The role of environmental assumptions in failures of DNA nanosystems

Thein Tun[1] Robyn Lutz[2] Brian Nakayama[2] Yijun Yu[1] Divita Mathur[2] Bashar Nuseibeh[1,3]
[1]The Open University, UK {t.t.tun,y.yu,b.nuseibeh}@open.ac.uk
[2]Iowa State University, USA {rlutz,nakayama,divita}@iastate.edu
[3]Lero, Ireland bashar.nuseibeh@lero.ie

***Abstract*—Many failures arise from complex and imperfectly understood interactions of a computational system with aspects of the environment in which it operates. By environment we mean the computational system's broader context, also called the problem world. In this work, we propose a new analysis technique called failure frames, a variation of Jackson's problem frames, to identify and model classes of environmental assumptions whose violation is known from experience to have prevented the requirements from being satisfied. We use instances of failure frames, called failure diagrams, to make explicit in the requirements model the environmental assumptions that contributed to past failures. Developers want to reuse such knowledge of past failures to prevent failures in similar, new systems. We show that failure frames and failure diagrams can capture environmental assumptions that developers need to check in order to prevent recurrence of certain failures in similar application areas. The new failure frame approach that we describe arose from our work in molecular programming of DNA nanosystems. Inaccurate assumptions about the environment are a source of many failures in DNA nanosystems and can be extremely challenging to resolve. We describe the structure of a failure catalog for DNA nanosystems that we have prototyped for use by molecular programmers. We hypothesize that the failure frame approach and catalog can be broadly useful for reducing failure recurrence in other large, distributed applications with autonomous or nondeterministic behavior that must operate in uncertain environments.**

***Index Terms*—Failure Model, Environmental Assumptions, Reuse of Failure Knowledge, DNA Nanosystems**

I. Introduction

Many system failures arise from complex and imperfectly understood interactions of its software with aspects of the environment in which it operates. By *environment* we mean the software's broader context, or *the problem world* [1], such as the hardware on which it runs, concurrently executing software components, and physical surroundings. The potential for complex, unexpected interactions makes predicting the safety, reliability and performance of large distributed systems difficult [2], [3].

In this paper we focus specifically on those failures caused by unanticipated interactions between the software and the larger system context. In one class of failures, unwanted behavior originates from mistaken assumptions made about the environments. A well-known example comes from the aerospace software involved in a 1995 accident. The software design assumed that *plane wheels are turning if and only if the plane is moving on the runway* [4]. In fact, when the plane was aquaplaning on a water logged runway, the plane was moving but the wheels were not turning, a possibility that did not match the environmental assumption encoded in the software, and which it could not handle.

We report progress in this paper towards classifying and preventing faults involving incorrect or incomplete environmental assumptions that have historically led to failures. We propose a new analysis technique called *failure frames*, a variation of Jackson's *problem frames* [5]. Failure frames describe classes of problems in which requirements are violated due to mismatches between assumptions and certain system contexts. We introduce four failure frames, each capturing a particular kind of environmental assumption that leads to certain failures. An instance of a failure frame records a specific past failure and its context in order to make explicit the environmental assumptions that did not hold.

The failure frames allow us to reason about mismatches between assumptions that the software makes about the environment and the actual environmental reality. This approach has the advantage that it records both the incorrect assumption that enabled the failure and the proposed fix to prevent the failure or reduce its impact in the future.

The new failure frame technique that we describe arose from our recent work in a relatively new field of computation, that of molecular programming [6], [7]. Molecular programming is programming in the literal sense, in that DNA can be programmed to implement computational algorithms, and DNA tile self-assembly is Turing universal [8]. Molecular programming, such as programmable DNA self-assembly [9], provides a novel perspective on failures in very large systems. The nano-components that comprise such a system have autonomous, probabilistic behaviors and operate in a stochastic chemical context. The development of failure frames allowed us to classify environment/assumption mismatches in the nanosystems we studied into a small set of recurring patterns, and to record the fixes for reuse in future nanosystems. Reusing this knowledge is important to laboratory scientists because it reduces failures in new nanosystems, thereby improving their reliability while saving both labor and costs in laboratory experiments.

We hypothesize that the failure frame approach that we developed for molecular programming is more broadly useful.

We describe it here in the hopes that it can reduce the incidence of environmental assumption failures in other large, distributed applications with autonomous or non-deterministic behaviors and uncertain operating environments.

The main contribution of the paper is thus a systematic approach for identifying and classifying environmental assumptions from failure reports, in order that recurrence of similar failures are prevented in future. Supporting the approach is a structured catalog of past failures that can be queried in order to identify "fixes" to past problems. The purpose of the catalog is to reduce failures due to inaccurate environmental assumptions in future systems by making knowledge of past failures in similar contexts easy for developers to access and use.

The rest of the paper is organized as follows. Section II discuss the background notions of problem frames and environmental assumptions. Section III presents the proposed approach by discussing the failure frames, the structure of the failure catalog, and the development process supporting it together with its usage scenarios. Application of the proposed approach to molecular programming is illustrated and evaluated in Section IV. Discussion, conclusion and future work can be found in Section V.

II. Background

This section discusses two background concepts, namely, environmental assumptions and problem frames.

A. Environmental Assumptions

We define environmental assumptions to be statements about the software system's operational context that are accepted as true by the developers. This definition is consistent with common usage on projects and with dictionary definitions of "assumption." Researchers have used several terms to explain similar and related ideas. Zave and Jackson [1] note that "all statements made in the course of requirements engineering are statements about the environment", and that those statements are either *indicative* or *optative*. The former are statements about the environment that are true regardless of the existence or actions of a machine, whilst the latter are statements about the environment that the users wish to be true (requirements). A machine is the hardware-software component to be developed to solve the problem of interest. An environmental assumption in Jackson [4] is any statement about the problem world.

Van Lamsweerde [10] uses *domain property* to mean a descriptive statement about the environment that is expected to hold regardless of how the system behaves (e.g., the laws of physics). This is very similar to our definition of environmental assumption, and we use the terms interchangeably here. However, when working with molecular biologists, we only use the term "environmental assumption" since a "domain" to biologists means a standalone DNA sequence. (Note also that, in contrast to our definition, van Lamsweerde defines an assumption to be a statement to be satisfied by the environment, formulated in terms of environmental phenomena.)

The environmental assumptions of interest in this paper are indicative statements that are, or can become, invalid because they do not correspond well with the world phenomena. To illustrate how an environmental assumption can become invalid, we consider the following two indicative statements: "If a book copy is on loan with a borrower, then the book copy is not available for loan," and "All books on the shelves are available for loan." These statements become invalid when a borrower, having borrowed a book, puts it on the shelves without informing the librarian, and another borrower attempts to borrow it. We will say that indicative statements ("All books on the shelves are available for loan") describe *assumed properties*, and that the relevant world phenomena ("A borrower puts a loaned copy on the shelves without informing the librarian") have *observed properties*. Jackson calls the matching of formal terms to the world phenomena *designation* [11]. Designation is particularly challenging in large, complex systems with many parts that operate in uncertain or probabilistic environments. DNA nanosystems have these characteristics, as do many distributed networking systems, some mobile multi-agent systems, and planned fleets of autonomous nano-satellites. This leads us to suggest that the approach described here to identify and mitigate recurrence of unrealistic environmental assumptions in future DNA nanosystems may be of interest to developers of other complex systems executing in rich and imperfectly understood contexts.

B. Problem Frames

Jackson [5] has proposed a number of problem frames, and the *required behavior frame* (Fig. 1), for example, describes a class of problems where there is a need to build a control machine in order to control the behavior of some part of the physical world (Controlled Domain). The behavior of the control machine, in the context of the controlled domain, should satisfy the requirement (Required Behavior).

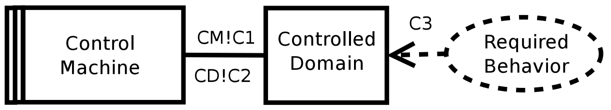

Fig. 1. Required Behavior Frame

This relationship is generally described as the entailment $W, S \vdash R$, where W, S and R stand for statements about the problem world, machine, and the requirements respectively.

When such a frame is instantiated in a problem diagram, for example, in the traffic light control application, the controlled domain is the light units, control machine is the lights controller and the required behavior is the desired light regime.

III. Proposed Approach

This section introduces the proposed approach by defining failure frames, the structure of the failure catalog, and the development process that underlies the approach.

A. Failure Frames

Failure in this work means the condition that a requirement is not satisfied. More precisely, failure is defined as "an event that occurs when the delivered service deviates from correct service" [12][3].

Like Jackson's *problem frames*, a failure frame shows the relationship among three artefacts: a requirement that was not satisfied, the problem world context (composed of problem world domains and shared phenomena between them) in which the requirement was not satisfied, and the machine involved. An instance of a failure frame is called a *failure diagram*.

Unlike in the problem frame, the challenge here is not to specify the machine properties needed to satisfy the requirement within the context. Rather, the challenge here is to specify the set of relevant properties for the world context in which a failure occurs, the requirement that was not met, and the machine involved. To indicate the fact that we are not interested in obtaining the machine specification, we do not use the machine symbol; instead, we indicate the machine component using the label "M".

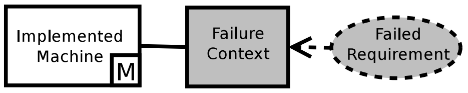

Fig. 2. Failure Frame

As with the traditional problem frames, we expect to show the relationship between the three artifacts using the entailment operator. However, the entailment for failure frames, unlike for problem frames, shows that the requirement is not satisfied, and therefore, that a failure has occurred. IM, FC, and FR refer to properties of the Implemented Machine, Failure Context and Failed Requirement, respectively in Fig. 2. Assuming that $FC \subset W$, and $FR \subset R$, the entailment for failure frames is:

$$IM, FC \nvdash FR$$

There are, thus, three main sources of failure:

1) The stated requirement FR is too idealistic ([13]);
2) The stated domain properties in FC do not match the observed domain properties;
3) The machine specification is incorrect.

In this work, we focus on the second of these in order to examine system failures caused by the mismatch between assumed and observed domain properties that have caused failures in the past.

In order to illustrate the mismatched assumptions, let us assume that the stated property of W is the propositional statement $a \rightarrow b$, the specification is the propositional statement a, and the requirement is another statement b. The entailment $\{a\}, \{a \rightarrow b\} \vdash \{b\}$ therefore holds.

However, in the failure context (FC) here, the world may turn out to have a much weaker property, for instance $a \rightarrow (b \vee c)$ (rather than $a \rightarrow b$ as stated in W). Here c is an environmental phenomenon that was not considered in the initial design. In this case, the failure will arise because $\{a\}, \{a \rightarrow (b \vee c)\}$ does not entail $\{b\}$.

When expected and observed domain properties differ, and as a result cause a failure, there are often many ways in which the error can be removed ("bug fixes"). First, the description W of the problem world may be modified to include the observed property. The machine specification S may also be modified, for instance, as $a \wedge \neg c$. In some cases, both W and S may be modified in order to prevent a failure. There are also cases where the requirement itself may be rewritten, e.g., as $\neg c \rightarrow b$, but we will not consider this possibility further in this work.

Therefore, methodologically, what a failure diagram provides is realistic domain properties that are found to exist in the world, which must be reflected in the assumptions made.

A failure point is a location in the context where an observed domain property differs from an assumed domain property, and as a result the system fails to meet its requirement. For instance, an observed domain property may be $a \rightarrow (b \vee c)$ while the property assumed is $a \rightarrow b$. On the basis of the kind of failure points involved, we propose the following failure frames:

- **Component Failure Frame**: In this frame, failure points can be linked to a specific problem world domain, or domain, where a single domain, including an implemented machine, has an observed property that is different from the assumed property.
- **Interaction Failure Frame**: In this frame, failure points are attached to interfaces between problem world domains that interact differently from the way it is assumed. This may be due to inappropriate abstraction of interface phenomena.
- **Compound Failure Frame**: In this frame, failure points are attached to several seemingly disparate domains and interfaces, where a particular combination of point failures has occurred. This reflects the well-known "Swiss cheese model" which postulates that certain critical failures occur due to alignment of a number of deviations from expected behaviors in several components of the systems [14].
- **Cumulative Failure Frame**: In this frame, failure points are failures that can be tolerated if their frequency or accrual is below a certain threshold. Unlike the three other kinds of failure, a single instance of violation of the requirements a single time may not be regarded as failure. Only when the same violation is repeated over time or in a population, does it then constitute a failure. Because of the stochastic behavior of molecular systems, many of the failures in DNA nanosystems are of this type. This frame can be seen as orthogonal to other failure frames.

We hypothesize that many of the Mandelbugs can be categorised using the last three frames.

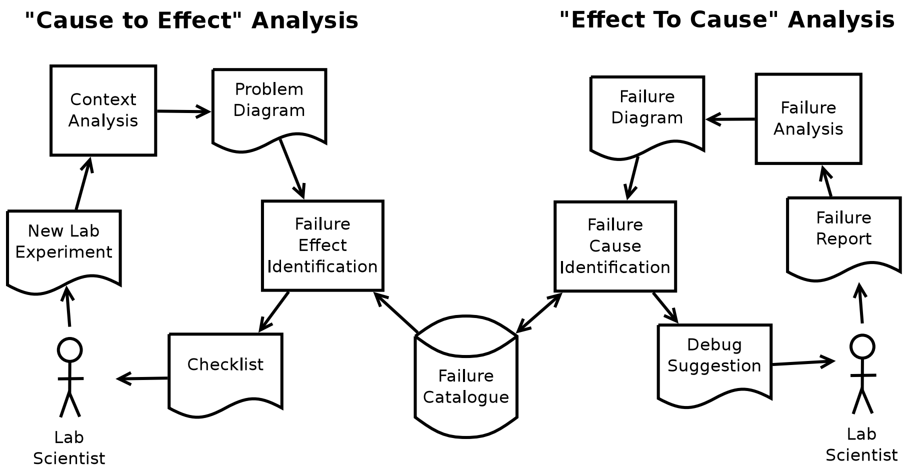

Fig. 3. Process Overview

B. Structure of catalog

To investigate the feasibility of our technique, we have manually populated a small catalog with failure frames pertaining to our area of study, DNA origami nanosystems (discussed in Section IV-A). The criteria for inclusion are as follows. Each failure frame in the catalog records (1) a significant failure, (2) described in the field's literature or laboratory notebooks, (3) that was caused in large part by a missing or incorrect assumption about the environment, and (4) that has recurred, or could feasibly recur, in similar, future nanosystems. Note that because molecular programming is a young field, many of the missing assumptions involve domain knowledge that did not exist at the time that the original failure occurred. Thus, inclusion of a failure in the catalog does not mean that the developer of the nanosystem in which the failure occurred was at fault, given the state of domain knowledge at the time. Rather, the effort is to disseminate the information about the problematic environmental assumption so that this failure will not recur.

Each failure diagram in the catalog contains the following fields: Assumed Properties, Affected System Goals, Failure Frame Type, Description, Origin, Cause, Detection, and Solution/Mitigation. *Assumed Properties* lists the domain properties and assumptions that were missing or contributed to the failure. *Affected System Goals* contains the requirements whose satisfaction is obstructed by the failure. The *Failure Frame Type* contains the type of the failure frame as introduced in Section III-A. The *Description* offers a short summary of the failure. *Origin* summarizes the literature while citing references for traceability. The *Cause* is similar to a failure mode in FMECA [15], as it describes how the domains or their interaction led to the failure. *Detection* identifies how the effect of the failure was observed, and the *Solution/Mitigation* field contains the "lessons learned" that one can reuse to prevent or mitigate the failure. The attributes Affected System Goals, Cause, and Detection make up the observed properties.

The catalog was populated with lessons learned from journal articles and the articles' often-lengthy published supplementary information (e.g., one article has 85 pages of supplementary information), as well as from lessons learned from one of the author's laboratory research in molecular biology. All failure frame entries also have traceability information to the published research describing the failed assumption and the proposed fix or mitigation strategy. The catalog currently contains 13 failure diagrams. 7 of these are of category (i.e., failure frame type) Component failures; 3 of these are of category Interaction failures; and 3 of these are of category Compound failures. Failure diagrams in any of the three categories can also be characterized as Cumulative failures if the failure only occurs beyond a certain threshold. 5 of the failure diagrams are also of category Cumulative failures.

To store the catalog, we used RQDA [16]. RQDA's coding function allows sources to be tagged with "codes" which one can later access through clicking. Thus, if one wanted to pull up all failure diagrams that affected the work of assembling DNA Origami, one would click "Assembly Work" to access those failure diagrams.

The catalog is available online for use by the molecular programming and DNA nanotechnology communities [17]. By making the repository available we hope to encourage other domain experts to add new failure diagrams or refine current ones. The catalog has been reviewed by a domain expert, but continued correctness of the repository requires ongoing review.

C. Process

The main input to our approach is a list of past failures. The main output is a checklist of conditions to check for a specific experimental setup or for a particular failure scenario, as shown in Figure 3.

There are two main processes in which the catalog can be used by others in this approach.

In the process to the right of the failure catalog, when a failure has occurred, perhaps due to invalid or changed environmental assumptions, the lab scientist wants to find out about possible cause(s) of the failure, as well as to record the failure for future reference. This process can be regarded as the debugging process and uses a form of "effect to cause" analysis. The process to the left of the failure catalog can be regarded as the failure prevention process. Here, the lab scientist is designing and implementing a new DNA nanosystem, and wishes to find out about failures attributed to invalid environmental assumptions that have been reported in similar systems, as well as to find out how to avoid them. This is a form of "cause to effect" ("what-if?") analysis. Elements of the figure are explained further below:

- *Failure Report*: A description of a failed lab experiment including the unwanted effect observed, the equipment used, the operations carried out, etc.
- *Failure Analysis*: Three aspects of the failure are identified from the failure report: the context of the failure, the requirement that has been violated, and the relevant environmental assumptions about the context. Towards this end, we draw a failure diagram.
- *Failure Diagram*: An instance of a failure frame showing the relationship between three artifacts: a requirement that was not satisfied, the problem world context in which the requirement was not satisfied, and the machine involved.
- *Failure Cause Identification*: The failure catalog is queried for instances of similar failures in the past, and their potential causes.
- *Debug Suggestion*: Known causes of similar failures found in the catalog, together with any recorded resolution, are provided as debug suggestions.
- *New Lab Experiment*: A description of the new experimental set-up which the lab scientist wishes to perform.
- *Context Analysis*: This step extracts the problem world domains, their assumed properties, and the requirements of the planned experiments.
- *Problem Diagram*: A Jackson Problem Diagram.
- *Failure Effect Identification*: The catalog is queried for past failures associated with the problem world domains and the requirements, looking for observed domain properties that were different from the assumed domain properties.
- *Checklist*: A list of domain properties that could cause certain known failures and possible ways to change the domain properties in order to avoid those failures.

IV. Application and Evaluation

This section discusses the key characteristics of DNA nanosystems, as well as the current development practices before presenting the initial evaluation results of our approach.

A. DNA Nanosystems Characteristics

The DNA nanosystems of interest are engineered products of a development process called molecular programming. The DNA nanosystems that we consider in this paper are programmed via the careful selection of short DNA strands (called staples) to self assemble a longer DNA strand (called a scaffold) into a required structure with required behaviors and required nonfunctional properties. Such DNA nanosystems are called DNA origami due to the folding the scaffold undergoes during its self-assembly [9], [18].

Molecular programming is, by its nature, probabilistic. Typically the number of instances of a nanosystem is very large, on the order of 10^{10} in a single drop of a chemical solution. DNA strands bind and unbind dynamically, but due to the physical characteristics of DNA, many failures will inevitably occur. Success is thus measured in probabilistic terms over a very large number of instances of the nanosystem. Often scientists will speak of yield; if "too many" instances fail, then the goal of the assemblage of instances of the nanosystem is a failure.

We illustrate the following discussion of the role of environmental assumptions in failures with an example of DNA nanopliers. This nanosystem is a bio-sensor that should detect the presence of a target molecule of concern within a solution (e.g., in a test tube). Our example is a simplified version of one previously developed by Kuzuya et al. [19], following the groundbreaking work of Yurke et al. [20]. While currently implemented only as a prototype in the laboratory setting, next-generation bio-sensor nanosystems will be deployed in cells, and eventually in the human body. Such safety-critical uses in future motivate our interest in identifying and better understanding potential failures and their causes.

Self-assembling the nanopliers. The mechanism that the DNA nanosystem uses to detect the target molecule is to program (by the choice of DNA strands) the self-assembly of a very large number of nanopliers. These nanopliers naturally self-assemble themselves mostly into the open position with each nanopliers' jaws open wide.

Sensing the target. In the execution stage, a solution containing a certain concentration of the target molecule is added to the test tube. Because of the way the nanopliers have been programmed to self-assemble with a target-specific ligand molecule lining their jaws, the intent is that most nanopliers will grasp a target molecule in their jaws. This grasping behavior pulls the jaws together, causing the nanopliers to change shape from open to closed (parallel) jaws.

Observing the result. The shape change of many nanopliers from open to closed jaws, caused by the presence of the target molecules, can be observed in a sample drawn from the test tube and imaged with an Atomic Force Microscope (AFM).

The kinds of failures that nanosystems experience are often caused by the chemical and physical environment in which the systems self-assemble and operate. Certain failures are thus linked to each of the phases of self-assembly (here, of the nanopliers), execution (here, sensing the target molecule), and observation (here, of whether the nanopliers jaws are open or close, i.e., whether they have captured the target molecule). For example, at self-assembly time, the short DNA strands selected to "staple" a long DNA strand into the right shape may unintentionally interact with each other, causing unplanned aggregation—essentially a blob. Another example of failure is that if the solution in which the DNA strands are to self-assemble is heated inadvertently to too high a temperature (over 75 °C), the bonds that hold the strands together will break, and the correct structure will fail to self-assemble. At execution time (runtime), a failure is that two target molecules together may keep a nanoplier from closing when it should, thus preventing the system from sensing the target's presence. At observation time (validation) a failure is that if the operator of the AFM observing the experimental results fails to count correctly the number of nanosystems displaying the intended behavior (here, the number of nanopliers moving from an initial open to a closed position), the result may be misreported. In that case the scientists will not be confident that the nanosystem reliably does what it is supposed to do. We thus focus on environmental failures that are significant enough that they have been reported to prevent a *threshold occurrence* [6] of the required behavior from occurring in previous nanosystems.

B. Current state of practice

Development of a new DNA nanosystem involves effort in the laboratory for the scientists as they repeatedly run experiments, the costs of the DNA and chemical supplies at each phase, and the expenses of the laboratory test environment and advanced imaging facilities. Failures increase the costs and effort by forcing lab scientists to redesign and re-implement their nanosystems.

The current state of practice is that laboratory scientists use failures as part of the process of elucidating incorrect or missing requirements, design errors, and invalid or missing environmental assumptions. Discovered failures are written in a laboratory notebook. Standards and formats for notebooks vary among researchers, laboratories, and fields. Notebooks are not routinely shared with other scientists. Documented protocols exist for many steps in the design, implementation, and imaging of DNA nanosystems. Zuccheri and Samori, for example, have edited a book of methods and protocols for DNA nanotechnology [21], and the Supplementary Information for journal articles may describe relevant protocols. New protocols often are shared informally, such as by being taught to visiting scientists.

The focus of almost all the protocols that we have examined, however, is on describing how to achieve the nominal case or the best yield. Rationales for steps in the process are often not provided, and explicit discussion of how to reduce risk of failure is largely absent. Similarly, the environmental assumptions that are made in support of the processes described are seldom discussed. Since many of the envisioned nanosystems will be safety-critical, e.g., DNA drug delivery devices or pollution monitoring, we see a need for greater attention to failures in order to achieve the required dependability. The failure-frame approach and the resulting catalog are efforts in this direction.

C. Evaluation

We have evaluated the techniques described in this paper in two directions. First, to evaluate the accuracy of the catalog, a domain expert in DNA origami nanosystems checked the entries in the catalog. The identified inaccuracies and missing information were discussed with us, researched further as needed, and corrected. We also updated the catalog entries as new information about the DNA origami environment was discovered by scientists and appeared in the literature.

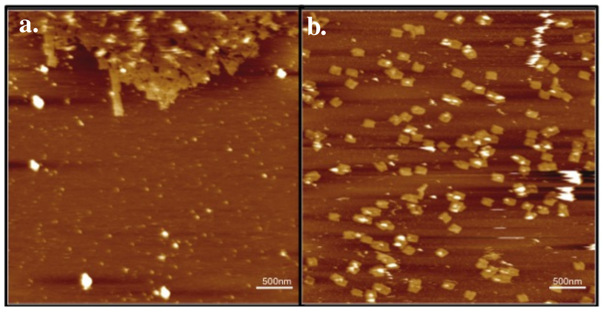

Fig. 4. AFM image of a DNA sensing nanosystem, a. with failure (aggregated origami), b. with failure corrected.

Second, to evaluate the usefulness of the catalog, we applied it retroactively to a real-world case study, a DNA origami sensing device under development locally. Our interest was in whether the effort and cost of that project to create a new DNA nanosystem could have been reduced if the catalog had been available at the time of development.

An early failure of the sensing nanosystem resulted from an unexpected interaction between adjacent DNA molecules, called stacking. The initial requirements for the nanosystem assumed that DNA only binds through Watson-Crick interactions (i.e., A with T and C with G). However, this assumption was invalid. In fact, researchers have learned in the past decade that the blunt ends of two helices of DNA can bind strongly to each other. In this case, observation with an AFM showed that the individual DNA components were aggregating by sticking to one another along their blunt ends (shown in Fig. 4(a)). The solution that was used successfully in this nanosystem was to instead select DNA sequences such that blunt ends were no longer present (shown in Fig. 4(b)). This interaction failure was subsequently entered in our catalog. To now find this failure in the catalog, a user would select a related phrase such as "inter-origami base stacking", "separation", or "independent origami" and a page describing the failure would display. The researcher estimated that preventing the failure in this case could have saved the project $2,000 in materials and 3 days of effort.

The second iteration of the sensing nanosystem also experienced an interaction failure, where the environmental assumption did not match the observed reality. The intended interaction between the DNA origami platform and the moving part of the origami device initially assumed that a 2-dimensional DNA platform maintained sufficient rigidity. The origami device had been programmed to emit a signal under certain conditions. However, no signal could be attained. Analysis showed that the origami platform had too much flexibility, allowing it to twist and disrupt the signal from the device. The solution that was used successfully was to use a more rigid (3-dimensional) origami structure. This interaction failure then was entered into the catalog, and to find it, a user would select “flexibility” or “2D origami.” The researcher estimated that preventing the failure in this case could have saved $3,000 in materials.

Subsequent iterations of the sensing nanosystem have brought additional functionality. As is to be expected in development of any novel system, significant time and money have been spent troubleshooting failures. What is noteworthy is that evidence from literature suggests other researchers already experienced many relevant failures, but that these failures were not widely known at the time of development. The failed assumptions and solutions that had been discovered by other researchers were either asides in articles focused on an experiment’s success or in the supporting information. As a result, significant effort and cost might have been saved if our current catalog had been available and used in this case. While these data are anecdotal, they tend to confirm our experience that preventing the recurrence of known failures in new systems continues to be challenging [22].

V. Discussion and Future Work

This paper has proposed a technique, failure frames, that are effective in structuring and cataloguing causes of failures involving invalid environmental assumptions. We envisage that community-based catalogs of structured information about failures will facilitate model-driven reuse of knowledge of past failures in many application areas. As an example, security failures in software systems tend to have complex causes, and failure frames might be applied to give additional perspective on contextual factors when analyzing them. For instance, the structuring of public security catalogs such as CAPEC [23] might be improved in this way to facilitate better knowledge reuse.

Although the need for structuring and cataloguing failures is clear, many questions remain as to how the catalog should be constructed and used. We have used natural language to describe properties, but more work is needed to determine whether that is an appropriate choice in general. Our initial interactions with lab scientists have suggested that the current software engineering techniques impose too high a barrier in terms of modeling language, terminology, and techniques when it comes to documenting and reusing knowledge about past failures. We are currently investigating ways to lower this barrier for adoption by finding a good balance between the usability of free text search and a more structured approach to knowledge reuse.

In conclusion, this paper argues that failure of DNA nanosystems should be studied from the perspective of requirements engineering, and in particular from the perspective of knowledge reuse. Effective methods can benefit the engineering of both DNA nanosystems and software systems alike.

Acknowledgment

Many colleagues have helped us shape the ideas in this paper. We especially thank Michael Jackson for valuable feedback on an earlier draft of the paper. This research was supported by NSF Grant 1247051 and by ERC Advanced Grant 291652. Part of this work was performed while the second author was on sabbatical at The Open University.

References

[1] P. Zave and M. Jackson, “Four dark corners of requirements engineering,” *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, pp. 1–30, Jan. 1997.

[2] J. Alonso, M. Grottke, A. P. Nikora, and K. S. Trivedi, “An empirical investigation of fault repairs and mitigations in space mission system software,” in *Int. Conf. on Dependable Sys. and Networks (DSN), Budapest, Hungary, June 24-27, 2013.* IEEE, 2013, pp. 1–8.

[3] J. Knight, *Fundamentals of Dependable Computing for Software Engineers*, 1st ed. Chapman & Hall/CRC, 2012.

[4] A. van Lamsweerde, “From worlds to machines,” in *Software Requirements and Design: The Work of Michael Jackson*, B. Nuseibeh and P. Zave, Eds. Lulu Press, 2009.

[5] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Boston, MA, USA: Addison-Wesley Longman, 2001.

[6] R. R. Lutz, J. H. Lutz, J. I. Lathrop, T. Klinge, D. Mathur, D. M. Stull, T. Bergquist, and E. Henderson, “Requirements analysis for a product family of DNA nanodevices,” in *RE*, M. P. E. Heimdahl and P. Sawyer, Eds. IEEE, 2012, pp. 211–220.

[7] S. J. Ellis, E. R. Henderson, T. H. Klinge, J. I. Lathrop, J. H. Lutz, R. R. Lutz, D. Mathur, and A. S. Miner, “Automated requirements analysis for a molecular watchdog timer,” in *ACM/IEEE Int. Conf. on Aut. Softw. Eng., ASE*, 2014, pp. 767–778.

[8] D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods, “The tile assembly model is intrinsically universal,” in *53rd Annual IEEE Symp. on Found. of Comp. Sci., FOCS*. IEEE Computer Society, 2012, pp. 302–310.

[9] P. W. K. Rothemund, “Folding DNA to create nanoscale shapes and patterns,” *Nature*, vol. 440, pp. 297–302, 2006/03/16/print.

[10] A. van Lamsweerde, *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[11] M. Jackson, *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

[12] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[13] A. van Lamsweerde and E. Letier, “Integrating obstacles in goal-driven requirements engineering,” in *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, Apr 1998, pp. 53–62.

[14] J. Reason, “Human error: models and management,” *BMJ*, vol. 320, no. 7237, pp. 768–770, 3 2000.

[15] N. G. Leveson, *Safeware: System Safety and Computers*. New York, NY, USA: ACM, 1995.

[16] R. Huang, “Rqda: R-based qualitative data analysis. r package version 0.2-3,” 2012. [Online]. Available: http://rqda.r-forge.r-project.org/

[17] B. Nakayama, “RQDA catalog of DNA origami failure diagrams,” http://www.cs.iastate.edu/ rlutz/cat, 2015.

[18] C. E. Castro, F. Kilchherr, D.-N. Kim, E. L. Shiao, T. Wauer, P. Wortmann, M. Bathe, and H. Dietz, “A primer to scaffolded DNA origami,” *Nature Methods*, vol. 28, no. 3, pp. 221–229, March 2011.

[19] A. Kuzuya, Y. Sakai, T. Yamazaki, Y. Xu, and M. Komiyama, “Nanomechanical DNA origami ‘single-molecule beacons’ directly imaged by atomic force microscopy.” *Nat Commun*, vol. 2, 2011.

[20] B. Yurke, A. J. Turberfield, A. P. Mills, F. C. Simmel, and J. L. Neumann, “A DNA-fuelled molecular machine made of DNA,” *Nature*, vol. 406, no. 6796, pp. 605–608, Aug. 2000.

[21] A. Zuccheri and B. Samori, Eds., *DNA Nanotechnology, Methods and Protocols*. Humana Press, 2011.

[22] R. Lutz, M. Lavin, J. Lux, K. Peters, and N. Rouquette, “Mining requirements knowledge from operational experience,” in *Managing Requirements Knowledge*, W. Maalej and A. K. Thurimella, Eds. Springer Berlin Heidelberg, 2013, pp. 49–73.

[23] MITRE, “Common attack pattern enumeration and classification,” 2015. [Online]. Available: https://capec.mitre.org/