Risk Containers – A Help or Hindrance to Practitioners?

Risk Containers – A Help or Hindrance to Practitioners?

Andrew Leigh, Michel Wermelinger, Andrea Zisman
The Open University, United Kingdom
andrew.leigh@open.ac.uk, michel.wermelinger@open.ac.uk, andrea.zisman@open.ac.uk

Abstract—**Finding problems at the design stage reduces the cost to resolve them. Previous studies have indicated that error-proneness risks can be isolated into risk containers created from architectural designs, to help mitigate such risks early on. Here we describe an ongoing experiment to establish whether presenting designs as a collection of such containers helps practitioners manage the isolated risks. Participants must identify cyclic dependencies that could result in error-proneness and assess the impact of design changes. The emerging results suggest it takes participants longer to locate cyclic dependencies in collections of container diagrams than it does in a single diagram representing the whole design. Participants who reviewed collections of container diagrams tended to identify more cyclic dependencies correctly than those using a single diagram. Although, the results suggest that presenting a design as a collection of containers has no overall bearing on a participant's ability to correctly identify impacts of design changes, in cases where changes span multiple container diagrams no errors in change impact detection were observed. Errors were observed when assessing the same change using a single diagram representing the whole design.**

Keywords-analysis; architecture; risk; software

I. Introduction

Our motivation is to improve the chances of software development projects being more successful in terms of quality, cost and schedule. Risk containers are defined as architecture subsets that can be used to separate areas of low and high risk within an overall architecture. The work in [1][2] suggests that risk containers based on design rules support the isolation of error-proneness risks at design time. The ability to isolate risks does not mitigate the risk by itself. To improve chances of project success, practitioners must be able to work with the risk container to manage the isolated risks. We concluded that more work is necessary to *determine whether different types of risk containers are meaningful to practitioners* [1].

We also suggested that risk containers would be meaningful to practitioners if they are 'understandable to both architects and developers so that architects can elicit risks and manage them during the implementation' [2, p. 295]. Therefore, testing whether practitioners can comprehend the contents of risk containers is not enough to determine whether they are meaningful.

To satisfy their purpose, risk containers must act as a useful device for mitigating risks and it must be possible to create them from design artefacts used in practice. Even if it were proven beyond all doubt that a type of risk container can isolate instances of a risk, the containers are only meaningful if practitioners can use them to take practical measures to manage the risks. In summary, risk containers are meaningful to practitioners if they can:

A. be used to isolate real project risks;
B. be formed from design artefacts used in practice;
C. help to better manage mitigations used in practice.

We investigated how well three different types of risk container can isolate error-proneness [1]. We observed that Design Rule (DR) Containers tend to have less sharing of classes and were more effective at isolating error-proneness risks than Use Case (UC) Containers. However, UC Containers are based on features that lend themselves to black box testing, which is a mitigation against error-proneness. Even if DR Containers are more effective containers of error-proneness than UC Containers, the latter could be superior overall if they are more helpful to practitioners for mitigating isolated risks. Clause C specifies that for risk containers to be meaningful, practitioners must be able to use them to better mitigate the isolated risks.

Our research [1][2] provides some evidence that DR Containers satisfy clauses A and B. In this paper, we investigate if containers also satisfy clause C. We present the emerging results of an **experiment**[1] **open until September 30, 2019**, to garner further participation from the community.

II. Related Work

The work in [1][2] describes DR Containers derived from the Design Rule Spaces (DRSpaces) proposed by Xiao et al. [3]. DRSpaces are graphs based on design rules (key interfaces) that split an architecture into independent modules. The vertices are related classes and the edges are the relationships between those related classes. Xiao et al. created DRSpaces by using a formal clustering algorithm called Design Rule Hierarchy (DRH) proposed in [4].

The efficacy of DRH for splitting implementations into error-proneness isolating subsets based on design rules has been previously tested [1][2][3]. DRSpaces [3] were extracted from source code using an implementation of DRH algorithm [4]. DR Containers [1][2] differ because they were manually populated from upfront Unified Modelling Language (UML) designs. In the latter case the design rule classes and subordinate related classes were manually identified from the UML diagrams using the same base rules.

These different approaches to applying DRH reflect different motivations. For example, in [3] the authors were interested in providing architectural insights into

[1] https://www.callforparticipants.com/study/8GYWX/analysis-of-uml-software-architectures

implementation source code; while in [1] and [2] we were interested in determining whether potential implementation risks can be identified in upfront designs.

Mo et al. [5] identified several common causes of error-proneness in the projects they analysed using DRSpaces: unstable interfaces, implicit cross module dependencies, unhealthy inheritance and cyclic dependencies.

Automation is a significant benefit of the approach in [3] because Titan can be run over any version of the implementation source code. This requires less effort to create the DRSpaces in the first place and allows DRSpaces to be easily rebuilt when source code changes are made. The ability to create DRSpaces from source code also means that the approach in [3] is open to all software development process models. In contrary, the approach in [1][2] requires an upfront UML class diagram, which may preclude it from some projects using agile methodologies [6], which favor "working software over comprehensive documentation".

The approach in [1][2] enables projects using upfront design to predict and identify error-proneness risks before they become a problem. This is advantageous because errors found during the design stage of software development are five to twelve times cheaper to correct when compared to errors found at the testing stage [7]. Another advantage of using upfront UML is programming language independence.

In [3], the authors tested one method of splitting the architecture. This means no evidence was provided of how effective the DRH algorithm is by comparison to other ways of splitting an architecture. In [1], we compared UC Containers and Resource Containers to DR Containers.

The work in [8] found that UML sequence diagrams were commonly used by half of the 171 UML users surveyed. In this case, UC Containers were selected because use cases are often represented by sequence diagrams. The UC Container is composed of each class shown on the use case sequence diagram [1].

Resource Containers were populated with classes dependent upon an external resource (database table, service, file, etc.) by seeding with the resource encapsulation class and recursively adding classes that depend upon the encapsulation class [1]. Resource Containers were selected to investigate the sensitivity to error-proneness in architectures due to changes in external resources. Sensitivity is expected due to the ripple effect [9]. If the resource changes it may require a change to the encapsulating class which may in turn require a change to a dependent class and so on. Omission of one of these changes could introduce a regression error.

There is some evidence to suggest that DRSpaces and DR containers created using the DRH algorithm can be used to identify and isolate error-proneness risks in software architectures [1][2][3][4][5]. However, none of the research to date has investigated whether the presentation of subsets of the architecture would help or hinder practitioners with finding and mitigating problems contained within such subsets. Addressing that gap is the purpose of this research.

III. DATA COLLECTION

We have designed an online experiment to test whether different types of risk containers (design subsets) are meaningful to practitioners. The experiment is based around UML designs for a fictitious pet food e-commerce software system. The UML design has been divided into an equal number of DR, UC and Resource Containers using the container population rules described in [1]. Each of these three sets of containers represents a different experimental test group. A 'no containers' control group is used in addition to the three risk container type groups. Participants have been recruited through the researcher's network of industrial and academic colleagues, as well as callforparticipants.com.

The fictitious design is composed of 48 classes. Control group participants are asked to reason about a single class diagram showing all of the classes. For each risk container group (DR, UC and Resource) participants must reason about a set of nine diagrams (one per container). On average each container diagram contains 8.33, 8.00 and 7.66 classes for DR, UC and Resource containers, respectively.

The experiment has three parts. In part one participants are asked questions about their practitioner and UML experience. These questions are designed to provide context to each participant's response. Part two of the experiment provides participants with an overview of the fictitious system and introduces an architecture review scenario. Participants are asked to play the role of peer reviewer in order to identify any cyclic dependencies and the impact of two change scenarios upon the design. Participants are provided with a reminder of what are cyclic dependencies, and how to identify them in class and sequence diagrams.

In part three participants browse UML diagram(s) representing each risk container in their randomly assigned group (or control). They are required to identify the pair of classes involved in 'planted' cyclic dependencies and the classes impacted by two change scenarios. Participants are shown class diagrams for the Control, DR and Resource Container groups. UC Container group participants are shown sequence diagrams. This reflects the types of diagrams used to create such risk containers [1].

Cyclic dependencies are used as an example of an error-proneness inducing design flaw [5] to test whether presenting the design as a collection of risk containers helps or hinders practitioners to identify such potential error inducing flaws. If the results show that participants identify more of the planted cyclic dependencies in one of the risk container groups than the others (or control) it would suggest that presenting the design with (or without) such containers is helping the participants to locate the flaws (isolated risks). This observation would support the view that risk containers are meaningful to practitioners using the definition provided in section I. The time taken to complete the experiment is also recorded to see if it varies between the different experimental groups. Two different timings were recorded for each participant: (i) the duration between first accessing and exiting the experimental platform, and (ii) the duration that each participant declared it took them to complete the experiment.

Wolfe and Horowitz explain that attention towards a search field of objects of interest is not random and is modulated by five factors [10]. They assert this is necessary because limits on visual processing make it impossible to recognize everything at once. These adaptations imply that the size and complexity of the search scene will influence how quickly an individual can locate an object of interest contained within it. We hypothesise that it should be easier for practitioners to locate flaws that are contained within the smaller single container diagrams than the larger control diagram that shows all classes.

The corollary is that it ought to be harder for practitioners to locate cyclic dependencies that are split across two container diagrams due to them representing a larger (and split) scene. Therefore, practitioners are asked to identify three single-container cyclic dependencies for each of the four groups. Participants assigned to the DR and UC Container groups are also asked to locate three cyclic dependencies that span multiple diagrams. It is impossible to ask that for Resource Containers. That is because they are populated with classes that are recursively dependent on the resource encapsulation class, i.e. if class A depends on class B and class B depends on the resource encapsulating class C, all three classes would be members of the resource container created from class C. This means that all classes with a direct or indirect dependency on class C will be shown in the same Resource Container diagram. It is also impossible to ask the control group about multiple-diagram dependencies because all classes are shown on a single diagram. This difference in the number of review questions means we must be mindful when comparing the time taken to complete the experiment between the different experimental groups.

Participants are also asked to assess the impact of potential changes. This is because if risk containers were to be adopted by practitioners to isolate risks, possible mitigations would include redesign. If containers are found to aid practitioners with design refactoring, as well as during the initial risk identification stage [1][2][3], it would provide further evidence that risk containers are indeed meaningful using the definition provided. Participants are asked to identify the classes that would have to be modified for one change that is isolated within a single container and a second change that spans multiple container diagrams for all three risk container groups. Control group participants are asked about both changes on the single class diagram.

Participants know they have identified a cyclic-dependency as soon as a pair of co-dependent classes have been found. For change impacts, participants must continue to search even when an impact has been found to see if there are further impacts of the change.

By asking participants in different groups the same or similar questions, we are testing whether the way a design is split into containers influences the ability of practitioners to identify error inducing flaws and change impacts. The rationale behind our experiment is that if containers improve the participant's performance when performing these tasks by comparison to those using the overall diagram, containers would satisfy clause C of the definition.

IV. EMERGING RESULTS

At the time of writing we have received submissions from 13 participants. Table I shows that participants have a median twenty years of industry experience and seven years of UML experience. Table I also shows that participants have typically worked on three projects using class diagrams, and two projects using sequence diagrams.

TABLE I. PARTICIPANT EXPERIENCE

N	Question	Min	Max	Median	Q_1	Q_3
13	Years of commerical experience	0.00	30.00	20.00	10.00	23.00
13	Years of using UML	0.00	20.00	7.00	5.00	15.00
13	Projects using Class Diagrams	0.00	30.00	3.00	2.00	3.00
13	Projects using Seqn. Diagrams	0.00	30.00	2.00	2.00	5.00

Owing to random assignment to experimental groups, we received four control, three DR, three UC and three Resource Container submissions to date. Table II shows how long it took participants in each group to answer review questions. The 'Containers' values are obtained by treating those nine submissions as a single group.

TABLE II. TIME TAKEN PER REVIEW QUESTION (MINUTES)

N	Group	Min	Max	Median	Q_1	Q_3
4	Control	3.00	19.60	6.50	3.75	11.65
9	Containers	5.63	18.00	8.63	7.50	11.25
3	DR Containers	7.50	11.25	7.50	7.50	9.38
3	Resource Containers	6.00	18.00	12.00	9.00	15.00
3	UC Containers	5.63	11.25	8.63	7.13	9.94

These results suggest it took participants in the container groups approximately 30% longer per review question than those in the Control group. It took DR Container participants only 15% longer. Since all types of container have approximately the same number of classes per diagram, these results suggest participants can answer review questions quicker for DR Containers than they can for Resource and UC Containers. Overall, the results suggest that presenting the design as a collection of container diagrams increases the time needed to complete the review.

The initial results for the accuracy with which participants can identify cyclic dependencies are shown in Table III. Note that there are no multiple diagram cyclic dependency questions for Resource Containers, as explained in section III.

TABLE III. % OF CYCLIC DPENDENCIES IDENTIFIED CORRECTLY

	N	Group	Min	Max	Median	Q_1	Q_3
Single Diagram	4	Control	33.33	100.00	33.33	33.33	50.00
	9	Containers	66.67	100.00	100.00	66.67	100.00
	3	DR Containers	66.67	66.67	66.67	66.67	66.67
	3	Resource Containers	100.00	100.00	100.00	100.00	100.00
	3	UC Containers	66.67	100.00	100.00	83.33	100.00
Multiple Diagrams	3	DR Containers	33.33	100.00	66.67	50.00	83.33
	3	UC Containers	0.00	100.00	33.33	16.67	66.67

The results in Table III are insufficient to render a compelling comparison between the container types. However, the combined 'Containers' results drawn from nine participants suggests that the smaller container diagrams do help practitioners identify cyclic dependencies contained within a single diagram. Participants in the container groups were typically two times more successful at identifying cyclic dependencies within single containers than participants in the control group. This is consistent with our hypothesis that it should be easier for participants to find these objects of interest in the smaller container diagrams.

The picture differs for multiple-container cyclic-dependencies because presenting the design as a collection of containers produces approximately the same number of correct answers as the control. This is expected because participants must memorize diagrams as they switch between them. Participants allocated to the DR Containers group found it easier to identify multiple-container cyclic dependencies than those allocated to the UC Containers group and the control group using a single large diagram.

Table IV shows how many change impacts were correctly identified in the different experimental groups.

TABLE IV. % OF CHANGE IMPACTS IDENTIFIED CORRECTLY

	N	Group	Min	Max	Median	Q_1	Q_3
Single Diagram	4	Control	50.00	100.00	100.00	87.50	100.00
	9	Containers	0.00	100.00	100.00	100.00	100.00
	3	DR Containers	0.00	100.00	100.00	50.00	100.00
	3	Resource Containers	100.00	100.00	100.00	100.00	100.00
	3	UC Containers	0.00	100.00	100.00	50.00	100.00
Multiple Diagrams	3	DR Containers	100.00	100.00	100.00	100.00	100.00
	3	UC Containers	100.00	100.00	100.00	100.00	100.00

The median values indicate that containers have no bearing on the practitioner's ability to detect change impacts over the control group. This is expected because participants must continue to search the remainder of the design even when an impact has been found to see if there are further impacts of the change. However, not a single error was observed in cases where participants were asked to impact changes spanning multiple container diagrams. Errors were observed when participants had to impact the same change on a single diagram. Cycling through smaller diagrams may help participants identify individual impacts more easily.

V. VALIDITY

With only 13 participants it is too early to determine whether the trends are significant and consequential. Participants were not timed under exam conditions and so it is unknown whether time taken was unduly influenced by disruptions. There is no reason to think that one group would be more disrupted than another and so this threat would be mitigated by more participants. We used just one error inducing flaw (cyclic-dependency) and so the results may not translate to other error inducing flaws such as implicit cross module dependencies and unhealthy inheritance [5]. Further work would be necessary to determine whether the results obtained using a relatively small toy architecture are generalizable to larger real architectures. The popularity of agile methods and limited use of UML in practice [11] may suggest the technique is only beneficial to some kinds of projects. As we have only tested the three ways of clustering architecture elements described in [1] more effective ways of presenting an architecture may remain to be determined.

VI. CONCLUSION

Previous studies indicate error-proneness risks can be isolated into containers [1][2]. This paper presents preliminary results of an ongoing experiment (see URL on the first page) to determine whether presenting designs as risk containers helps practitioners manage isolated risks. Participants must identify cyclic dependencies that could result in error-proneness risks and assess the impact of design changes. The results so far suggest all 3 container types are meaningful to practitioners.

In particular, analyzing a design as a collection of containers leads to more accurate detection of cyclic dependencies isolated within smaller container diagrams. A general improvement was not observed for cyclic dependencies spanning multiple container diagrams, but an improvement for DR Containers was observed.

As for identifying the impact of design changes, participants made no errors if changes spanned multiple container diagrams, but some participants assessing the same change on a single diagram made errors.

The trade-off for the accuracy improvements is that it takes longer to analyze a design presented as a collection of containers than one presented as a single class diagram.

REFERENCES

[1] A. Leigh, M. Wermelinger and A. Zisman, "Software Architecture Risk Containers," Proc. 11th European Conference on Software Architecture, Springer, 2017, pp. 171-179.

[2] A. Leigh, M. Wermelinger and A. Zisman, "An evaluation of design rule spaces as risk containers," Proc. 13th Working International Conference on Software Architecture, IEEE, 2016, pp. 295–298.

[3] L. Xiao, Y. Cai, R. Kazman, "Design Rule Spaces: A new form of architectural insight," Proc. 36th International Conference on Software Engineering, ACM, 2014, pp. 967-977.

[4] S. Wong, Y. Cai, G. Valetto, G. Simeonov, K. Sethi, "Design rule hierarchies and parallelism in software development tasks," Proc. 24th International Conf. on Automated Software Engineering, ACM, 2009, pp. 197–208.

[5] R. Mo, Y. Cai, R. Kazman, L. Xiao, "Hotspot patterns: The formal definition and automatic detection of architecture smells," Proc. 12th Working IEEE/IFIP Conf. on Software Architecture, IEEE, 2015, pp. 51-60.

[6] P. Hohl et al., "Back to the future: origins and directions of the 'Agile Manifesto' – views of the originators," Journal of Software Engineering Research and Development, vol. 6(15), Springer, 2018.

[7] K. Akingbehin, "A quantitative supplement to the definition of software quality," Proc. ACIS Int'l Conf. on Software Engineering Research, Management and Applications, IEEE, 2005, pp. 348-352.

[8] B. Dobing and J Parsons, "How UML is used," Communications of the ACM, vol. 49, May 2006, pp. 109-113.

[9] M. Lindvall, R.T. Tvedt, P. Costa, "An empirically-based process for software architecture evaluation," Empirical Software Engineering, vol. 8(1), Springer, 2003, pp. 83-108.

[10] J.M. Wolfe and T.S. Horowitz, "Five factors that guide attention in visual search," Nature Human Behaviour, vol. 1(3), March 2017.

[11] M. Petre, "UML in practice," Proc. International Conference on Software Engineering, IEEE, 2013, pp. 722-731.