Are Your Lights Off? Using Problem Frames to Diagnose System Failures*

Thein Than Tun[1,2] Michael Jackson[2] Robin Laney[2] Bashar Nuseibeh[2] Yijun Yu[2]
[1]PReCISE Research Centre, Faculty of Computer Science, University of Namur, Belgium
[2]Department of Computing, The Open University, UK
ttu@info.fundp.ac.be {m.jackson, r.c.laney, b.nuseibeh, y.yu}@open.ac.uk

Abstract

This paper reports on our experience of investigating the role of software systems in the power blackout that affected parts of the United States and Canada on 14 August 2003. Based on a detailed study of the official report on the blackout, our investigation has aimed to bring out requirements engineering lessons that can inform development practices for dependable software systems. Since the causes of failures are typically rooted in the complex structures of software systems and their world contexts, we have deployed and evaluated a framework that looks beyond the scope of software and into its physical context, directing attention to places in the system structures where failures are likely to occur. We report that (i) Problem Frames were effective in diagnosing the causes of failures and documenting the causes in a schematic and accessible way, and (ii) errors in addressing the concerns of biddable domains, model building problems, and monitoring problems had contributed to the blackout.

1 Introduction

In mature branches of engineering, failures and "the role played by reaction to and anticipation of failure" are regarded as essential for achieving design success [11]. Identification of the causes of past system failures, organisation and documentation of them in a way accessible by engineers within an engineering community, and application of knowledge of failures when designing future systems, all play a central role in establishing "normal design" practices [15]. Although there have been several excellent reports on high-profile system failures involving software systems [5, 7, 9], development practices for dependable systems have not exploited input from incident or accident investigations in a systematic way [2]. This work is a small step in the direction to address the gap.

Requirements Engineering (RE) is concerned with defining the behaviour of required systems, and any error introduced or prevented early in the development significantly contributes to the system dependability. In this respect, RE has a valuable role to play in systematising and documenting causes of past failures, and utilising this systematised knowledge in the development of future systems. In the same way that system failures can be attributed to programming, design, and human/operational errors, it is possible to attribute certain failures to RE errors. RE errors may be due to missing requirements, incorrect assumptions about the problem context, weak formulation of requirements and unexpected interactions between requirements.

Although the broader context—such as the organisational settings, regulatory regimes and market forces—often plays an important role in failures, we deliberately focus on the role of the software system in its physical context in order to bring out clear lessons for requirements engineers. Therefore, a framework is needed for investigating failures, which looks beyond the scope of software and into its physical context, and directs attention to places in the system structures where failures are likely to occur.

In this paper, we report on our experience of using Problem Frames [4] to identify, organise and document knowledge about the causes of past system failures. In the Problem Frames framework, potential causes of failures—known as "concerns"—are named and associated with a specific pattern of problem structure, a style of problem composition, a type of problem world domain, the requirement and the specification. An instantiation of a pattern, for instance, will immediately raise the need to address certain concerns in the sys-

*The title is inspired by [3]. An extended version of this paper can be found in [13]. This research is supported by the EPSRC, UK and the CERUNA programme of the University of Namur. Helpful comments and suggestions by the anonymous reviewers are gratefully acknowledged.

tem structures. This is, in a sense, similar to generating "verification conditions" for a program in order to prove its correctness with respect to the specification [1]. In this case, concerns raised will have to be discharged by requirements engineers, perhaps in collaboration with other stakeholders.

The rest of the paper is organised as follows. Section 2 gives an overview of the power blackout case study, the methodology used in the investigation, and some of the key principles of Problem Frames. The role of the software systems in the blackout is described and analysed in Section 3. Related work is discussed in Section 4. Section 5 summarises the findings.

2 Preliminaries

This section provides an overview of our case study, the research methodology used to investigate the failures, the conceptual framework of Problem Frames, and the expected outcome of our study.

2.1 2003 US-Canada Electricity Blackout

The electricity blackout that occurred on 14 August, 2003 in large parts of the Midwest and Northeast United States and Ontario, Canada, affected around 50 million people, according to the official report by the U.S.–Canada Power System Outage Task Force [14]. The outage began around 16:00 EDT (Eastern Daylight Time), and power was not fully restored for several days in some parts of the United States. The effect of the outage could be seen in satellite images of North America, whilst financial losses reportedly ran into billions of US dollars. The official report concluded that "this blackout could have been prevented", and software failures leading to the operator's reliance on outdated information was identified as one of the two "most important causes" of the blackout [14, p. 46].

2.2 Methodology

Investigating real-life system failures is difficult not least because of the size and complexity of these systems and limited availability of verifiable information about the failures and the systems involved [5]. Even when it is possible to master these difficulties, it is still a challenge to locate exactly when in the development an error was introduced [10]. The official report makes clear that factors such as the sagging of power lines, overgrown trees, poor communication, and lack of personnel training all contributed to the blackout.

Since our interest was to learn RE lessons, our methodology for investigating failures examined the chain of events leading up to the failure, and isolated the role of software systems in the failure. We ascertained what the components of the system did, what they should have done, and how it would have been possible to identify the causes at the RE stage. Therefore, a framework was needed that allowed us to structure the potential causes of failures in a schematic way.

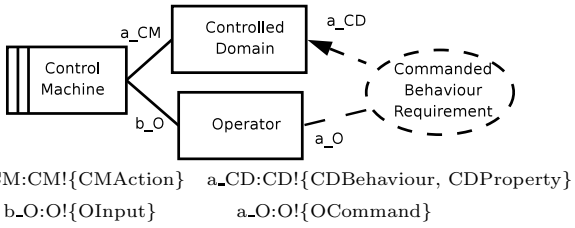


Figure 1. The Commanded Behaviour Frame

2.3 Problem Frames

The Problem Frames framework [4] is based on certain principles, four of which are relevant to the discussion. First, the framework encourages a systematic separation of descriptions into *requirements*, *problem world context* and *specifications*. For example, Figure 1 shows a high-level description of a type of software problem known as *Commanded Behaviour Frame*. In this problem, a software system, Control Machine, is required to apply control on a domain in the physical world, the Controlled Domain, according to the commands of a human agent, the Operator. Exactly how the Controlled Domain should behave, or what property it must have, when the Operator issues commands is described by the Commanded Behaviour Requirement. Therefore the *requirement* states the relationship between the operator command OCommand at the interface a_O, and the behaviour and property of the controlled domain CDBehaviour and CDProperty at the interface a_CD.

Description of the operator behaviour is concerned with the relationship between OInput at the interface b_O and OCommand at the interface a_O, namely what input the operator produces when a command is issued. Similarly, description of the Controlled Domain is concerned with the relationship between CMAction at the interface a_CM and CDBehaviour and CDProperty at the interface a_CD, namely what behaviour or property the controlled domain produces when machine actions are performed. The Operator and the Controlled Domain constitutes the *problem world context* of the Control Machine. The *specification*, description of the Control Machine, is concerned with the relationship between OInput at the interface b_O and CMAction at the interface a_CM, namely what actions the machine must perform when operator input is observed.

The operator may be a lift user and the controlled domain, a lift. The requirement will state how the lift should behave when the lift user issues commands. The

specification will state what operations the lift controller will perform when the operator input is received.

Second, this framework emphasises the need to understand the physical structure of the problem world context, and the behaviour of the domains involved. Third, the framework is based on recurring patterns of software problems, called frames. Each frame capture "concerns" of a certain type of software problems. For instance, the main concern of the "Commanded Behaviour" frame is to ensure that the system obeys the operator commands in imposing control on the behaviour of the system. An instantiation of a frame implies generation of certain conditions that need to be discharged.

Fourth, the framework provides a rich scheme for categorising and recording causes of failures. For instance, there are concerns specific to problem world domains, such as *reliability*, *identity* and *breakage*; there are frame concerns such as that of the required behaviour frame; and there are composition concerns such as *conflict*, *consistency* and *synchronisation*.

Therefore, we hypothesised that the Problem Frames framework provides an appropriate foundation for diagnosing failures involving software systems.

2.4 Expected Outcomes

There were two expected outcomes of this study. First, to establish whether Problem Frames are appropriate for investigating systems failures in terms of (i) locating causes of failure in the system structures, and (ii) recording them in a schematic way accessible by engineers within a community. Second, to identify causes of the blackout and either confirm them as known concerns or expand the repertoire of existing concerns by recording them schematically.

3 The Case Study

We now discuss two software-related failures that contributed significantly to the blackout. We briefly recount the chain of events leading to the blackout before discussing how Problem Frames were applied to diagnose the causes of failures and record the causes of failures.

3.1 Problem #1: State Estimator and Real Time Contingency Analysis

The infrastructure of the electric systems are large and complex, comprising many power generation stations, transformers, transmission lines, and individual and industrial customers. Providing reliable electricity through "real-time assessment, control and coordination of electricity production at thousands of generators, moving electricity across an interconnected network of transmission lines, and ultimately delivering the electricity to millions of customers" is a major technical challenge [14].

Reliability coordinators and control operators use complex monitoring systems to collect data about the status of the power network. In addition, they use a system called State Estimator (SE) to improve the accuracy of the collected data against the mathematical model of the power production and usage. When the divergence between the actual and predicted model of power production and usage is large, State Estimator will "produce a solution with a high mismatch". Information from the improved model is then used by various software tools, including Real Time Contingency Analysis (RTCA), to evaluate the reliability of the power system, and alert operators when necessary, for instance when the power production is critically low. This evaluation can be done periodically or on demand of the operator.

> "On August 14 at about 12:15 EDT, MISO's [Midwest Independent System Operator] state estimator produced a solution with a high mismatch [...] To troubleshoot this problem the analyst had turned off the automatic trigger that runs the state estimator every five minutes. After fixing the problem he forgot to re-enable it [...] Thinking the system had been successfully restored, the analyst went to lunch. The fact that the state estimator was not running automatically on its regular 5-minute schedule was discovered about 14:40 EDT."

When the automatic trigger was subsequently reenabled, the state estimator produced a solution with a high mismatch due to further developments on on the network. The official report assesses the situation as follows.

> "In summary, the MISO state estimator and real time contingency analysis tools were effectively out of service between 12:15 EDT and 16:04 EDT. This prevented MISO from promptly performing precontingency "early warning" assessments of power system reliability over the afternoon of August 14."

3.1.1 Problem Analysis

Based on this information, we constructed several problem diagrams to analyse relationships between the problem world domains mentioned in the description. Figure 2 shows a composite of two problem diagrams.

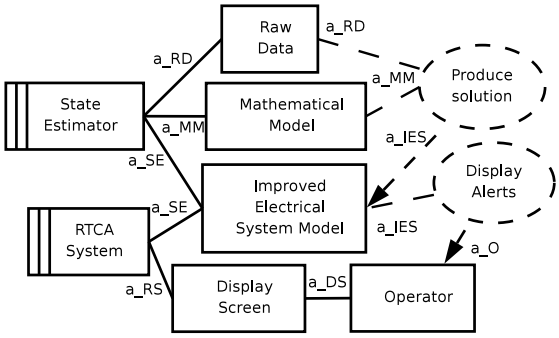


Figure 2. High Level Composite Problem Diagram of SE and RTCA systems

The problem of State Estimator is to produce RevisedData for the Improved Electrical System Model of the grid, based on StatusData, and Estimates produced by the Mathematical Model. In Problem Frames, this type of problem is known as a "model building problem". The problem of RTCA System is to examine RevisedData and raise appropriate alerts on the Display Screen used by the Operator. This type of problem is known as an "information display problem".

3.1.2 A Requirements Engineering Error?

On August 14, when the SE could not produce a consistent model, the operator turned off the automatic trigger of the SE in order to carry out maintenance work. Figure 3 shows the problem diagram, where the Maintenance Engineer uses the machine SE Trigger to turn on or turn off the State Estimator. This problem fits the *Commanded Behaviour Frame* shown in Figure 1. Part of the requirement here is to ensure that when the engineer issues the command OffNow, the SE should cease running.

When the maintenance work was done, the engineer forgot to re-enable the SE, leaving the electrical system model which the operators rely on, outdated. The resulting reliance by the operator on the outdated information was a significant contributing factor.

Clearly, the maintenance engineer should not have forgotten to re-engage the monitoring systems, and as a result, the problem would not have arisen. However, there is more to the problem than this being a "human error". Perhaps the fallibility of human operators should have been better recognised in the system's model of the world context.

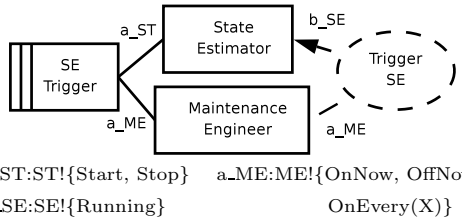


Figure 3. Manage SE Trigger

3.1.3 Naming and Categorising Concerns

A key part of the problem is the requirement that says that the operator commands always have precedence over the system actions. This requirement relies on the world assumption that the biddable domain—i.e., a human agent such as the maintenance engineer—always gives the correct commands. However, the Commanded Behaviour frame recognises that the operator is a biddable domain, whose behaviour is non-causal and may not be reliable. Therefore, the operator always giving the correct command may be too strong a condition to discharge. This gives rise to two concerns: one related to the biddable domain and the other, related to the Commanded Behaviour frame.

We will call the concern related to the biddable domain the *reminder concern*, which raises the following conditions to discharge: (i) Whenever the biddable domain overrides the system operations, which system domain(s) should be reminded about the override? (ii) How long should the override last? (iii) What happens when the length of time expires? In the case of the blackout, this may be translated into a requirement that says (i) whenever the SE has stopped, the system should remind the operator of the SE status and how long it has had that status, and (ii) at the end of a maintenance procedure, the system should remind the engineer of the SE status. Such a reminder could make the engineer's behaviour more reliable and perhaps could have helped prevent the failure.

A concern related to the Commanded Behaviour frame is whether the system should ignore the operator commands and take control of the system under certain circumstances. We will call this the *system precedence concern*. This may mean that the system should monitor the actions by the biddable domain, and intervene when the domain does not seem to be reliable. In that case, the requirement should be formulated as follows: Whenever maintenance work is thought to have been completed, the automatic trigger should be enabled.

Another key part of the problem is related to the issue of fault-tolerance in information display: What happens when the input the system receives from the

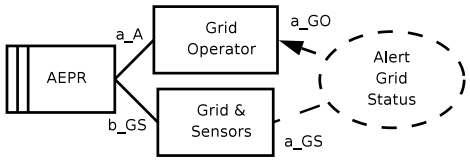
analogous model is unexpected? This may be due to an incorrect data type or an untimely input from the analogous model. We will call this the *outdated information concern*. Pertinent questions in this case are: 1) Can RTCA know that the Improved Electrical System Model is outdated? 2) What should it do about it? Had requirements engineers asked such questions, it could have led to a requirement such as "The Improved Electrical System Model must have a timestamp of when it was last updated successfully" and "If the Improved Electrical System Model is older than 30 minutes, the RTCA system should alert the operator that the electrical system model is now outdated". This will at least warn the operator not to rely on the information provided by the improved electrical system model.

3.2 Problem #2: Alarm and Event Processing Routine (AEPR) System

Another significant cause of the blackout was due, in part, to the Alarm and Event Processing Routine (AEPR) system, "a key software program that gives grid operators visual and audible indications of events occurring on their portion of the grid" [14].

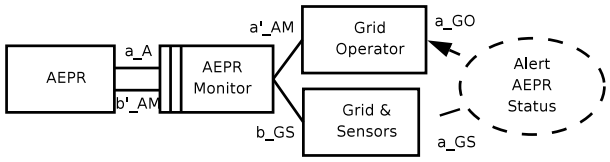
> "Alarms are a critical function of an EMS [Energy Management System], and EMS-generated alarms are the fundamental means by which system operators identify events on the power system that need their attention. If an EMS's alarms are absent, but operators are aware of the situation and the remainder of the EMS's functions are intact, the operators can potentially continue to use the EMS to monitor and exercise control of their power system. In the same way that an alarm system can inform operators about the failure of key grid facilities, it can also be set up to warn them if the alarm system itself fails to perform properly. FE's EMS did not have such a notification system."

The problem of alerting the Grid Operator of the grid status, ascertained from the Grid & Sensors is shown in Figure 4. This problem fits a type of problem known as the *Information Display Frame*. The requirement is to raise a separate alarm to the operator (`GOAlertedGrid`) if and only if there are events on the grid that threaten the system reliability (`GridOK`): ¬`GridOK` ↔ `GOAlertedGrid`. The specification of AEPR could be to raise an alert (`RaiseAlert`) if and only if danger is detected on the grid (`DangerDetected`): `DangerDetected` ↔ `RaiseAlert`. In the case study, the AEPR system



a_GO:GO!{GOAlertedGrid} a_A:A!{RaiseAlert}
b_GS:GS!{DangerDetected} a_GS:GS!{GridOK}

Figure 4. Alert Grid Status



b_GS:GS!{DangerDetected} a_A:A!{RaiseAlert}
b'_AM:AM!{DangerDetected} a_GS:GS!{GridOK}
a'_AM:AM!{RaiseAlert, RaiseSecondaryAlert} a_GO:GO!{GOAlertedGrid} GOAlertedAEPR}

Figure 5. Alert AEPR Status

failed silently, leading the operators to continue to rely on outdated information, and was one of "the most important causes" of the blackout.

3.2.1 A Requirements Engineering Error?

The official report is very clear about the fact that there was a missing requirement "to monitor the status of EMS and report it to the system operators." The British Standard 5839 on fire detection and fire alarm systems [12] is also concerned with monitoring systems, and anticipates such a requirement. Since fire alarms may fail when electricity is disconnected, the standard requires that alarms are fitted with a secondary independent source of power. In addition, when the source of power is switched from the primary to secondary source, the system should raise an alarm.

3.2.2 Naming and Categorising Concerns

The cause of this failure can be called a *silent failure of alarm systems*. Addressing this concern could raise questions such as: What happens if AEPR fails silently? Is it possible to detect such failures? What should be done when such failures are detected. This could have led the designers to the requirement that the system should monitor the behaviour of AEPR and raise an additional alarm when AEPR is thought to have failed. Figure 5 shows a problem diagram in which a wrapper intercepts the input to and output from the AEPR and when AEPR fails to respond as expected, a separate alarm is raised

(GOAlertedAEPR). The wrapper AEPR Monitor can pass on danger detection from the grid to AEPR (DangerDetected@b_GS ↔ DangerDetected@b'_AM) and pass on the alert trigger from AEPR to the grid operator (RaiseAlert@a_A ↔ RaiseAlert@a'_AM). Then the requirement to alert silent failure of AEPR is ¬GridOK ∧ ¬GOAlertedGrid ↔ GOAlertedAEPR. The specification for AEPR Monitor is DetectDanger@b_GS ∧ ¬RaiseAlert@a'_AM ↔ RaiseSecondaryAlert@a'_AM. An implementation of such a specification could have prevented the failure.

4 Related Work

There are many studies of software-related failures. Leveson, for instance, carried out several studies of software-related accidents, including those involving Therac-25 [7]. Johnson also has contributed an extensive literature on system accidents and incidents [5, 6, 2]. However, those studies of system failure of which we are aware have not been based on a clear conceptual structure for identifying, classifying, and recording the lessons learned at the level of detail appropriate for use by software engineers. For instance, the software engineering lessons Leveson and Turner [7] draw from the Therac-25 accidents include: "Documentation should not be an afterthought", and "Designs should be kept simple". Johnson investigated this power blackout in order to "sketch arguments for and against deregulation as a cause of the black-out" [6]. In this paper, we have applied a systematic approach to learning software engineering lessons, structured and described in ways that software engineers can relate to specifically.

Several variants of the Failure Modes and Effect Analysis (FMEA) method have been developed and applied in the development of dependable systems. Lutz and Woodhouse [8], for instance, applied a FMEA-based method to identify critical errors in requirements documents of two spacecraft systems. Our work is complementary to such methods, in the sense that we are concerned with identifying, structuring and documenting past software failures, which can then be used to narrow the search space in failure analysis.

5 Summary

Our experience of using Problem Frames to investigate system failures involving software systems showed that the framework of Problem Frames was appropriate for identifying causes of system failures and documenting the causes in a schematic and accessible way. The suggestion by the framework that requirements engineers should "look out" into the physical world, rather than "look into" the software was useful in directing and focusing the attention, because many of the causes of failures originated in the physical world context.

The separation of descriptions into requirements, problem world context and the specification enabled us to locate sources of failures in specific descriptions. Some failures were related to the requirements (such as missing requirements) and others to the problem world context (such as mismatch between the assumed and actual behaviour of the problem world domains). Furthermore, associating concerns to the requirement, problem world context, frame, domain type, style of composition, and the specifications provides a good basis for recording concerns in a schematic way.

In summary, specific lessons learnt from the blackout case study are: (i) a further specialisation of the reliability of the biddable domain, called the *reminder concern*, (ii) a further specialisation of the concern of the Commanded Behaviour frame where the system may have to take precedence over the operator action, called the *system precedence concern*, (iii) a further specialisation of the Information Display frame called the *outdated information concern*, and (iv) *the silent failure concern* related to the monitoring systems.

References

[1] J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security.* Addison-Wesley, 2003.

[2] S. Basnyat, N. Chozos, C. Johnson, and P. A. Palanque. Incident and accident investigation techniques to inform model-based design of safety-critical interactive systems. In *Proc. of DSV-IS*, pages 51–66, 2005.

[3] D. C. Gause and G. M. Weinberg. *Are Your Lights On?: How to Figure Out What the Problem Really Is.* Dorset House Publishing, New York, 1982.

[4] M. Jackson. *Problem Frames: Analyzing and structuring software development problems.* ACM Press & Addison Wesley, 2001.

[5] C. W. Johnson. *Failure in Safety Critical Systems: A Handbook of Accident and Incident Reporting.* University of Glasgow, 2003.

[6] C. W. Johnson. Public policy and the failure of national infrastructures. *Intl. J. of Emergency Management*, 2007.

[7] N. G. Leveson and C. S. Turner. Investigation of the therac-25 accidents. *IEEE Computer*, 26(7):18–41, 1993.

[8] R. R. Lutz and R. M. Woodhouse. Requirements analysis using forward and backward search. *Ann. Soft. Eng.*, 3:459–475, 1997.

[9] P. G. Neumann. Risks to the public in computers and related systems. *ACM SIGSOFT Soft. Eng. Notes*, 2003-2008.

[10] B. Nuseibeh. Ariane 5: Who dunnit? *IEEE Software*, 14(3):15–16, 1997.

[11] H. Petroski. *Success through Failure: The Paradox of Design.* Princeton University Press, 2006.

[12] TTS Fire & Security Ltd. BS5839: Part 1: 2002 Testing and Maintenance. Website, 2008. http://www.ttsfire.co.uk/bsdefinition.html.

[13] T. T. Tun, M. Jackson, R. Laney, B. Nuseibeh, and Y. Yu. Are your lights off? Using Problem Frames to diagnose system failures. Technical Report 2009/07, Department of Computing, The Open University, June 2009.

[14] U.S.-Canada Power System Outage Task Force. Final Report on the August 14th Blackout in the United States and Canada. Website, 2004. https://reports.energy.gov/.

[15] W. Vincenti. *What Engineers Know and How They Know It.* Johns Hopkins Press, Baltimore, 1990.