using the reference approach. In A3, subjects have to extend the given design to handle such a failure using the MAPE approach. The scenarios of A1 and A3 are different, but the problem difficulty is equal. In assignments A2 and A4, subjects have to add support to add dynamically a new camera to the system, respectively with the reference approach and the MAPE approach. Similarly as for A1 and A3, the scenarios of A2 and A4 are different, but the difficulty of the problems is equal. Fig. 4 shows an example scenario for openness.

Both assignments comprise the following parts: (1) a problem description; (2) a description of the task; (3) a set of assumptions and constraints, and (4) the given design. The problem description explains the adaptation property that needs to be added to the system, illustrated with a scenario (example shown in Fig. 4). The problem description lists the concrete tasks the students have to do. Assumptions describe things that a subject can take for granted when he or she solves the problem. As an example, subjects could assume that the Communicator component of a camera has the necessary logic to communicate with other cameras based on the camera ID.

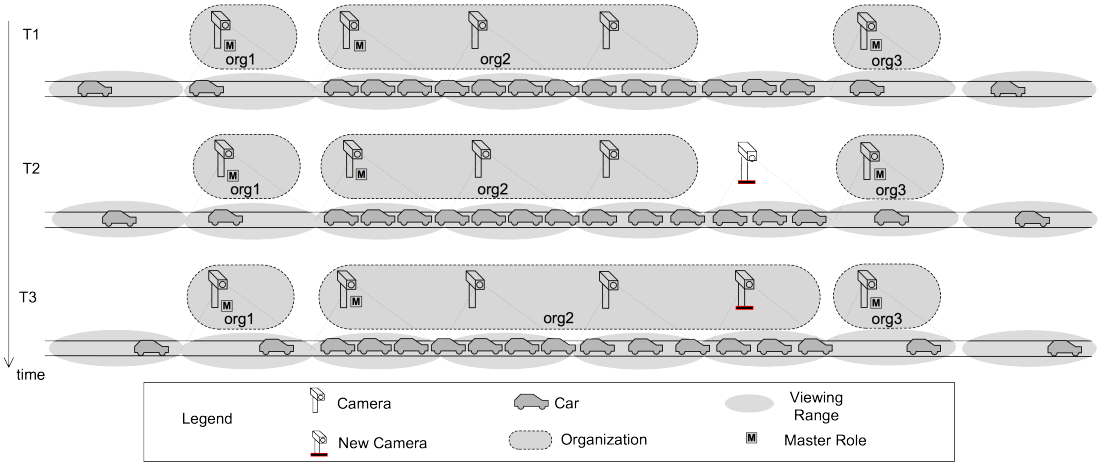


Figure 4. Example scenarios of the assignments for openness.

Constraints describe restrictions with respect to the solution. For example, when using the MAPE approach, the list of methods of existing components can only be extended with get- and set-methods, which represent probes and effectors respectively (the get- and set-methods are used by the proxies, see Section II). The design consists of a specification of the following parts: (1) the components with their dependencies, (2) the component interfaces, (3) the messages exchanged between nodes, and (4) interaction diagrams[4]. The design is specified in the form of a template that allows subjects to extend and adapt the existing design elements; e.g., add a component, add/remove a message, and add new interaction diagrams. Subjects had to use pen and paper during the test.

B. Experiment Design

The experiment was conducted as a block subject-object quasi-experiment. Blocked subject-object means that each subject receives both treatments (the reference approach and MAPE approach). Concretely, each subject received both treatments two times, one for each adaptation property (robustness and openness). This allows paired comparison of samples. The experiment is a quasi-experiment [14] because it is performed on a single group and there is no randomization of the order in which the treatments are applied to the subjects. Fig. 5 shows a summary of the experiment design.

Each box represents a test by a group of subjects (i.e., half of the total group) using a particular treatment. Thus in the first experiment session (week 3), half of the subjects receives assignment A1, the other half receives assignment A2. This is done in a randomized fashion. In the second session (week 4), the assignments are switched: subjects who received assignment A1 in experiment session I now receive assignment A2, and vice versa. A similar approach is used in the second part of the course with assignments A3 and A4 (in weeks 8 and 9). Additional aspects relevant to the experiment design are:

[4]The design consisted of all the required material to solve the assignment. However, subjects could use the complete architecture specification of the traffic monitoring application during the tests.

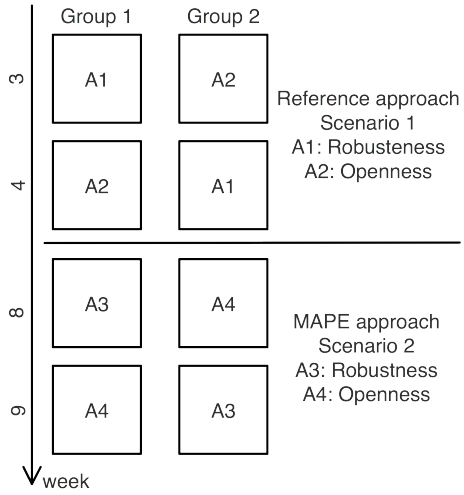


Figure 5. Experiment design

the subjects were not aware of the experiment and did not receive any feedback on the experiment sessions (including their grades) until after the fourth experiment session.

C. Hypotheses Formulation

The research goal of this experiment is to compare the use of internal adaptation mechanisms and external MAPE loops with respect to design complexity, fault density, and productivity (see Section I). This goal can be refined in 3 sub-goals that map to a set of hypotheses. In particular, each sub-goal maps to a null hypothesis to be tested, and an alternative hypothesis in favor and to be accepted if the null hypothesis is rejected. We formulate 3 null hypotheses (H_0) and three alternative hypotheses (H_a):

- H_{01}: There is no difference in *design complexity* between a design created using the reference approach and a design created using the MAPE approach.

$$H_{01} : \mu_{complexity_{Ref}} = \mu_{complexity_{MAPE}} \quad (1)$$

$$H_{a1} : \mu_{complexity_{Ref}} > \mu_{complexity_{MAPE}} \quad (2)$$

- H_{02}: There is no difference in *fault density* between a design created using the reference approach and a design created using the MAPE approach.

$$H_{02} : \mu_{fault.density_{Ref}} = \mu_{fault.density_{MAPE}} \quad (3)$$

$$H_{a2} : \mu_{fault.density_{Ref}} > \mu_{fault.density_{MAPE}} \quad (4)$$

- H_{03}: There is no difference in *design productivity* between using the reference approach or using the MAPE approach.

$$H_{03} : \mu_{productivity_{Ref}} = \mu_{productivity_{MAPE}} \quad (5)$$

$$H_{a3} : \mu_{productivity_{Ref}} < \mu_{productivity_{MAPE}} \quad (6)$$

D. Independent and Dependent Variables

Each hypothesis requires the definition of a set of independent and dependent variables, and a selection of proper metrics to measure the dependent variables.

1) Independent Variables: Independent variables are variables in the experiment that can be manipulated and controlled. In our experiment, there are two independent variables:

- Approach: The treatment used by a subject to solve an assignment. This variable is the factor of the experiment that is changed to see the effect on the dependent variables. The two possible values of this factor are the reference approach and the MAPE approach.
- Assignment: The problem to be solved by the subject (assignments A1 to A4). Since the assignments have the same level of difficulty, the assignment is not considered as a factor but as a fixed variable.

2) Dependent Variables: Dependent variables are variables that we want to study to see the effect of different treatments. For each hypothesis, we defined the corresponding dependent variables and selected proper metrics to measure the effects.

a) Design Complexity: Two representative measures for design complexity are: (1) average activity complexity (AC) per process, and (2) average control flow complexity (CFC) per process [13], [15], [16]. AC per process is measured by counting the number of activities in a process. We measured AC as follows:

$$AC\ per\ process = \frac{\sum_{p\ \epsilon\ process} AC(p)}{\#\ process} \quad (7)$$

A process is a logically grouped sequence of activities in the system. An example of a process is the sequence: a component receives a message, updates state, and replies with a message. Another example is: a timer expires and as a consequence, a component sends messages to a set of other components.

Control flow complexity (CFC) is measured by counting the number of control flow primitives (ifs, loops, etc.) per process. We measured CFC as follows:

$$CFC\ per\ process = \frac{\sum_{p\ \epsilon\ process} CFC(p)}{\#\ process} \quad (8)$$

b) Fault Density: Fault density is commonly defined as the number of known faults divided by the product size [12]. A fault is a mistake in a software product as the result of a human error. In our experiment, we measured fault density in the design. Simply counting the number of faults, however, does not measure the impact of faults. A more precise metric is to measure the amount of change that is required to make the design work [12]. Size is measured as the amount of functionality that is supported by the design. To measure the amount of functionality we used Albrecht's approach to calculate function points [17]. Function points measure the amount of functionality in a system and can be used as a measure for software size [12]. Examples of function points in the assignments are: a component sends a message to another component, a component receives a message, a component updates the state of a variable, etc. The total number of function points per assignment is shown in the following table:

Assignment ID	Scenario	Total function points
A1	Robustness scenario 1	22
A2	Openness scenario 1	21
A3	Robustness scenario 2	28
A4	Openness scenario 2	26

Based on this, we measured fault density as the number of changes divided by the number of supported function points:

$$fault\ density = \frac{\#\ changes}{\#\ supported\ function\ points} \quad (9)$$

Examples of changes are: add/change an if statement, add a communication link, etc. The experiment website gives an overview of the supported function points and cost of changes.

c) Productivity: Productivity is defined as the amount of functionality that developers can produce per time unit [12]. We measured functionality as follows:

$$functionality = \#\ supported\ function\ points \quad (10)$$

We measured productivity as follows:

$$productivity = \frac{functionality}{time\ spend\ on\ design} \quad (11)$$

IV. Analysis of the Results and Discussion

Data was collected independently by two instructors. For each assignment, the instructors determined the supported function points, trimmed the design, and corrected the trimmed design. The number of supported function points were counted using the interaction diagrams. Trimming excluded parts of the design that do not contribute to the actual functionality.[5] Then the metrics for each dependent variable are applied to the corrected design, and differences between the collected data of the two instructors where resolved if needed.

A. Analysis

In total, 21 subjects provided usable data for paired comparison[6] of fault density, complexity, and productivity.

[5]The parts excluded during trimming were similar for both approaches.

[6]A number of subjects provided incomplete or inconsistent data, or did not attend all experiment sessions, invalidating their data for paired comparison.

Table I and Fig. 6 summarize the measurements for all dependent variables, and the paired difference between the treatments. The table also shows the number of subjects that performed better, equal, or worse for the MAPE approach.

The paired difference Z_i is defined as $Z_i = M_i - R_i$ for $i = 1, ..., n$. M_i and R_i are the measurements of subject i for respectively the MAPE and the reference approach; n is the number of subjects that produced data for both treatments.

To select a proper statistical test, we compare the distribution of Z_i for each dependent variable with the standard normal distribution, using the Anderson-Darling test [18]. This results in the following p-values:

Variable	p-value Anderson-Darling test
AC per process	0.0659
CFC per process	0.3359
Fault density	0.0031
Functionality	0.5152
Productivity	0.0211

With a significance level (α) of 0.05, we assume Z_i to be normally distributed only for the dependent variables AC per process, CFC per process, and functionality. Based on this assumption, we use the paired t-test [19] to test our hypotheses for AC per process, CFC per process, and functionality, and the Wilcoxon signed-rank test [20] to test our hypotheses for the two other dependent variables.

This results in the following p-values for our hypotheses:

	p-value	Statistical test
AC per process	0.7842	paired t-test
CFC per process	0.0028	paired t-test
Fault density	0.0310	Wilcoxon signed-rank test
Functionality	0.0001	paired t-test
Productivity	0.0060	Wilcoxon signed-rank test

With a significance level of 0.05, every null hypothesis is rejected except for AC per process.

B. Discussion

Except for activity complexity, the descriptive analysis shows that there is a clear improvement for the dependent variables when subjects use the MAPE approach compared to the reference approach. This is confirmed by the statistical tests.

The only dependent variable that is not improved is activity complexity. The measurements show somewhat higher values for the MAPE approach (4.516) compared to the reference approach (4.147). The functionality that is required to add the requested adaptation properties is similar for both approaches, which explains the similar values measured for activity complexity. The main difference between both treatments in this respect is the allocation of the required functionality to components (with the reference approach, the functionality is integrated in the existing components, with the MAPE approach, the functionality is allocated to components of the external feedback loop).

There are several explanations for the improvements of the other dependent variables. On average, control flow complexity is about 37% lower with the MAPE approach (0.787) compared to the reference approach (1.253). Thus, using MAPE reduces the number of control flow primitives per process that are required to realize the adaptation properties. Closer examination of the designs with the MAPE approach reveals that subjects create more processes (allocated to the components of the MAPE loop). We counted the number of processes and on average subjects used 2.66 processes with the reference approach and 5.0 with the MAPE approach. This divide and concur approach results in processes that are simpler in terms of number of control flow primitives, which explains the decrease of control flow complexity of the MAPE approach.

On average, the number of required changes per supported function point (fault density) is about 32% lower with the MAPE approach (1.001) compared to the reference approach (1.458). With the MAPE approach subjects divide the responsibilities of the adaptation functions (monitor, analyze, plan, execute) over different components. We counted the number of extra components used with the MAPE approach. About half of the subjects define explicit components for each of the adaptation functions, with 2.75 extra components on average. This better modularization and separation of concerns allows the subjects to focus on the specific functions of adaptation, which likely lead to a smaller chance on faults.

Using the MAPE approach has also a positive impact on productivity. The total number of supported function points for the reference approach is 8.037 and 13.185 for the MAPE approach. This is an increase of 64%. We measured the total time that subjects have spent on their design, which is on average 2h07min for the reference approach and 2h34min for the MAPE approach. This results in an average productivity (supported function points per hour) of 3.896 for the reference approach and 5.420 for the MAPE approach, an increase of 39%. Two possible explanations for this improvement are: availability of higher level building blocks (MAPE components), and reduced design complexity. The availability of higher level building blocks that map to the core functions of adaptation and a reduction in design complexity makes it more easy for subjects to reason about the adaptation behavior of the system, improving productivity.

Despite significant improvements of productivity, we make a remark regarding the results. The dependent variables we used to check the hypothesis of productivity measure how productive subjects are with respect to realizing functionality in terms of supported function points per time unit. During analysis of the results, we also looked at productivity with respect to the realization of the overall functionality that is required for the requested adaptation requirements. To that end, we introduced an additional variable, called effective productivity, defined as follows:

$$supported\ part\ of\ design = \frac{\#\ supported\ function\ points}{\#\ required\ function\ points} \quad (12)$$

$$effective\ productivity = \frac{supported\ part\ of\ design}{time\ spend\ on\ design} \quad (13)$$

The effective productivity expresses what fraction of the total required functionality the subjects realize per time unit.

The resulting p-value for effective productivity is 0.202 (paired t-test), which gives only a weak indication that subjects are also more productive in terms of realizing the overall functionality for an adaptation property when using the MAPE approach compared to the reference approach.

V. Threats to Validity

The design of the experiment introduces some threats to validity [21], [19]. We discuss the main identified threats.

A. Threats to Construct Validity

Construct validity is the degree to which the operationalization of measures in the study actually represent the constructs in the real world. We see two such threats. First students might try to perform better with the MAPE approach to impress the course holders (who do research in self-adaptive systems). To reduce this threat, the students were not aware of the experiment and students were graded on all experiment sessions (and were aware of this). Second, the experiment relies on a set of assignments that might not fully represent the type of problem on which we want to test both treatments. We belief that the assignments provide representative problems for self-adaptation, as the problem domain has been used for illustration and evaluation of self-adaptive systems in several research articles, e.g., [3]. We further tried to reduce this threat by using two different types of problems for each treatment.

B. Threats to Internal Validity

Internal validity is the extent to which independent variables are actually responsible for the effects seen to the dependent variables. For all subjects, the effects of the first treatment (reference approach) are observed in the first and second experiment sessions (weeks 4 and 5), and the effects of the second treatment (MAPE approach) are observed in the third and fourth sessions (weeks 8 and 9). This introduces three potential threats:

1) Increased Understanding: A subject's understanding of certain concepts (the design of the traffic monitoring system, basic tactics, etc.) can increase between the first and the second series of observations. Increased understanding of the concepts may affect the measurements, in particular, fault density and productivity. To reduce this threat, the students had three full weeks to learn the application domain, the design of the traffic monitoring system, and the basic tactics to realize self-adaptation, prior to the first experiment session. In addition, the design was documented using standard UML, which is basic knowledge for the students. Furthermore, students had already basic knowledge of most of the tactics. The students were asked to fully master both the design and the tactics. Students with questions could discuss the material using a communication platform, and during an intermediate discussion session. At the end of week 3, an evaluation session was organized where students had to demonstrate that they had mastered the study material. From that point on, the application domain and basic tactics were considered to be known and no more time was spent on it, limiting possible learning effects.

To verify the possible effect of increased understanding, we looked at the measurements of only test II (week 5) and test IV (week 9), anticipating a possible learning effect obtained from the preceding tests[7]. Analysis shows the following results:

Variable	Wilcoxon signed-rank	Paired t
AC per process	0.8365	0.8046
CFC per process	0.0249	0.0166
Fault density	0.0884	0.0404
Functionality	0.0014	0.0010
Productivity	0.0615	0.0437

The underlined values represent the results with appropriate tests according to the distributions of measurements. Although there is no longer strong statistical evidence for reduced fault density and improved productivity (with the reduced number of measurements), the results support the assumption that the possible effects of increased understanding of concepts are limited.

2) Maturing: Subjects can mature between the two observations, for example, by taking the experiment more serious. This threat is reduced by keeping the subjects unaware of the experiment and grading results (during the experiment), but aware that they will be graded on each experiment session.

3) Learning the type of Problem: Subjects can learn the problems they have to solve, making them better prepared in the second series of experiment sessions. To reduce this threat, we used different concrete problem scenarios in subsequent sessions when subjects had to solve a problem for the same adaptation requirement.

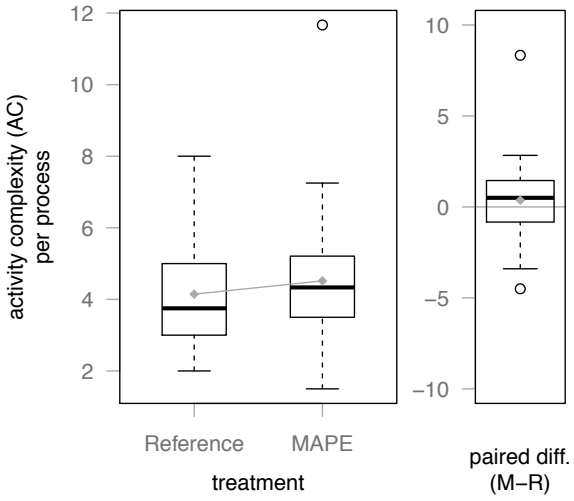
C. Threats to External Validity and Conclusion Validity

External validity is the degree to which findings of a study can be generalized to other subject populations and settings. Conclusion validity concerns generalizing the result of the experiment to the concept or theory behind the experiment.

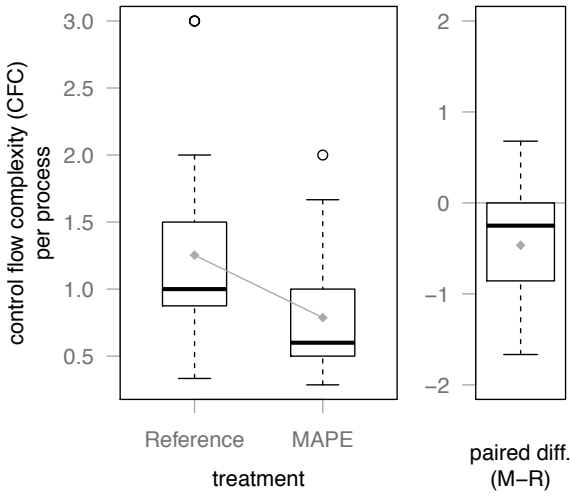
Due to practical restrictions, we used students of a Master in Software Engineering program as subjects for our study. Although these students do not represent expert software engineers, they are the next generation of software professionals and are relatively close to this population [22].

The experiment relies on assignments that are created by the authors of this paper, two of them being researchers in the area of self-adaptation. This creates a potential threat that the assignments are no realistic representation of the underlying problem, and favor the MAPE approach over the reference approach. This threat could have been reduced by asking external experts to produce the assignments; however, due to constraints, this was not possible for the experiment. To minimize bias towards the MAPE approach, we used the design of an existing system in the experiment and asked the creator of this system (and author of the design documentation) to review all the assignments. The feedback was used to make

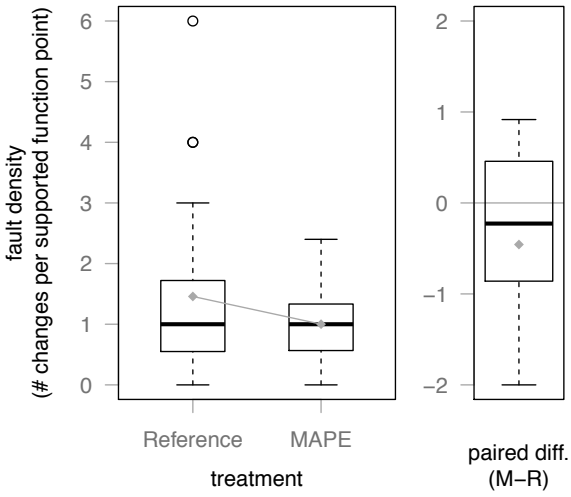
[7]To get enough measurements for paired comparison, besides regular data, we also used the results of three subjects from their first test (reference approach) that produced unusable data in the second test (while they produced useful results in the fourth test).



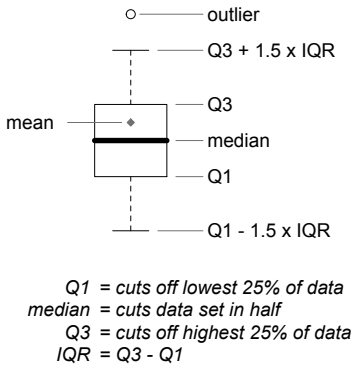
(a) Average activity complexity (AC) per process.



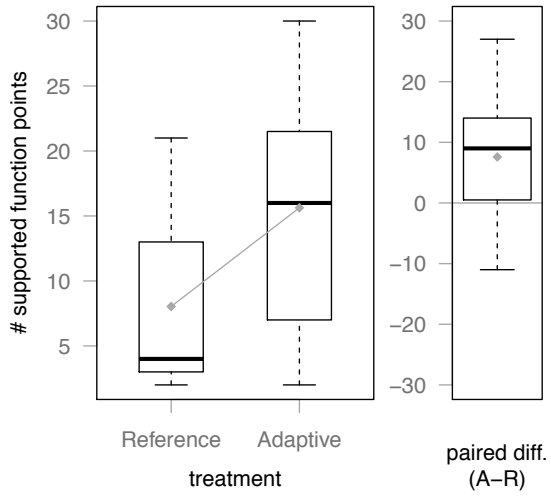
(b) Average control flow complexity (CFC) per process.



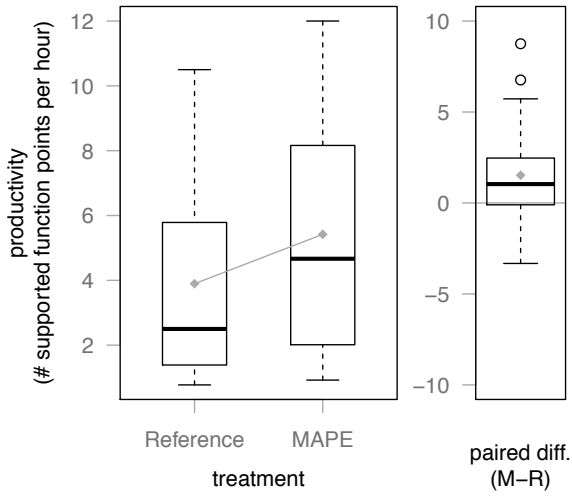
(c) Fault density.



(d) Key.



(e) Functionality.



(f) Productivity.

Figure 6. Box plots for all measurements.

Table I
MEASUREMENTS FOR ALL DEPENDENT VARIABLES.

Activity complexity (AC) per process						
	mean (μ)	median	st.dev	better	equal	worse
Reference	4.147	3.750	1.443			
MAPE approach	4.516	4.333	1.978	9	2	16
Paired Diff ($M_i - R_i$)	0.370	0.500	2.405			
Average control flow complexity (CFC) per process						
	mean (μ)	median	st.dev	better	equal	worse
Reference	1.253	1	0.726			
MAPE approach	0.787	0.600	0.449	18	3	6
Paired Diff ($M_i - R_i$)	-0.466	-0.250	0.801			
Fault density (number of changes per supported function point)						
	mean (μ)	median	st.dev	better	equal	worse
Reference	1.458	1	1.399			
MAPE approach	1.001	1	0.636	16	0	11
Paired Diff ($M_i - R_i$)	-0.457	-0.227	1.218			
Functionality (number of supported function points)						
	mean (μ)	median	st.dev	better	equal	worse
Reference	8.037	4	6.400			
MAPE approach	13.185	14	7.696	20	0	7
Paired Diff ($M_i - R_i$)	5.148	4	6.106			
Productivity (number of supported function points per time unit)						
	mean (μ)	median	st.dev	better	equal	worse
Reference	3.896	2.5	3.194			
MAPE approach	5.420	4.667	3.436	18	0	9
Paired Diff ($M_i - R_i$)	1.523	1.038	2.792			

some adaptations to the assignments, to minimize bias towards the MAPE approach.

In addition, we made a number of practical decisions that should be taken into account when generalizing our findings:

- using pen and paper to write down designs;
- allowing subjects to make a number of assumptions about the systems and their context;
- using standard UML notations.

Finally, there might be a threat with respect to the reliability of the measures. The measures are applied to corrected and trimmed designs. To increase the reliability and objectiveness of measures, all designs were corrected and analyzed independently by two instructors. Then the results were compared and discussed in case of conflicts to come to an agreement.

VI. RELATED WORK

To the best of our knowledge, so far, no controlled empirical experiments have been performed that provide evidence for the claims associated with self-adaptation, in particular w.r.t. the impact on engineering these systems. Nevertheless, there are a number of papers that report results of empirical studies that are related to the experiment presented in this paper.

[23] reports a controlled experiment on the use of design patterns with respect to maintenance activities. The subjects were professional software engineers that performed nine software maintenance scenarios employing various design patterns and simpler alternatives. For most tasks, the authors found improvements of using patterns, such as a reduction of maintenance time. They also found some negative effects, including simpler solutions that were less error-prone compared to solutions with patterns. There are similarities between this study and our experiment, where an external feedback loop can be considered as a pattern to realize self-adaptation. However, the focus of [23] is on different patterns and on maintenance activities, while our focus is on MAPE loops and design activities. Evidently, our findings, in particular design complexity, are potentially valuable for maintenance activities as well, but that was not the focus of our experiment.

In [24] the authors report the results of an experiment that verifies whether aspect-oriented approaches support improved modularization of crosscutting concerns relative to design patterns. The study shows that most aspect-oriented solutions provided improved separation of concerns. Some aspect-oriented solutions resulted in higher coupled components, more lines of code, and more complicated designs. Aspects provide a means to separate concerns by encapsulate them in modules, resembling some similarities with external feedback loops that encapsulate the functionality of adaptation concerns. However, the study in [24] focuses on the code level, while our study is concerned with the impact on design.

Finally, a recent experiment [25] evaluates the impact on the design of a set of higher level abstractions for modeling adaptive collaborations of service-based systems. The study is performed with Master-level students and compares the use of state of the art modeling abstractions for service-based systems with new proposed modeling abstractions, including

collaboration, role, interaction, and capability. The experiment results show that the use of these new abstractions can reduce fault density and design complexity, and improve reuse and productivity. Except for reuse, we studied similar properties related to design activities. However, the abstraction level and focus in both studies is different. [25] considers the design at the level of business processes and their interactions, while the study presented in this paper looks at design at the level of software components and their interactions. Furthermore, the focus of [25] is on the design of a service-based system using new modeling abstractions, while we focus on the impact of the design when using an external feedback loop to add adaptation properties to an existing design.

VII. Conclusions

In this paper, we investigated the question whether external feedback loops provide more effective engineering solutions for self-adaptation than internal mechanisms. The results of the study confirm common sense in the community of self-adaptive systems, by providing evidence that external feedback loops indeed provide more effective engineering solutions to self-adaptation, for the problems considered in the study.

The study yields the following insights. First, external MAPE loops do not reduce the average number of activities of the processes that are used to realize self-adaptation, compared to internal mechanisms. Thus there is no significant difference in the size of solutions created with both treatments. However, and more importantly, external MAPE loops do simplify the design in terms of control flow primitives for the processes. The significant reduction of control flow complexity increases understandability of the design, and can improve maintainability and testability of the system. Second, the use of external MAPE loops reduce fault density. Reduced fault density increases the quality and reliability of the software design, and add to customer satisfaction. It can also reduce the effort required for testing. Third, external MAPE loops realize a separation of concerns, which yields easier to understand designs, having a positive effect on productivity. Improved productivity can save the time and cost to develop software. Whereas we measured a significant increase of productivity in terms of supported functionality per time unit, we observed only a weak effect on effective productivity that measures the supported part of the overall design per time unit.

As with any empirical experiment, this study adds one piece of the puzzle towards obtaining a well-founded understanding of a topic of interest. What makes this study special is that it is (to the best of our knowledge) the first controlled experiment that investigates the effectiveness of engineering self-adaptive systems. We hope that our study will be an impetus towards more empirical studies that add pieces to the puzzle, guiding us towards a well-founded understanding of engineering self-adaptive systems.

References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, pp. 14:1–14:42, 2009.

[2] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, Oct. 2004.

[3] D. Weyns, S. Malek, and J. Andersson, "Forms: Unifying reference model for formal specification of distributed self-adaptive systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 7, no. 1, pp. 8:1–8:61, May 2012. [Online]. Available: http://doi.acm.org/10.1145/2168260.2168268

[4] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing: degrees, models, and applications," *ACM Computer Surveys*, vol. 40, pp. 7:1–7:28, 2008.

[5] B. H. Cheng *et al.*, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 1–26.

[6] R. Lemos *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 1–32.

[7] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.

[8] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[9] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," *Future of Software Engineering, International Conference on Software Engineering*, 2007.

[10] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley, 2004.

[11] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Commun. ACM*, vol. 32, no. 12, pp. 1415–1425, Dec. 1989.

[12] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed. PWS Publishing Co. USA, 1998.

[13] D. Garmus and D. Herron, *Function Point Analysis: Measurement Practices for Successful Software Projects*. Addison-Wesley Information Technology Series, 2001.

[14] D. Campbell and J. Stanley, *Experimental and Quasi-experimental Design for Research*. Rand McNally, 1963. [Online]. Available: http://books.google.se/books?id=3UfptgAACAAJ

[15] J. Cardoso, "Control-flow complexity measurement of processes and weyuker's properties," *Enformatika*, vol. 8, pp. 213–218, Oct. 2005.

[16] ——, "Process control-flow complexity metric: An empirical validation," in *Services Computing, 2006. SCC '06. IEEE International Conference on*, sept. 2006, pp. 167–173.

[17] A. Albrecht and J. Gaffney, J.E., "Software function, source lines of code, and development effort prediction: A software science validation," *Software Engineering, IEEE Transactions on*, vol. SE-9, no. 6, pp. 639 – 648, nov. 1983.

[18] J. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*, ser. Statistics Textbooks and Monographs. Marcel Dekker, 2003.

[19] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[20] M. Hollander and D. Wolfe, *Nonparametric statistical methods*, ser. Wiley series in probability and statistics: Texts and references section. Wiley, 1999.

[21] T. Cook and J. Stanley, *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Company, 1979.

[22] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *Software Engineering, IEEE Transactions on*, vol. 28, no. 8, pp. 721 – 734, aug 2002.

[23] L. Prechelt, B. Unger, W. Tichy, P. Brossler, and L. Votta, "A controlled experiment in maintenance: comparing design patterns to simpler solutions," *Software Engineering, IEEE Transactions on*, vol. 27, no. 12, pp. 1134 –1144, dec 2001.

[24] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa, "Modularizing design patterns with aspects: a quantitative study," in *Proceedings of the 4th international conference on Aspect-oriented software development*, ser. AOSD '05. New York, NY, USA: ACM, 2005, pp. 3–14.

[25] R. Haesevoets, D. Weyns, and T. Holvoet, "Architecture-centric support for adaptive service collaborations," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2013 (accepted) http://homepage.lnu.se/staff/daweaa/papers/2012TOSEM.pdf.