2017

Identifying Decision Patterns Using Monterey Phoenix

Quartuccio, John; Giammarco, Kristin; Auguston, Mikhail

Quartuccio, John, Kristin Giammarco, and Mikhail Auguston. "Identifying decision patterns using Monterey Phoenix." System of Systems Engineering Conference (SoSE), 2017 12th. IEEE, 2017.

https://hdl.handle.net/10945/59395

Identifying Decision Patterns Using Monterey Phoenix

John Quartuccio[a], Kristin Giammarco[b], and Mikhail Auguston[c]

***Abstract*—An automated means to identify behavior patterns would enable system developers to gain the benefit of past results in current and future system design. Analysis of decision patterns captures the interactions of human operators, systems, and their interface with the environment. Monterey Phoenix (MP) uses a formal language thereby exploiting the precision of lightweight formal methods as a means to model behaviors across domains, with applicability to the design of systems of systems, operational procedures, communication networks, business processes, and social interaction. This paper illustrates a concept of identifying patterns in a model of a decision process for a surgical procedure and an aviation emergency serving as case analyses. The results support a Bayesian belief network and an analysis of the decision structure. Finding embedded behavior patterns may be used as a means to catalog and classify intrinsic structures or architectures that are useful to system of system developers.**

***Keywords*—behavior patterns, pattern classification, model checking, system of systems, lightweight formal methods, small scope hypothesis.**

I. Introduction

The authors seek a means to catalog common or repeated behavior patterns in order to build a repository of validated architectures that can be used to address complex systems development. Pattern classification is an area of ongoing research [3, 11, 15], yet to the best of our knowledge, this is one of the first attempts to formalize decision pattern concept in order to establish a repository for the the purpose of reuse.

Monterey Phoenix (MP) defines activities and interactions of a decision sequence and it captures the interactions between the system and its environment, or among multiple systems shared in the environment. Behaviors are captured as a set of events with two basic relations: precedence and inclusion. The schema, or code, defines the structure of possible event traces including the event sequence, interactions, and other constraints; execution of the schema identifies all possible event traces within a defined scope [2]. This paper outlines the approach used to model a cross-domain example and to develop templates that describe the patterned behavior, dependent upon both the topology of the decision structure and the semantics of each decision. MP is available for anyone to use (http://firebird.nps.edu/).

II. Case Analysis

Syed [14] outlined two distinct cases, showing consistent decision-making patterns leading to failures, within a surgical procedure and an aviation mishap. The narratives are summarized as follows:

A. Medical procedure narrative

On March 29, 2005, a woman undergoes a routine surgical procedure. While under anesthesia, a rare side-effect occurs, causing a restriction in her airway. The surgeon immediately focuses on procedures to insert a breathing tube through her airway, but fails to succeed. The surgeon continues to focus on the airway tube, losing time. A nurse correctly observes the situation and retrieves a tracheotomy kit, that would enable an alternate airway and thereby alleviate the problem. The nurse recommends this solution to the surgeon, but this advice is not heeded by the surgeon. Time lapses, the patient loses oxygen, falls into a coma and eventually dies [14, 5].

B. Aviation mishap narrative

On December 28, 1978, United Airlines Flight 178 attempts a routine landing in Portland, Oregon. Upon descent, the pilot notices that the landing gear light does not give a positive indication that the gear is locked in position. The pilot immediately enters a holding pattern so that he can investigate the situation, attempting to understand the landing gear status, whether locked or unlocked. As time lapses, the flight engineer notices that the fuel level is reaching a critically low state. The flight engineer indicates the condition to the pilot, several times. The pilot fails to heed the engineer's recommendation until it is not possible to reach the runway. Engines fail upon approach, causing a crash landing, killing ten people, including the flight engineer [14, 4].

C. Decision patterns

As illustrated in Figure 1, consistency within the decision process becomes readily discernible. A hierarchical structure failed to support awareness of the truly critical scenario facing the participants. The leader, or decision maker, became overly focused on what he perceived to be the problem, but the conditions changed over time, causing a failure to reach the desired end-state. For this paper, a pattern is defined as a particular set of choices that leads an outcome or final state. The model expands these decisions to consider all possible choices, then adds interactions and constraints, thereby identifying relevant patterns.

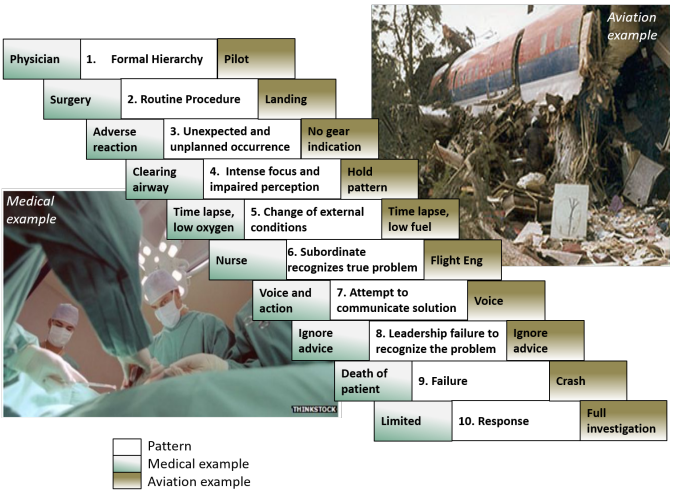

Fig. 1. Decision pattern, showing a consistent pattern for medical and aviation examples, derived from Syed [14], and documented by Fong [5], and the FAA lessons learned repository [4]. Photographs were downloaded from each associated reference.

III. Formal methods applied to model behaviors

Lightweight formal methods gain the advantage of applying a concise specification by implementing formal language in its description without incurring the cost of formal proofs [10, 9]. A formal

[a]student, Naval Postgraduate School, Systems Engineering Department
[b]Associate Professor, Naval Postgraduate School, Systems Engineering Department
[c]Associate Professor, Naval Postgraduate School, Computer Science Department
Manuscript submitted May 15, 2017

specification of the model is fundamental to enabling automated checking so that the developer can quickly investigate and change the model. A formal specification enables system definition at a higher level of abstraction, before investment in detailed design.

A. Monterey Phoenix (MP) model

Monterey Phoenix (MP) is a lightweight formal method built with the intent to model behaviors comprised of events and their interactions [1, 6]. Monterey Phoenix (MP) was inspired by the Small Scope Hypothesis and the associated Alloy Analyzer [7, 8] that models objects and their relations but not their behaviors. MP does model behaviors of a system architecture, and it enables the developer to interact with the model definition, add interactions, apply constraints, and adjust the definition as needed. Behavior in MP is modeled as a set of events, making up an event trace, where an event is an abstraction of any detectable activity, task, or component. Incorporating a concise language, MP employs event grammar rules. Within these grammar rules, the schema, or code, separates the definition of the behavior sequence from the definition of the interactions among the behaviors [2].

Two basic relations are defined for events: precedence and inclusion [2]. Precedence defines the ordering of events, enabling a description of structured order while also allowing concurrency. Inclusion defines the hierarchy of events, stemming from a root event that is logically distinct from other events and yet it may interact with other events. A root event is comprised of atomic events that are singular in description, or composite events that contain multiple atomic events and/or additional composite events. Interactions among behaviors allow sharing of an event and coordination of events.

The following sections outline the MP code that describe behaviors and interactions, along with constraints placed on the decision model applicable to the medical and aviation scenarios described in the previous narrative. Note that the MP code commands are shown in purple, the root events are shown as green, atomic events are shown as blue, and composite events are shown as orange.

B. MP applied to the decision pattern

Within this MP schema, root events include the leader (surgeon or pilot), the subordinate(s) (nurse or flight engineer), and the environment (problems associated with the routine surgery and landing). The routine procedure is shared among the leader, subordinate, and environment. The leader and subordinate have an independent perception that either correctly recognizes the environment or does not recognize the environment. The subordinate decides whether to communicate their individual perception to the leader, and the subordinate perception could be either correct or incorrect. The leader then has a context that receives input from a subordinate or rejects that input. The leader then makes either a correct or incorrect decision. Finally, the environment includes the occurrence of a problem, or no problem; and provides an indication of a successful or failed outcome.

1) Behavior definition in MP: Behaviors include the sequence of events associated with each root event, comprised of the leader, one or many subordinates, and the environment.

The leader is a root event, and the subsequent activities or events "belong to" that root event; thus an inclusion relationship is established.

This first segment of code is listed as follows:

```
ROOT Leader:
    Routine_procedure
    Perception
    Leadership_context
    Decision;
```

Each of the composite events are comprised of a series of decisions. The description of the perception of both the leader and all subordinates is such that each participant can either recognize the environment or not recognize the environment. The MP code code is listed below, where the "pipe" represents an alternative between the events on either side of it, as follows:

```
  Perception:
    ( Recognize_environment
      | Not_recognize_environment );
```

Next, the leader has two more decision points. 1) There exists some context within which the leader can either receive input from the subordinate or not receive input from the subordinate, and 2) the leader makes either a correct or incorrect decision. The MP code for these composite events, is listed as follows:

```
  Leadership_context:
    ( Receive_input
      | Not_receive_input );

  Decision:
    ( Correct_decision
      | Incorrect_decision );
```

Next, a separate root event is defined to include one or many subordinates. Each subordinate shares in the routine procedure, has an independent perception of the environment, and decides whether to communicate that perception to the leader.

The MP code for the subordinate segment designates one or many subordinates, using the {+...+} notation. This notation indicates that that the root "Subordinates" is actually a set of one or more "Subordinate" events. Each subordinate participates in the procedure, and has an independent perception and communication, as follows:

```
ROOT Subordinates:
  {+ Subordinate +};

  Subordinate:
    Routine_procedure
    Perception
    Communication;
```

In addition to the subordinate perception, as previously defined, each subordinate must also chose whether to communicate to the leader. The subordinate may be subject to intimidation, over-work, or a lack of training that could inhibit the decision of whether to communicate the perception with the leader. The MP code for the subordinates communication decision follows:

```
  Communication:
    ( Communicate_observation
      | Not_communicate_observation );
```

Finally, the MP code for the environment is as follows:

```
ROOT Environment:
    Routine_procedure
    Problem_state
    Outcome_state;
```

The environment consists of the problem state, either a problem or no problem; and the final outcome state, either successful or failed. The code for each follows:

```
  Problem_state:
    ( Problem
      | No_problem );

  Outcome_state:
    ( Successful_outcome
      | Failed_outcome );
```

2) Interaction definition in MP: The previous section described the behaviors of each scenario participant as independent root events, including inclusion and precedence of those events associated with each root (i.e the leader, all subordinates, and the environment). Since each root event is an independent behavior specification, this definition needs to be associated with interactions among the root events. The interaction definition captures basic logical dependencies among events in each root.

Interaction 1: Shared procedures.

First, let us consider that the routine procedure is shared among all participants and the environment. The code is to specify this fact formally is as follows:

```
Leader, Environment, Subordinates  SHARE ALL
  Routine_procedure;
```

Interaction 2: Leadership context dependency upon communication by the subordinate

Second, let us consider that the outcome of the leadership context must follow a decision by the subordinate on whether to communicate with the leader. A DO loop coordinates the dependency for each subordinate, and potential decision, while the ADD command adds a PRECEDES relation between each possible pair of events, as follows:

```
COORDINATE
  $a: Leadership_context FROM Leader
  DO COORDINATE
    $b: ( Communicate_observation
        | Not_communicate_observation ) FROM
            Subordinates
    DO ADD $b PRECEDES $a; OD;
  OD;
```

Interaction 3: A decision leads to an outcome.

The leadership decision leads to an outcome in the environment, listed as follows:

```
COORDINATE
  $a: Decision FROM Leader,
  $b: Outcome_state FROM Environment
DO ADD $a PRECEDES $b; OD;
```

3) Constraints imposed to the MP model: Logical constraints are necessary to impose on the model so that only reasonable results are derived. Using the ENSURE command, the following constraints were applied to the overall schema. In each case, the "#" character indicates the number of events (e.g. #A is total number of events A in the trace).

Constraint 1: If there is no problem in the environment, then always have a success.

```
ENSURE (#No_problem FROM Environment == 1 ->
          #Successful_outcome
              FROM Environment == 1);
```

Constraint 2: If all subordinates do not communicate, then the leader receives no input.

```
ENSURE(#Not_communicate_observation FROM
    Subordinates - #Subordinate == 0 ->
          #Not_receive_input FROM Leader == 1);
```

Constraint 3: If the leader recognizes the environment and does not receive input, then the leader makes a correct decision.

```
ENSURE(#Recognize_environment FROM Leader
          - #Not_receive_input == 0 ->
              #Correct_decision == 1);
```

Constraint 4: If the leader receives communication, the leader makes a correct decision, and its corollary.

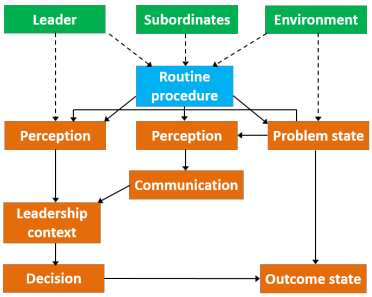

Fig. 2. Decision model derived in Monterey Phoenix (MP). Inclusion relationships are shown as dotted lines, precedence is shown as solid lines, root events are shown in green, atomic events shown in blue, and composite events are shown in orange in collapsed form so that included alternative events are not shown in this view.

```
ENSURE (#Receive_input FROM Leader == 1 ->
          #Correct_decision FROM Leader == 1);
ENSURE (#Not_receive_input FROM Leader == 1 ->
          #Incorrect_decision FROM Leader == 1);
```

Constraint 5: A correct decision leads to a successful outcome, and its corollary.

```
ENSURE (#Correct_decision FROM Leader == 1 ->
          #Successful_outcome FROM Environment
              ==1);
ENSURE (#Incorrect_decision FROM Leader == 1 ->
          #Failed_outcome FROM Environment ==1);
```

Figure 2 illustrates the resulting decision pattern construct, given the behavior definition, interactions, and constraints. Each composite event includes two alternatives, and each trace represents a possible combination of events. Running the model at scope one, without constraints, the seven decisions-events resulted in 128 traces ($2^7 = 128$); with the constraints, the model produced twelve traces.

Running the same model at scope two allows all previous traces plus those with two subordinates, now resulting in seventy-six traces, or combinations of events. Similarly, scope three results in 364 traces. The effect of scope is shown in Figure 15. Note that the number of valid possible traces that can be generated greatly exceeds the scenario variants discussed in the historical case studies, underscoring the value of using automated tools in analyzing case studies.

4) Templates describing behavior patterns: Notice that the topology of the model, shown in Figure 2, remains consistent, regardless of the decisions made by the leader and subordinates. Therefore, additional meaning, or semantics must be added in order to characterize templates for the derived behavior. Simple semantics were used to indicate a favorable or unfavorable condition.

```
Recognize_environment:          favorable;
Not_recognize_environment:      unfavorable;

Receive_input:                  Leader_favorable;
Not_receive_input:              Leader_unfavorable;

Correct_decision:               Leader_favorable;
Incorrect_decision:             Leader_unfavorable;

Communicate_observation:        Sub_favorable;
Not_communicate_observation:    Sub_unfavorable;

No_problem:                     Env_favorable;
Problem:                        Env_unfavorable;

Successful_outcome:             Env_favorable;
Failed_outcome:                 Env_unfavorable;
```

Now that the model definition is complete with behaviors, interactions, and constraints imposed, searching for templates becomes possible. All twelve potential outcomes for a single subordinate were generalized in a template similar to the one illustrated below, and are illustrated in Figures 3 through 14. Each of these figures illustrate a favorable condition as gray and an unfavorable condition as black. The code inspects each trace, and inspects for the presence of the specified behavior pattern using an "IF...THEN" statement. Each trace satisfying the condition is then marked using the "MARK" and "SAY" commands so that each trace associated with a template is readily identified. Comments in the code are listed as gray and bracketed by the "/*" and "*/" characters, while variables begin with the "$" character.

```
IF EXISTS DISJ
/* From leader */
      $L1: favorable,
      $L2: Leader_favorable,
      $L3: Leader_favorable,
/* From subordinates */
      $S1: favorable,
      $S2: Sub_favorable,
/* From environment */
      $E1: Env_unfavorable,
      $E2: Env_favorable
/* Impose ordering relationship on events */
  (   Leader CONTAINS $L1 AND
      $L1 BEFORE $L2 AND
      $L2 BEFORE $L3 AND
      $S1 BEFORE $S2 AND
      $S2 BEFORE $L2 AND
      $E1 BEFORE $E2 AND
      $L3 BEFORE $E2    )
THEN MARK; SAY("T1:  Both leader and subordinate
    see a problem"); FI;
```

The associated variable name definitions are as follows:

```
$L1: leader perception of the environment
$L2: leader context for communication
$L3: leader decision whether correct or incorrect
$S1: subordinate perception of the environment
$S2: subordinate communication to the leader
$E1: environment condition, problem or not
$E2: environment outcome, success or failure
```

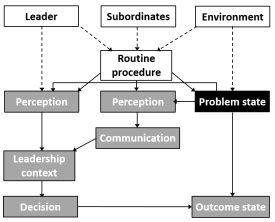

Fig. 3. Template 1 (T1): Both leader and subordinate perceive a real problem.

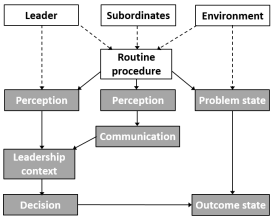

Fig. 4. Template 2 (T2): Both leader and subordinate perceive no problem.

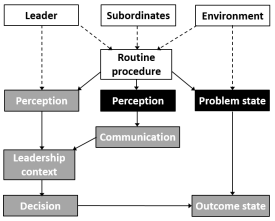

Fig. 5. Template 3 (T3): Subordinate fails to perceive a real problem.

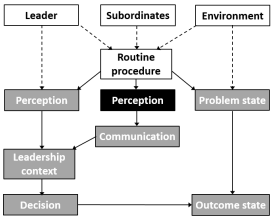

Fig. 6. Template 4 (T4): Subordinate perceives an unreal problem.

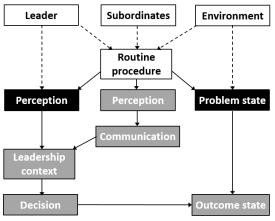

Fig. 7. Template 5 (T5): Leader listens to subordinate.

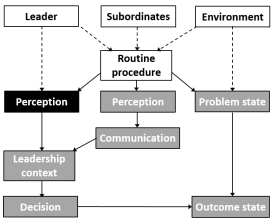

Fig. 8. Template 6 (T6): Leader perceived an unreal problem.

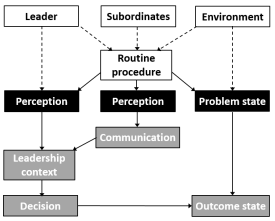

Fig. 9. Template 7 (T7): Incorrect perception, but correct decision.

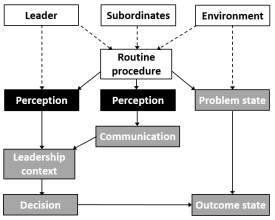

Fig. 10. Template 8 (T8): Incorrect perception, but no problem exists.

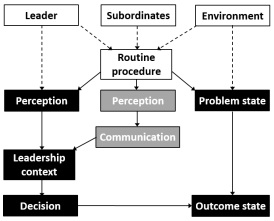

Fig. 11. Template 9 (T9): Leader fails to consider subordinate input.

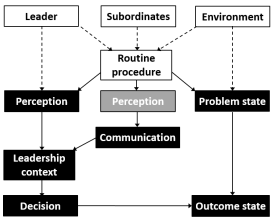

Fig. 12. Template 10 (T10): Subordinate fails to communicate a problem.

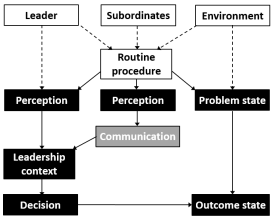

Fig. 13. Template 11 (T11): Subordinate communicates incorrect perception.

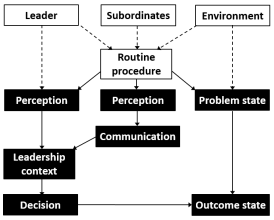

Fig. 14. Template 12 (T12): All conditions unfavorable.

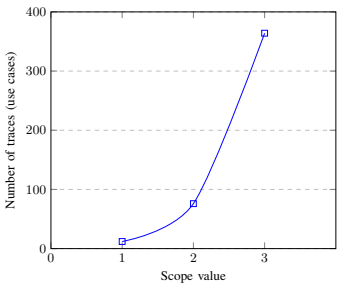

Fig. 15. Number of traces, or use case scenarios, for increased scope

IV. Discussion

A. Probability relationship to the decision model

MP has been demonstrated to support stochastic analysis of system architectures [13]. Particular to this paper, the model topology produced a directed graph. The authors propose further investigation of probability by applying a Bayesian belief network, as illustrated in Figure 16. Note that the structure of the belief network stems directly from the decision model. Additional constraints added direct relationships from the perception of the leader, L_1, to the leadership decision, L_3, indicated by blue dotted lines. An extensive description of associate probabilities was also documented [12].

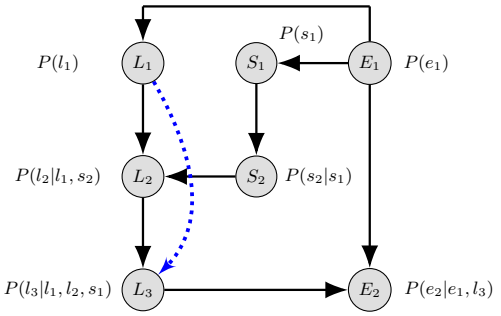

Fig. 16. Bayesian belief network derivation from the decision model, indicating conditional probabilities for each node.

B. Implications of the decision model

The decision model and the associated probability belief network provides insight into where changes can be made to the system, and a mathematical construct determining the impact of changes. Syed's main point in analyzing the failure method, was captured in Figure 11, where the leader fails to consider the input of a subordinate, then resulting in a failed and serious outcome [14]. The objective of the case study was to illustrate that the aviation community immediately responded by conducting an independent analysis of the failure, communicating these results to the aviation community, and instituting changes in training. These responses encouraged the subordinates to have effective communication in an emergency and changed the leader's willingness to receive information during the crisis. Conversely, the medical community has a history of not disclosing the details of a failed outcome, which leads to a greater likelihood of not changing the probabilities associated with patterns of behavior.

V. Conclusion

This study showed that lightweight formal methods can be used to automatically identify patterns of behavior in decision logic. The resulting model, formulated in Monterey Phoenix (MP), supports probability analysis and may form a basis for compiling a catalog of successfully validated architectures.

With respect to the the decision model for the surgical procedure and aviation mishap, both use cases were represented in MP, and the same template was recognized on both models following the proper assignment of event attributes.

This paper sets the framework for developing an automated means to search system architectures for the presence of consistent patterns in decision making, lending insight to reuse of behavior architecture models. Assigning consistent semantics, cataloging templates of interest, and formatting them for training and governance are critical aspects to resolve; however, the result provides a means to implement lessons learned in support of architecture design practice.

References

[1] Mikhail Auguston. Monterey Phoenix, or how to make software architecture executable. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming*

systems languages and applications, pages 1031–1040. ACM, 2009.

[2] Mikhail Auguston. *Monterey Phoenix, System and Software Architecture and Workflow Modeling Language Manual (version 2)*. Naval Postgraduate School, Monterey California, 2 edition, April 2016. https://wiki.nps.edu/display/MP/MP+Crash+Course.

[3] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2001.

[4] FAA. Lessons learned, United Airlines Flight 173, december 1978. published on-line, downloaded November 12, 2016. http://lessonslearned.faa.gov/ll_main.cfm?TabID=1&LLID=42.

[5] Kevin Fong. What we can learn from fatal mistakes in surgery. on-line, downloaded November 12, 2016, March 2013. http://www.bbc.com/news/health-21829540.

[6] Kristin Giammarco. A formal method for assessing architecture model and design maturity using domain-independent patterns. *Procedia Computer Science*, 28:555–564, 2014.

[7] Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, April 2002.

[8] Daniel Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[9] Daniel Jackson and Jeannette Wing. Lightweight formal methods. *ACM Comput. Surv.*, 28(4es):121, 1996.

[10] Cliff B. Jones. A rigorous approach to formal methods. *ACM Comput. Surv.*, 28(4es):121, 1996.

[11] Rudolf K Keller, Reinhard Schauer, Sébastien Robitaille, and Patrick Pagé. Pattern-based reverse-engineering of design components. In *Proceedings of the 21st international conference on Software engineering*, pages 226–235. ACM, 1999.

[12] John Quartuccio, Kristin Giammarco, and Mikhail Auguston. Deriving probabilities from behavior models defined in Monterey Phoenix. In *System of Systems Engineering (SoSE), 2017 12th International Conference on*, 2017.

[13] Songzheng Song, Jiexin Zhang, Yang Liu, Mikhail Auguston, Jun Sun, Jin Song Dong, and Tieming Chen. Formalizing and verifying stochastic system architectures using Monterey Phoenix. *Software & Systems Modeling*, 15(2):453–471, 2016.

[14] Matthew Syed. *Black Box Thinking: the surprising truth about success*. John Murray, 2015.

[15] Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In *International Conference on Web Information Systems Engineering*, pages 156–168. Springer, 2006.