Systematic Analysis of Faults and Errors

Sam Procter, John Hatcliff; Kansas State University

General Tips

1. These instructions are designed to be used with publically-available templates
2. A partially-worked example is also available
3. The templates often assume one element where there may be multiple. In nearly all cases, the analyst can simply add rows.
4. Spreadsheet cells are quite small, so...
   - Cells can be made multiline by setting Text-Wrapping to "Wrap"
   - Using the name as an index, fill in all notes in the "Explanations" Section
5. Reference cells can (and should) be actual references to keep the worksheet elements synchronized
6. When the term "element" is used, it signifies either a component or a connection between two components.
7. When component A is a *predecessor* of component B, A provides input to B. When component A is a *successor* of component B, A gets its input from B.
8. A *link* is the pathway between a component and a connection (or a connection and a component). It is infallible: any failures are considered to be part of either the source or destination element.

Activity 0: Fundamentals

Overview: In this step the analyst fills in basic information about the system, like its name, component pieces, and the problems that need to be avoided. This corresponds to the "Fundamentals" Chapter of Leveson's *Engineering a Safer World*.

0.1: System Fundamentals

Overview: Here, the analyst considers basic information about the system as a whole. In the second substep (described below) she will be directed to consider the individual elements of the system.

1. Identify the System
   1. Guide:
      - This is the name of the system you're considering.
      - Enter the name of the system:
        1. Row: System: (1)

        2. Column: (B)
    2. Example:
        - PCA Interlock
2. Identify Accident Levels
    1. Guide:
        - These are the levels of accidents we'll want to avoid.
        - Enter the name of the accident levels:
            1. Row: Accident levels (5)
            2. Column: Name (B).
        - The reference column will not be used.
        - Names are typically prefixed with "AL."
        - The term "Accident Levels" comes from Leveson (see, e.g., Section 7.1 of *Engineering a Safer World*), but corresponds well to similar notions of loss categorization from other domains:
            - Medical: Qualitative Severity Levels (ISO 14971, Section D.3.4.2)
            - Avionics: Consequences of Failure Conditions (FAA AC 25.1309-1A, Figure 1)
    2. Examples:
        1. AL.Death
        2. AL.Discomfort
3. Identify the System and Environment Elements
    - Research questions:
        - How reasonable is it to identify components without relationships at this point in the process?
    1. Guide
        - This is just a listing of the elements that are under the system designer's control and in the environment
        - Enter the names of the components, one per cell
            1. Column: System (J), starting on row 3
            2. Column: Environment (K), starting on row 3
        - Add more components in more rows as necessary
        - Though this step seems simple, there are actually two tasks being performed, both of which should be carefully considered:
            1. Determining the system boundary: Which components are going to be directly controlled by the system developer and which are not
            2. Determining the level of abstraction: What defines a "component" -- each component could (in all likelihood)

itself be considered a system, with its own (sub)components and environment.
    - There is no right answer, just be able to justify the choices that get made.
- Avoid the temptation to allocate the components to a control structure -- these lists will get modified in subsequent steps, and it will be easier to simply add / remove components without changing the architecture.

2. Examples
    1. System Components
        - PCA Pump
        - App Logic
        - App Display
        - Patient Sensors
    2. Environment Components
        - Patient

4. Identify Accidents
    1. Guide
        - These are the bad things that may happen. They should be traceable to the accident levels from 2.
        - Enter the name and associated accident level:
            1. Row: Accidents (7)
            2. Column: Name (B), Reference (C)
        - Names are typically prefaced with "ACC."
        - We use Leveson's terminology here as in step 2. Engineering a Safer World defines an Accident in section 7.1 as "An undesired or unplanned event that results in a loss, including loss of human life or human injury, property damage, environmental pollution, mission loss, etc."
        - Accidents are typically a pairing of an environmental component with an accident level. They should not speak to *how* the harm occurs (which is instead covered by a hazard).
            - We use the term in such a way that it is interchangeable with the term Mishap as defined in MIL-STD-882C and D.
    2. Examples
        1. ACC.Patient Dies (AL.Death)
        2. ACC.Patient is in Pain (AL.Discomfort)
5. Identify the System Hazards
    - Research questions
        - Is it possible to consistently identify hazardous component and environmental states at this point in the process?

1. Guide
    - These are ways that the accidents could happen. They should be traceable to accidents from step 3.
    - In addition to the standard name and reference, hazards also involve identifying:
        - A hazardous factor, which will be released in
        - A system state that, by a
        - System Component, that when combined with
        - The worst case state of an
        - Environmental component

      ... that will lead to the referenced accident.
    - There are equivalencies for some of these terms with, eg, Ericson's terminology (See pg 17 of Hazard Analysis Techniques for System Safety):

Our term	*Ericson's Term*
Hazardous Factor	Hazardous Element
System State	Initiating Mechanism
Environmental Component	Target / Threat

    - Enter the hazard and its information:
        1. Row: Hazards (9)
        2. Column: Name (B), Reference (C), Hazardous Factor (D), System Element (E), System Element State (F), Environment Element (G), Environment Element State (H), Manifestation (I)
    - Names are typically prefaced with "H."
2. Examples
    1. H.Patient Overdose (ACC.Patient Dies, Analgesic, PCA Pump, Pumping, Patient, Patient Cannot Tolerate More Analgesic, Improper Transmission)
    2. H.Patient Underdose (ACC.Patient is in Pain, Analgesic, PCA Pump, Not Pumping, Patient, Patient is in pain and can tolerate more analgesic, Delay / Drop)

6. Identify the System Safety Constraints
    1. Guide
        - These are constraints that, if they hold, guarantee the avoidance of the Hazards.
        - They are stated in nearly the same terms as the hazard, but with a minimal change that avoids the hazard (typically a different state of the element on the system boundary)
        - Enter the associated Safety Constraints
            1. Row: Safety Constraints (11)

2. Column: Name (B), Reference (C)
- Names are typically prefaced with "SC."
- Often there is a one-to-one correspondence of hazards to safety constraints (e.g., the safety-constraint is simply a negation of the hazard), but sometimes a number of constraints collectively or individually prevent a single hazard.
- Note that safety constraints are not simply safety requirements for the system, but rather high-level safety goals that will get discharged to the various components of the system as safety requirements on those components.
  - This generic nature makes it easier to catch less-traditional hazards, ie those resulting from things like: component interactions, degradation of the component over time, or the component's use by other actors as part of a larger process.

2. Examples
   1. SC.Dont Over Administer Analgesic (H.Patient Overdose, Analgesic, PCA Pump, Not Pumping, Patient, Near Harm)
   2. SC.Dont Under Administer Analgesic (H.Patient Underdose, Analgesic, PCA Pump, Pumping, Patient, Healthy but in pain)

7. (Optional) Determine the graphical candidate control structure
   1. Guide
      - This involves the allocation of system components to a structure which will allow each component to get the information it needs about the state of the controlled process to make safe decisions
      - This cannot be done in a spreadsheet format, though Google Spreadsheets allows insertion of diagrammatic drawings into sheets.
      - The control structure can be diagrammed as:
        - Components are drawn as boxes
        - Connections are directional connectors between components
        - System components and connections are drawn with solid lines
        - Environmental components and connections are drawn with dashed lines
      - Note that, in step 1.1.2, the components in this structure will be extended with process models (which cannot be determined at this point in the process)
        - Process models are drawn as solid boxes within components

        - Process variables are collections of process values, which are drawn text within a process model
    2. Examples
        - See the “Control Structure” sheet of the examples.

0.2: Component Fundamentals

Overview: For each element in the system, the analyst now creates a copy of the Element spreadsheet (which won't get filled out completely as part of this step) and fills in basic information. This is effectively the creation of a textual version of the system’s control structure.

1. Identify the element
    1. Guide
        - This is a name for the element under analysis
            - The first element examined should be the element closest to the system boundary (but still inside the system) as identified in Steps 4 and 7 of part 1.
            - Following the first element, the analyst should work backwards up the control structure (so, after examining an element, consider its immediate predecessor)
        - The name for should correspond to either:
            - One of the components inside the system boundary, or
            - A name for a connection between two components, one or both of which must be inside the system boundary
        - Enter a reference to the element:
            1. Row: Element (4+)
            2. Column: (A-B)
        - Note that we deterministically derive the element under analysis -- and examine all components in the control structure -- rather than manually choosing a control action, as in Leveson's *Engineering a Safer World*.
            - Other researchers, like Zahid H. Qureshi, have interpreted Leveson's methodology to involve three elements of the control structure (1. Controller errors, 2. Failure by the actuator to execute a control action, and 3. Bad feedback). In a similar spirit, we interpret Leveson's work as well.
    2. Examples
        - Component: PCA Pump
        - Connection: IVLine, PCA Pump --> Patient
2. Identify the successor link name
    1. Guide

- This is the link between this element and (one of) its successor(s).
- If the element has more than one successor link and plays a different role for those two links, *fill out a copy of the worksheet for each role/link pairing*
  - This step is necessary because a component playing multiple roles is essentially multiple components combined into one "box." The individual components should be considered individually.
- This is essentially the generic form of what Leveson refers to as "control actions" -- it's generic in that we do not restrict ourselves to links that carry control / command messages -- that are *leaving* the component.
- Enter the link name:
  1. Row: Successor Link Name: (4+)
  2. Column: (C-D)

2. Examples
   - Component: PCA Pump -> IV Line
   - Connection: IV Line -> Patient

3. Identify the predecessor link name(s)
   a. Guide
      - This is the set of links between this element and its predecessor (or, if there are multiple predecessor components, all of them). That is, these are the links over which the component's input arrives.
      - This is essentially the generic form of what Leveson refers to as "control actions" that are *entering* the component.
      - Enter the link name:
        1. Row: Predecessor Link Name: (4+)
        2. Column: (E-F)
   b. Examples
      - Component: AppLogicCommands -> PCA Pump
      - Connection: App Logic -> AppLogicCommands
4. Identify the element's classification(s)
   - Research questions:
     - What's a good set of component classifications?
     - What's a good set of connection classifications?

   1. Guide
      - This is the classification of the element according to the plays in the system.
      - Enter the classification

1. Row: Architectural (4),
2. Column: (H-I)

- Architectural classifications should be one of:
  - Sensor
  - Actuator
  - Controller
  - Controlled Process
- Connection architectural classification should be the classification of the source component and the destination component (eg, Sensor --> Controller)

2. Examples
   - Component: Actuator
   - Connection: Actuator --> Controlled Process

5. Repeat for the source of all predecessor links
   - By repeatedly applying Step 0.2 to all predecessor links, the analyst will work backwards through the control structure of the application.

Activity 1: Externally Caused Unsafe Interactions

1.1: Deriving the Successor Dangers

Overview: These are the things that can go wrong with the current element's immediate successor (ie, the component that is the destination of the Successor Link identified in 0.2-2). Our whole analysis of a given component will be to avoid these problems.

1. Pull in the Successor dangers
   1. Guide
      - These can typically be imported from the previous worksheet.
        - If this is the first element considered after the full system, then the successor dangers are simply violations of the system's safety constraints (Column B, Row ~11).
        - If this is the second or later element, the successor dangers are the manifestations of the successor component (Columns D-I, Row ~9)
      - Enter references to the dangers:
        1. Row: (13+)
        2. Column: Successor Dangers (A-B)
   2. Examples
      - Component: IVLine.Overinfusion

        - This would be a successor danger for the PCA Pump --- the pump's goal is to avoid the IV line's "overinfusion" danger, by not being in it's *pumping* state when the patient is in the *near harm* state.
    - Connection: H.PatientOverdose
        - This would be a successor danger for the IV Line. Since the line exists at the system boundary, its successor dangers are the system-level hazards.

2. (If Component) Document the Process Model
    1. Guide
        - A process model is a collection of process variables which are essentially collections of abstract states (termed process values) of the component relative to the notion(s) of danger identified in the previous step.
        - The control structure, created in step 0.1.7, should be updated to contain these process variables and their values.
        - Enter references to the dangers:
            1. Row: (17+)
            2. Column: Process Variable (A), Process Value (B-I)
        - Process models are required for controller components, but can be documented for sensors and actuators as well. This stems from the realization that most components are, at a lower level of abstraction, entire systems consisting of internal sensors, controllers, and actuators.
    2. Examples
        - Component: PCA Pump
            - Process Variable: Ticket Duration
            - Process Value: 1, …, 600
        - Connection: N/A

1.2: Deriving the Element's Dangers

Overview: Here the analyst considers if problems with the input to this component would cause problems with its output. This step is similar to, but much deeper than, STPA's Step 1.

Note also that, from this point on, the analysis of one element does not depend on the analysis of its predecessors -- ie, the analysis is compositional from here on out.

1. Select a Predecessor Link
    1. Guide
        - These are the links identified in 0.2-3 (Column E-F, Row 3+)
        - Create a reference to the selected link:

1. Row: (13+)
2. Column: Pred. Link (C)

2. Example
    - Component: AppLogicCommands -> PCA Pump
    - Connection: App Logic -> AppLogicCommands

2. Consider the four manifestations
    1. Guide
        - Here, the analyst should consider if it would be hazardous if the predecessor link exhibited any of the four manifestations
            - Note that we do not consider here if it's possible for the link to exhibit the given manifestations, only if their being exhibited would be hazardous
            - These manifestations are from Avizienis et-al's Basic Concepts and Taxonomy of Dependable and Secure Computing, where they are called Failure Domain's (see Fig. 8, pg 9)
        - The manifestations are:
            - Content -- ie, the value of messages on the link are incorrect, optionally divided further into:
                - High
                - Low
            - Halted -- ie, messages on the link stop arriving
            - Erratic -- ie, messages on the link appear out of the blue
            - Timing -- ie, messages on the link appear at the wrong time, typically divided further into
                - Early
                - Late
        - Record...
            - The result of the manifestation occurring --ie, a new danger-- (typically formatted as ComponentName.NameOfOccurrence)
            - Not Hazardous if messages on the link could not cause this hazard, or
            - Not Applicable if messages on the link could not exhibit this manifestation
            1. Row: (13+)
            2. Column: (D-I, as labelled)
3. Return to step 1.2-1 and repeat for all predecessor links identified in step 0.2-3

1.3 Examining the Externally Caused Dangers

Overview: Here the analyst explains how the successor dangers (identified in step 1.1) could be caused by bad input to the element (ie, the manifestations identified in step 1.2)

1. Select a Successor Danger
    1. Guide
        - The first thing an analyst needs to do is to select one of the successor dangers (identified in step 1.1, stored in column A-B, row ~9+)
        - Enter a reference to the danger:
            1. Row: (23+)
            2. Column: Successor Dangers (A)
        - Each successor danger may have more than one row -- this signifies that multiple errors in the current element will cause the same danger in the successor component
        - In some cases, more than one successor danger will occur simultaneously -- in this case, list all the successor dangers in the table cell.
    2. Examples
        - Component: IVLine.Overinfusion
        - Connection: H.PatientOverdose
2. Record the name of the danger
    1. Guide
        - Each previously-identified danger (from step 1.2-2) should show up at least once in this column
            - In general, though, there is a many-to-many mapping from successor dangers to externally caused dangers
        - Enter the name of the danger
            1. Row: (23+)
            2. Column: Name (B)
    2. Example
        - Component: PCA Pump.Spontaneously Give Drug
        - Connection: IV Line.Overadminister Drug
3. Identify the relevant process variable name and incorrect value
    1. Guide
        - Each component can be thought of as having a model of the controlled process
            - Sensors read the state of the controlled process directly

        - Controllers have a model of the controlled process provided by the sensors
        - Actuators get commands from controllers, which provides a (greatly reduced) view of the state of the controlled process
    - A mismatch between the controlled process state (identified in the previous step) and the component's process model lies at the root of every externally caused danger. This and the previous step combine to make that mismatch explicit.
    - Leveson's *Engineering a Safer World* gives an good primer on process models in Section 4.3 (pages 87-89)
    - Enter the process variable name and value
        1. Row: (23+)
        2. Column: Process Var. Name (C) and Value (D)
2. Example
    - Component: PatientHealth, Ok
4. Interpret the danger
    1. Guide
        - Since one guideword or manifestation can be interpreted in different ways, the analyst should now provide a concrete interpretation that explains how the danger name in column B causes the successor danger in column A
        - Enter the interpretation of the danger
            1. Row: (23+)
            2. Column: (E-F)
    2. Example
        - Component: The PCA pump receives a command to run even though it is unsafe to do so
        - Connection: There is more analgesic put into the IVLine than the patient can safely tolerate
5. Identify any Co-occurring Dangers
    1. Guide
        - Sometimes dangers will only manifest in the presence of other dangers -- this may be reflected in the natural language of the environmental state and / or interpretation, but should be made explicit here by referring to the other dangers (separated by commas) here.
            - Note that elements with only one predecessor link will not typically have co-occurring dangers

- Each cause of the danger will need its own entry. So, if two components (A and B) have to fail simultaneously for the danger to occur, four rows will need to be created:
  - Two which cause the successor danger:
    - A's failure will have one row in the table (with B's failure as a co-occurring danger)
    - B's failure will have its own subsequent row (with A's failure as a co-occurring danger)
  - Two which are not hazardous:
    - A's failure without a simultaneous failure of B
    - B's failure without a simultaneous failure of A
- Enter the name of co-occurring dangers, or "None" if not required
  1. Row: (23+)
  2. Column: Co-occurring Dangers, (G)
- Dangers are assumed to be combined via AND joins --- more complex relationships (OR, M-of-N, etc.) can be explained in the interpretation / global env. state columns.

2. Example
   - Component: N/A
   - Connection: N/A

6. Run-time Detection
   1. Guide
      - This allows an analyst to specify a mechanism to detect the occurrence of a danger at runtime
      - Avizienis et-al. state that there are two ways errors can be detected at runtime (see Fig. 16, page 16), either
        - Concurrently (as the element is performing its job), or
        - Preemptively (while the element is suspended for testing)
      - Record the mechanism (typically prefaced with either "Concurrent" or "Preemptive"), or "None" if run-time detection is impossible
        1. Row (23+)
        2. Column Run-time Detection (H)
   2. Example:
      - Component: None
      - Connection: Concurrent: Flow metering
7. Run-time Handling
   1. Guide
      - This allows an analyst to specify a mechanism to correct the occurrence of a danger at runtime

- Avizienis et-al. state that there are three ways errors can be handled at runtime (see Fig. 16, page 16),
  - Rollback (restoring the system to a saved state),
  - Rollforward (moving to a state without errors, ie a known-safe state), or
  - Compensation (use redundancy to mask the error)
- Record the mechanism of correction (typically prefaced with "Rollback", "Rollforward", or "Compensation"), or "None" if run-time compensation is impossible
  1. Row (23+)
  2. Column Run-time Handling (I)

2. Example:
   - Component: None
   - Connection: Rollforward: Stop Analgesic Flow

Step 2: Working with Internal Faults

Overview: These are the things that can go wrong with the element itself. There are 18 classes of faults, 15 of which come from Avizienis's work (they're a condensed form of the 31 fault classes from Fig. 5, page 6) and 3 come in response to problems with compositional verification.

Num.	Guideword	Possible Compensation	Description
1	Software Bug	Static Verification	Mistakes made in software creation
2	Bad Software Design		Poor choices made in software creation
3	Compromised Software	TPM-like + Chain-of-trust	Adversary tampers with software in development
4	Compromised Hardware	“Exotic” only	Adversary tampers with hardware in development
5	Hardware Bug		Mistakes made in hardware development
6	Bad Hardware Design		Poor choices made in hardware development
7	Production Defect		Hardware production defects (due to natural phenomena)
8	Deterioration	Periodic inspection	Internal hardware fault at runtime due to natural phenomena

9	Environment damages hardware	Shielding, ECC	Externally caused hardware fault at runtime due to natural phenomena
10	Operator HW Mistake	Thoughtful UI, Authorization, Access Control	Operator makes a mistake while interacting with hardware
11	Operator HW Wrong Choice	Thoughtful UI, Re-training, Authorization, Access Control	Operator makes a poor choice while interacting with hardware
12	Adversary Accesses Hardware	Physical Security, "Exotic"	Adversary tampers with hardware at runtime
13	Adversary Accesses Software	Access Control (Network and Local), Physical Security, TPM-like + Chain-of-Trust	Adversary tampers with software at runtime
14	Operator SW Mistake	Thoughtful UI, Authorization,Access Control	Operator makes a mistake while interacting with software
15	Operator SW Wrong Choice	Thoughtful UI, Re-training, Authorization, Access Control	Operator makes a poor choice while interacting with software
16	Syntax Mismatch		The current element uses a different syntax than its predecessor
17	Rate Mismatch	QoS Specification + Enforcement	The current element expects input at a different rate than its predecessor outputs
18	Semantic Mismatch		The current element and its predecessor do not interpret a given value in the same way

2.1: Eliminating Classes of Internal Faults

Overview: While an analyst can consider each guideword individually, we also provide the following questions which can be used to eliminate entire classes of faults. Note that the default choice is italicized, and non-default answers should be justified in the "Faults Not Considered" cells, Row 32+, Columns A-B (Guideword), C-I (Justification)

1. Phase of Creation or Occurrence -- "Should faults from the element's development be considered?"
   - *Yes* -- Development and operational faults
   - No -- Operational faults only (Remove 1-7)
2. Dimension -- "Does the element involve hardware, software, or both?"
   - Hardware -- Hardware only (Remove 1-3,13-15)
   - Software -- Software only (Remove 4-12)
   - *Both* -- Both hardware and software
3. Phenomenological cause, pt 1 (unless Software dimension only) -- "Will the hardware elements be protected from natural phenomena?"
   - Yes -- No Natural faults (Remove 7-9)
   - *No* -- Natural faults included
4. Phenomenological cause, pt 2 -- "Does the element receive input from directly from a human operator?"
   - *Yes* -- Human-made operational faults included
   - No -- Human-made operational faults excluded (Remove 10-11,14-15)
5. Objective -- "Is it possible that an adversary could gain access to the element?"
   - *Yes* -- Malicious and Non-Malicious faults
   - No -- Non-Malicious faults only (Remove 3-4,12-13)
6. Interaction -- "Have the two components joined by this connection either worked together before or been developed together?"
   - Yes -- No interaction faults (Remove 15-18)
   - *No* -- Interaction faults

2.2 Examining the Internally Caused Dangers

Overview: Here the analyst explains how the successor dangers (identified in step 1.1) could be caused by faults internal to the element using the guideword table from above.

1. Select a guideword
    1. Guide
        - The first thing an analyst needs to do is to select one of the non-eliminated guidewords
        - Enter a reference to the guideword:
            1. Row: (38+)
            2. Column: Guideword (B)
        - Each guideword may have more than one row -- this signifies that the same guideword may cause multiple dangers in the successor component
    2. Examples

- Component: Operator SW Mistake
- Connection: Compromised Software

2. Select a Successor Danger that this guideword could cause
    1. Guide
        - Next, the analyst should pick one of the successor dangers the guideword from 2.2-1 could cause
        - Enter a reference to the danger:
            1. Row: (38+)
            2. Column: Successor Danger (A)
        - Each successor danger may have more than one row -- this signifies that different faults in the current element will cause the same danger in the successor component
    2. Examples
        - Component: IVLine.Overinfusion
        - Connection: H.PatientOverdose
3. Interpret the danger
    1. Guide
        - Since one guideword can be interpreted in different ways, the analyst should now provide a concrete interpretation that explains how the guideword in column B causes the successor danger in column A
        - Enter the interpretation of the danger
            1. Row: (38+)
            2. Column: (D-E)
    2. Example
        - Component: The PCA pump runs even though it's not commanded to
        - Connection: The connection drops the message
4. Identify any Co-occurring Dangers
    1. Guide
        - Sometimes dangers will only manifest in the presence of other dangers -- this may be reflected in the natural language of the interpretation, but should be made explicit here by referring to the other dangers (separated by commas) here.
            - Note that these can be other internal faults, or external dangers from Step 1
        - Each cause of the danger will need its own entry. So, if two faults (A and B) have to occur simultaneously for the danger to occur, four rows will need to be created:
            - Two which cause the successor danger:

            - Fault A will have one row in the table (with fault B as a co-occurring danger)
            - Fault B will have its own subsequent row (with fault A as a co-occurring danger)
        - Two which are not hazardous:
            - Fault A without a simultaneous fault B
            - Fault B without a simultaneous fault A
    - Enter the name of co-occurring dangers, or "None" if not required
        1. Row: (38+)
        2. Column: Co-occurring Dangers, (E)
    - Dangers are assumed to be combined via AND joins --- more complex relationships (OR, M-of-N, etc.) can be explained in the interpretation / global env. state columns.
2. Example
    - Component: N/A
    - Connection: N/A

5. Design-time Detection
    1. Guide
        - This allows an analyst to specify a mechanism to detect the presence of a fault at the design-time of a system
        - Avizienis et-al. state that there are five verification approaches (see Fig. 19, page 18),
            - Static Analysis,
            - Theorem Proving,
            - Model Checking,
            - Symbolic Execution, or
            - Testing
        - Record the mechanism of detection (which may be one or more of the five verification methods or a domain-specific approach), or "None" if design-time compensation is impossible
            1. Row (38+)
            2. Column Run-time Handling (F)
    2. Example:
        - Component: Model Checking
        - Connection: Testing
6. Run-time Detection
    1. Guide
        - This allows an analyst to specify a mechanism to detect the occurrence of a danger at runtime

- Avizienis et-al. state that there are two ways errors can be detected at runtime (see Fig. 16, page 16), either
  - Concurrently (as the element is performing its job), or
  - Preemptively (while the element is suspended for testing)
- Record the mechanism (typically prefaced with either "Concurrent" or "Preemptive"), or "None" if run-time detection is impossible
  1. Row (38+)
  2. Column Run-time Detection (G)

2. Example:
   - Component: None
   - Connection: Concurrent: Flow metering

7. Run-time Error Handling
   1. Guide
      - This allows an analyst to specify a mechanism to correct the occurrence of a danger at runtime
      - Avizienis et-al. state that there are three ways errors can be handled at runtime (see Fig. 16, page 16),
        - Rollback (restoring the system to a saved state),
        - Rollforward (moving to a state without errors, ie a known-safe state), or
        - Compensation (use redundancy to mask the error)
      - Record the mechanism of correction (typically prefaced with "Rollback", "Rollforward", or "Compensation"), or "None" if run-time compensation is impossible
        1. Row (38+)
        2. Column Run-time Handling (H)
   2. Example:
      - Component: None
      - Connection: Rollforward: Stop Analgesic Flow
8. Run-time Fault Handling
   1. Guide
      - This allows an analyst to specify a mechanism to correct the cause of a fault at runtime
      - Avizienis et-al. state that there are four ways faults can be handled at runtime (see Fig. 16, page 16),
        - Diagnosis (identifying and recording the causes of the issue),
        - Isolation (physically or logically excluding the faulty components from further participation in the system),

    - Reconfiguration (switching in spare components or reassigning tasks among non-failed components), or
    - Reinitialization (“rebooting” the system)
  - Record the mechanism of correction (typically prefaced with "Diagnosis", “Isolation”, “Reconfiguration”, “Reinitialization”), or "None" if run-time handling is impossible
    3. Row (38+)
    4. Column Run-time Handling (I)
2. Example:
  - Component: Reinitialization: Reboot the pump
  - Connection: None