

# STPA Step 4 Building Scenarios

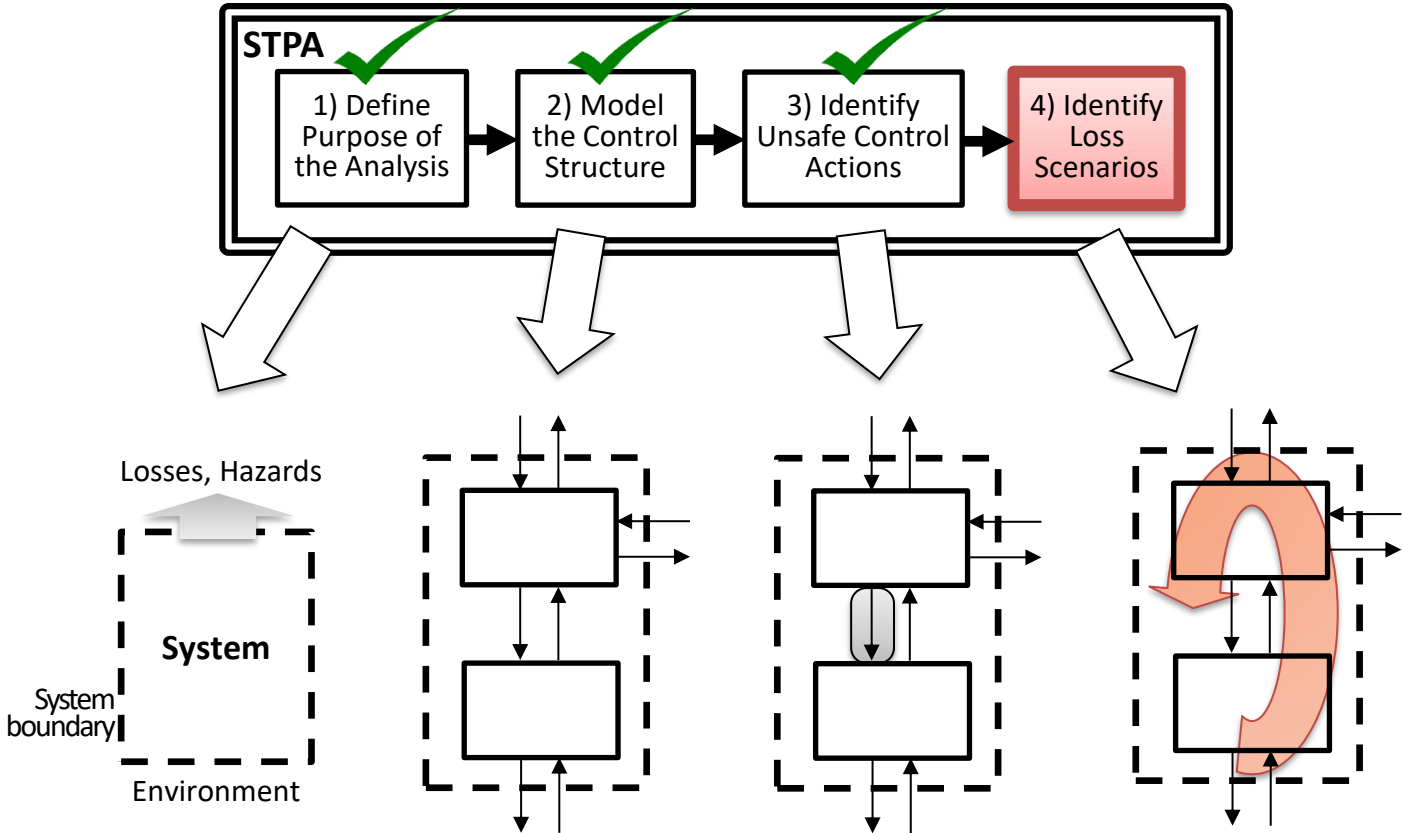
## A Formal Scenario Approach

John Thomas

Any questions? Email me! [JThomas4@mit.edu](mailto:JThomas4@mit.edu)

# Agenda

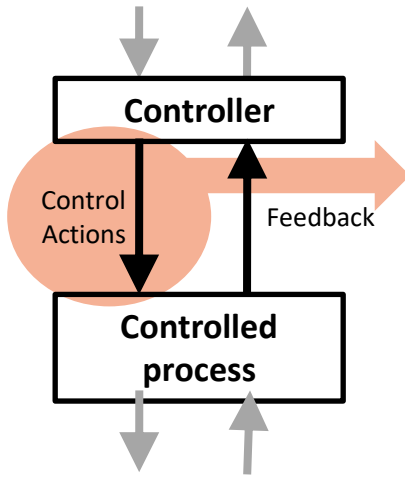
- Goals for a new scenario development process
- The new process
- Results from real-world testing of the process



# Goals for a Formal Scenario Building Approach

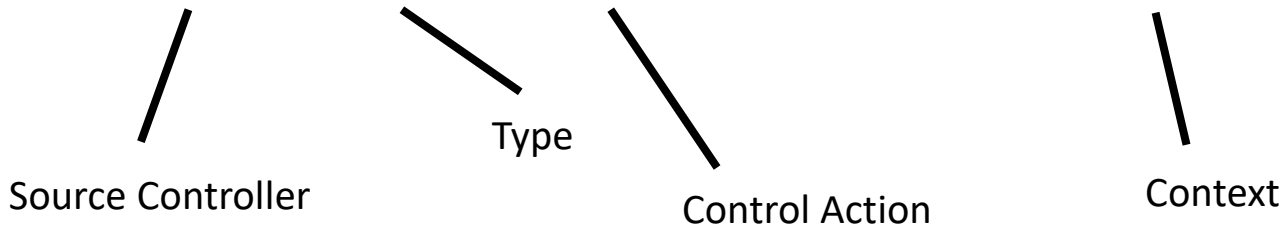
- Provide more scenario guidance for new practitioners who may get stuck
- Provide a formal structure for scenarios (similar to UCA syntax)
- Provide a way to review scenario completeness and find gaps
- Handle more complexity with less effort
  - Use top-down approach, not a backward search
  - Reduce repetition in scenarios

# Example of STPA Rigor: Unsafe Control Actions (UCA)



Example UCA:

“UCA-1: Computer provides Shift-to-Park cmd while vehicle is moving [H-1]”



“The UCA process works well, but the scenario process feels ad-hoc. Can you make the scenarios (STPA step 4) more like this?”

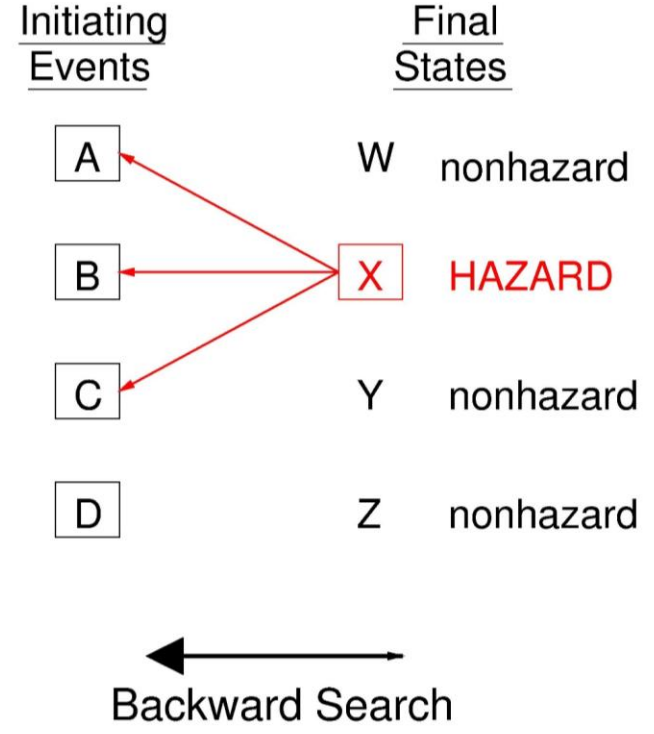
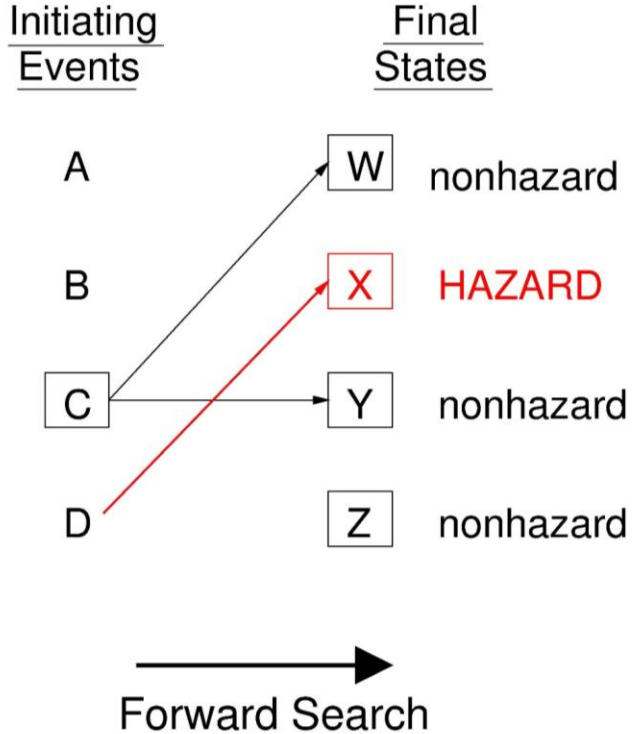
# BLUF: Bottom Line Up Front

- A formal STPA scenario approach has been defined and tested
- Result: It works, with advantages and disadvantages
  - Advantage: Increased rigor and repeatability; frequently catches previously unknown flaws
  - Disadvantage: Introduces more steps and rules, which take longer to teach
- We'll discuss the validation, testing, and other observations at the end

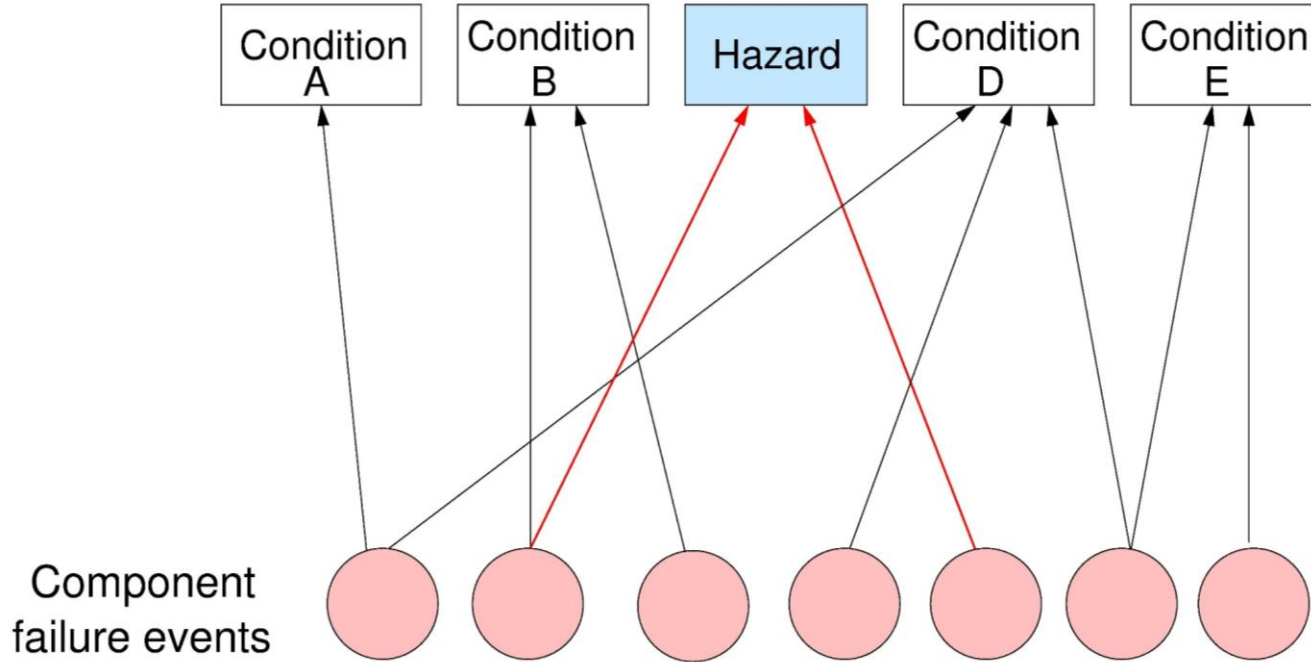
Quick review:

Hazard Analysis Search Strategies

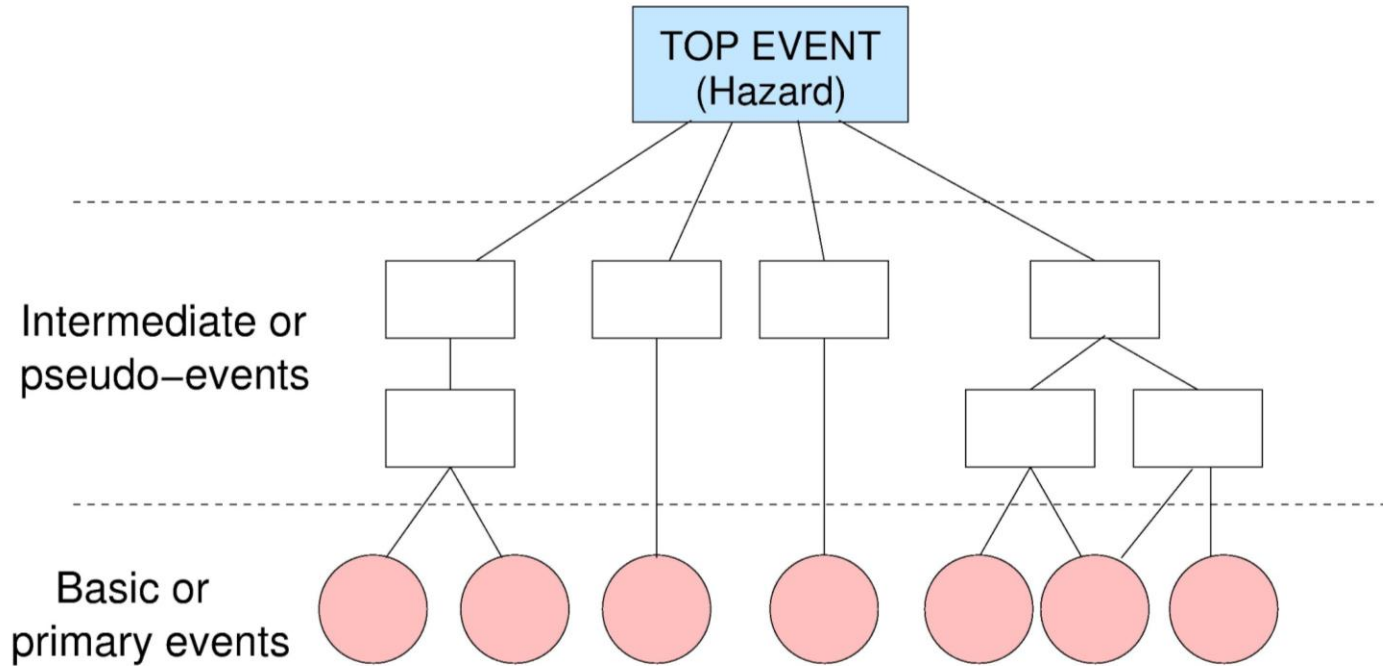
# Forward vs. Backward Search



# Bottom-Up Search



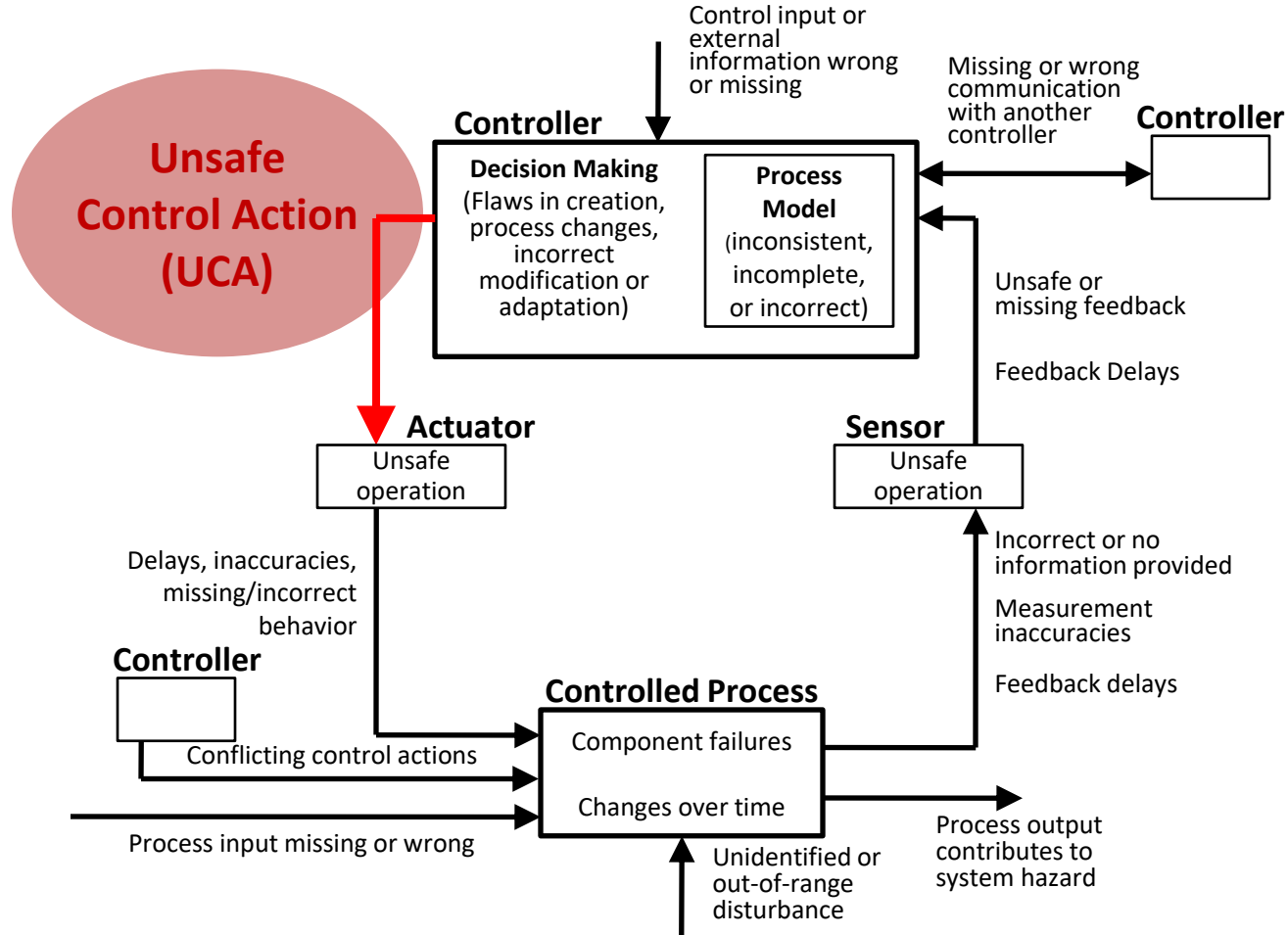
# Top-Down Search



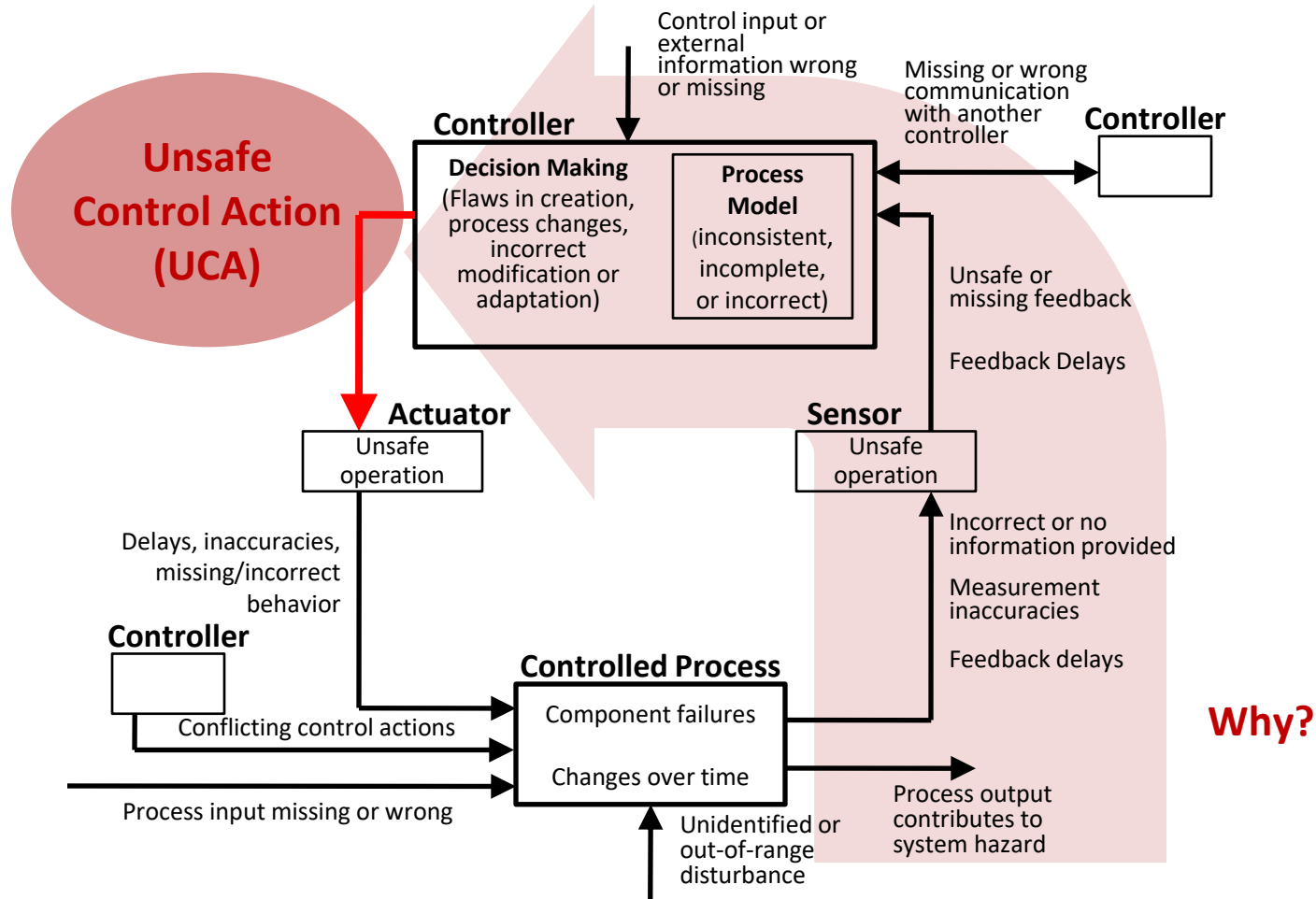
# STPA Step 4: Scenario Building

## The Old Process

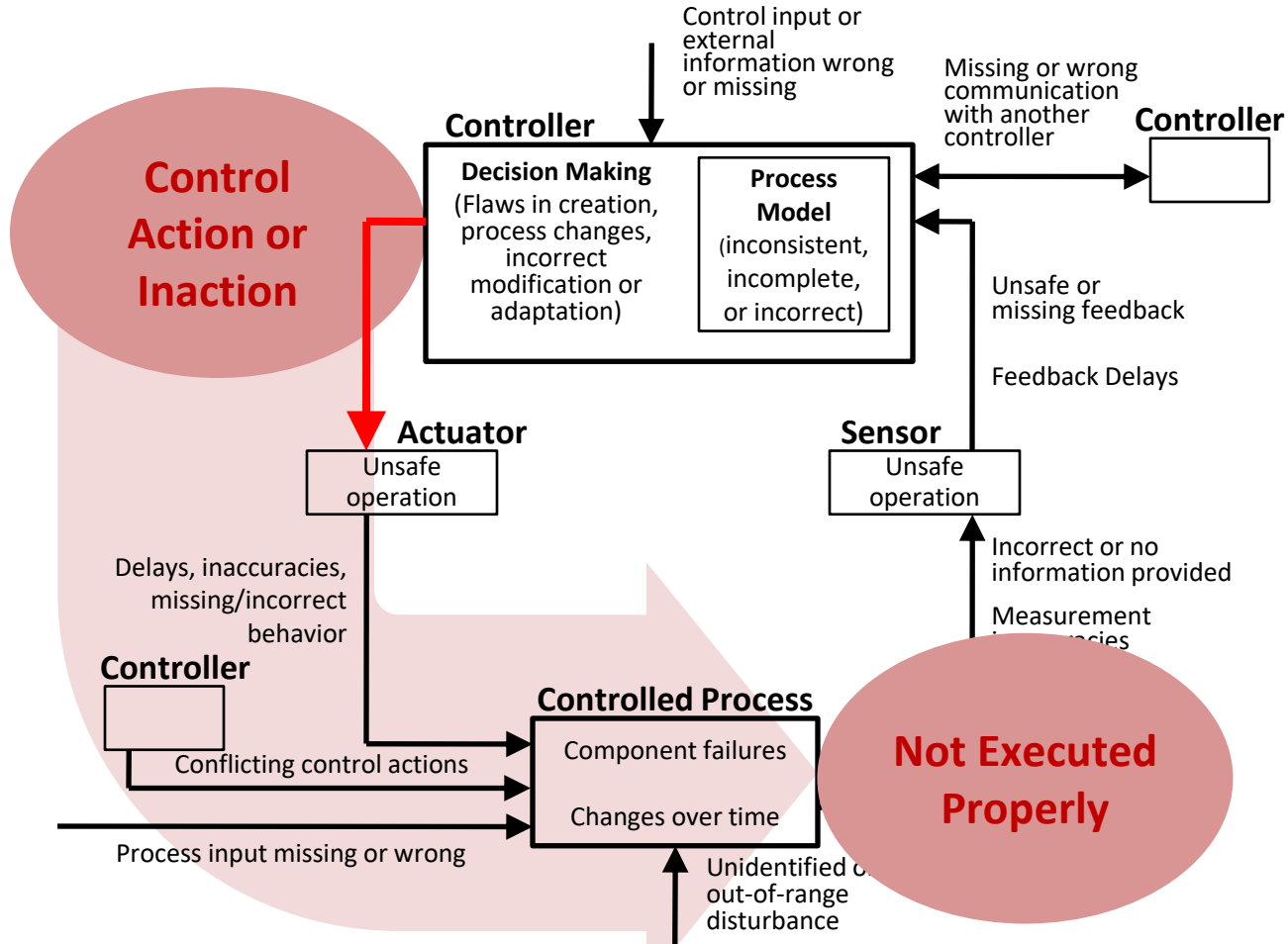
# STPA Step 4A: Identify scenarios that cause UCAs



# STPA Step 4A: Identify scenarios that cause UCAs

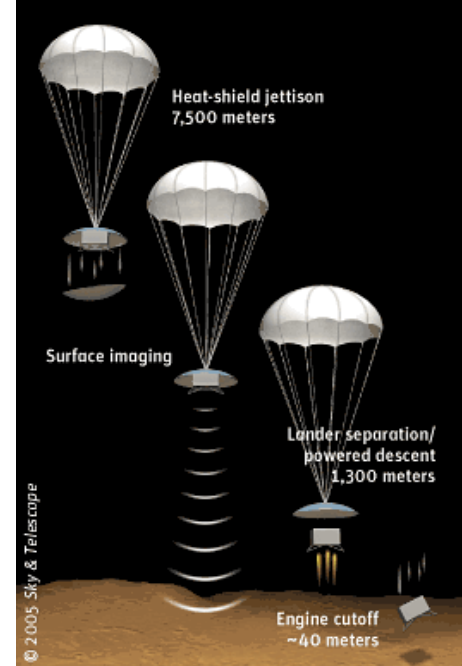


# STPA Step 4B: Potential control actions not followed properly



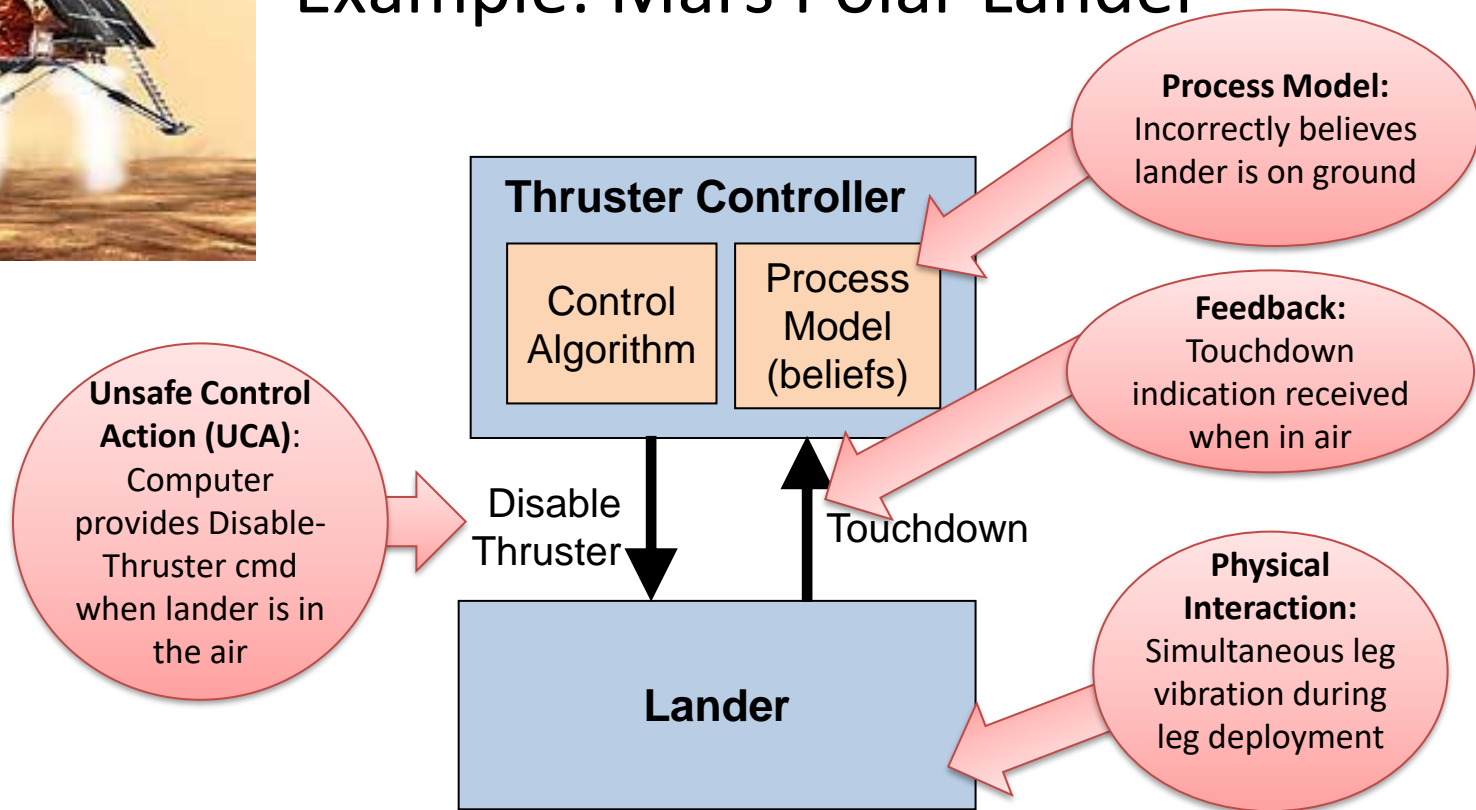
# Mars Polar Lander

- During descent to Mars, legs deployed (as planned)
- Footpad sensors detected vibration (within design spec)
- Momentary signal sent to computer (as required)
- Computer shut down the descent engines (as specified)
- The vehicle free-fell, fell to surface at 50 mph (80 kph), destroyed





# Example: Mars Polar Lander



# STPA Scenario Development

The New, Formal Process

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

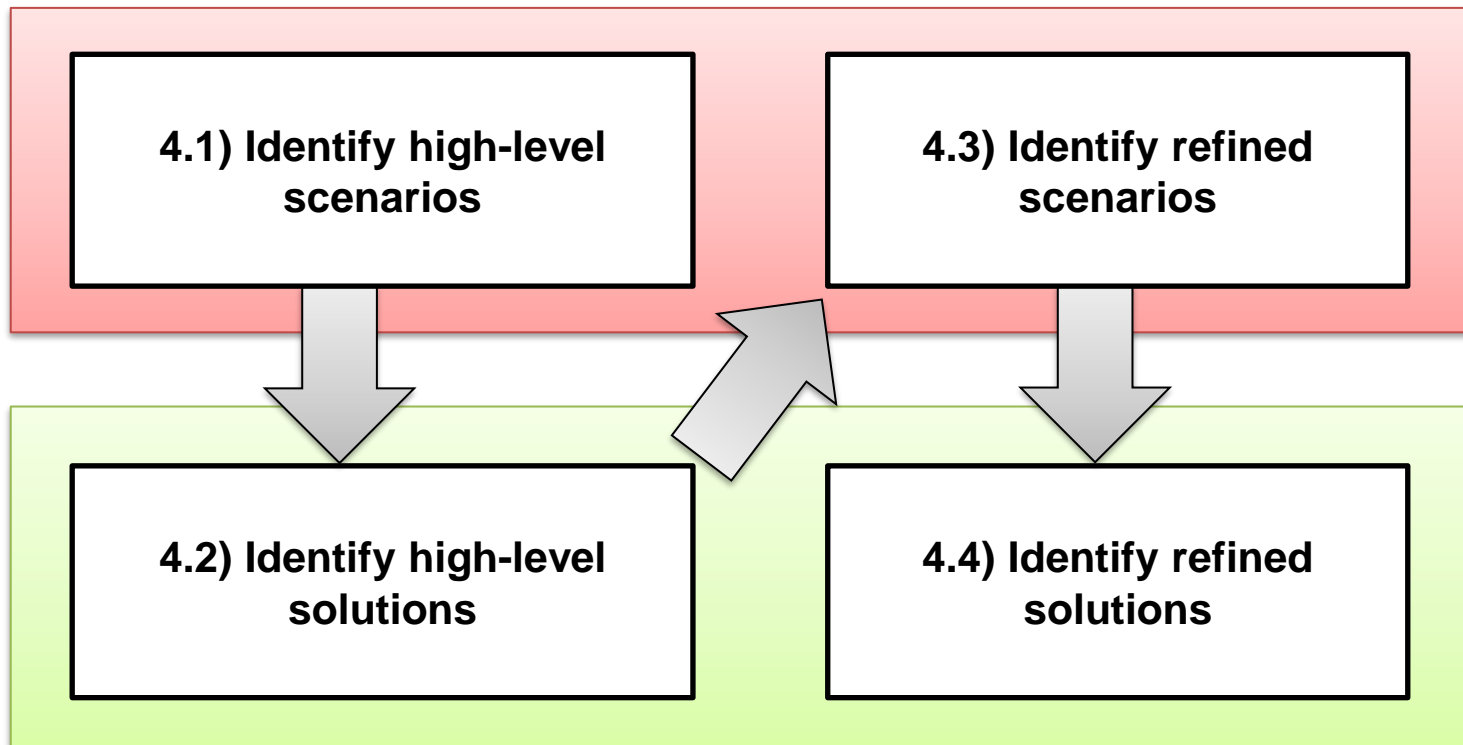
**4.1) Identify high-level scenarios**

**4.3) Identify refined scenarios**

**Solution Space:**  
What must be done to prevent losses?

**4.2) Identify high-level solutions**

**4.4) Identify refined solutions**



# STPA: Formal Scenario Building Process

**Problem Space:**  
What can go wrong?

## 4.1) Identify high-level scenarios

- Start with broad, abstract scenarios
- Follow formal syntax for each scenario class

## 4.3) Identify refined scenarios

- Use more detailed model or design info
- May be done in parallel with development

**Solution Space:**  
What must be done to prevent losses?

## 4.2) Identify high-level solutions

- Requirements
- Modify control actions
- Modify types of feedback
- Modify responsibilities
- Etc.

## 4.4) Identify refined solutions

- Use more detailed model or design info
- May be done in parallel with development

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

## 4.1) Identify high-level scenarios

- Class 1
- Class 2
- Class 3
- Class 4

## 4.3) Identify refined scenarios

- Class 1
- Class 2
- Class 3
- Class 4

**Solution Space:**  
What must be done to prevent losses?

## 4.2) Identify high-level solutions

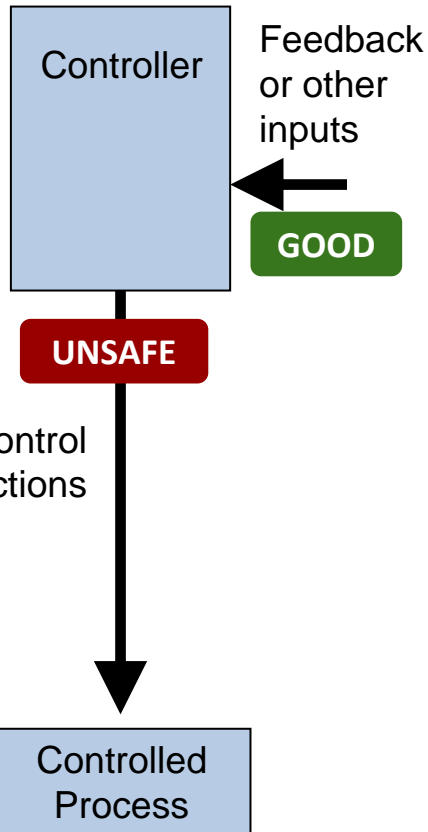
- Class 1
- Class 2
- Class 3
- Class 4

## 4.4) Identify refined solutions

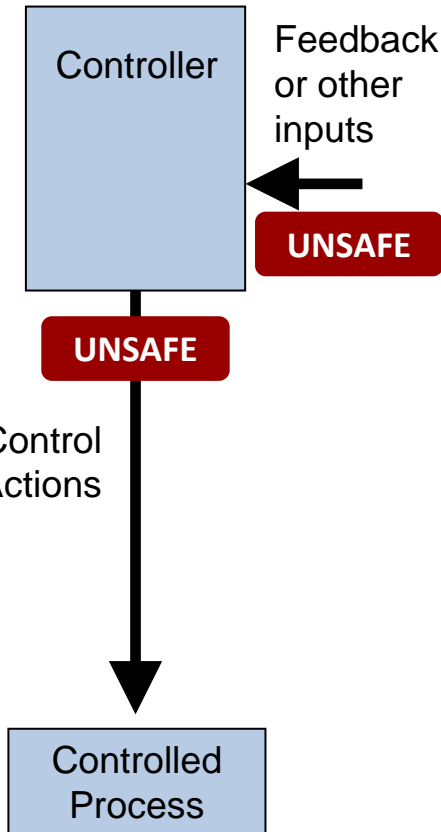
- Class 1
- Class 2
- Class 3
- Class 4

# STPA: Four Classes of Formal Scenarios

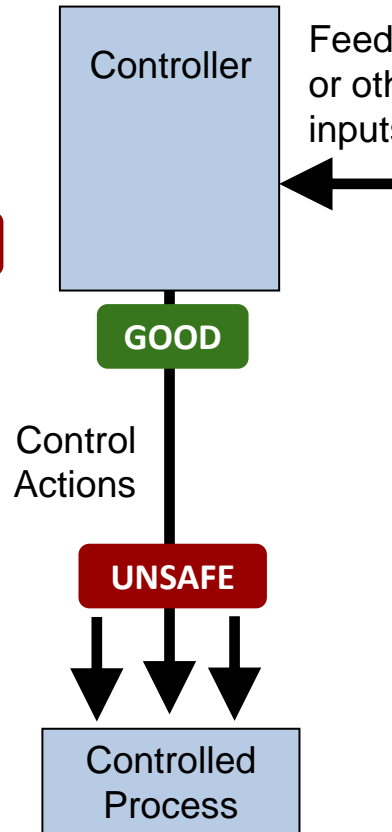
## Class 1



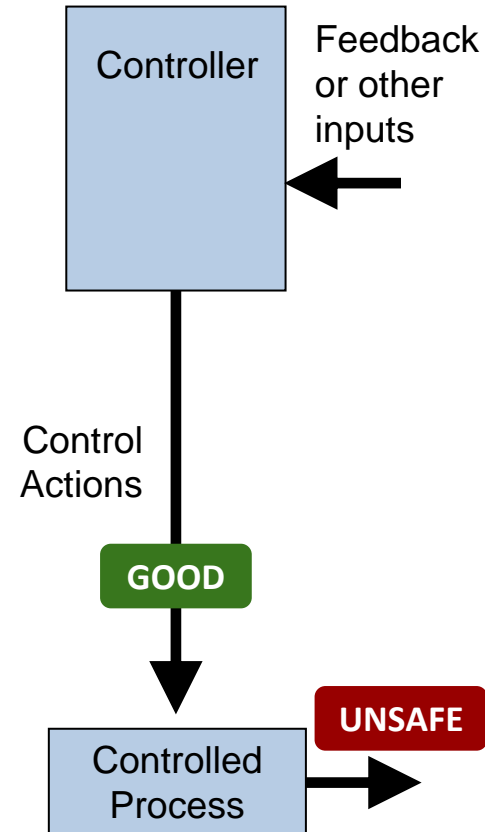
## Class 2



## Class 3

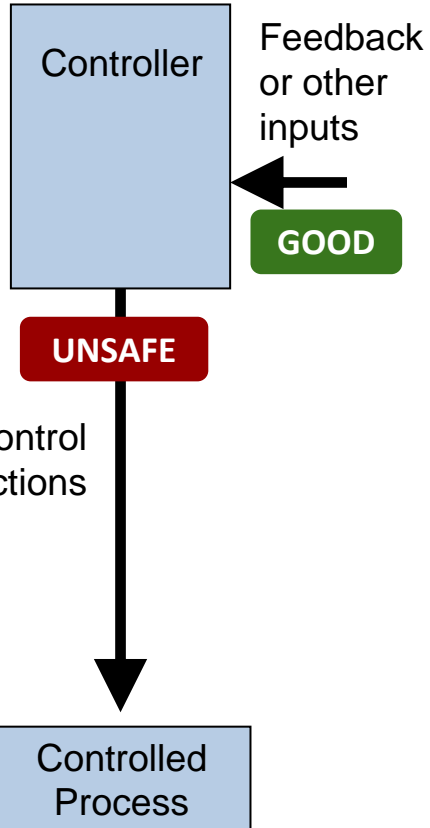


## Class 4

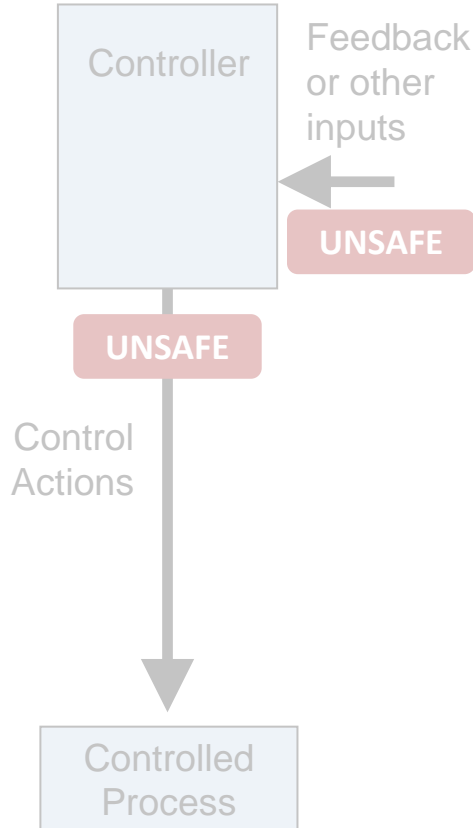


# Four Classes of Formal Scenarios

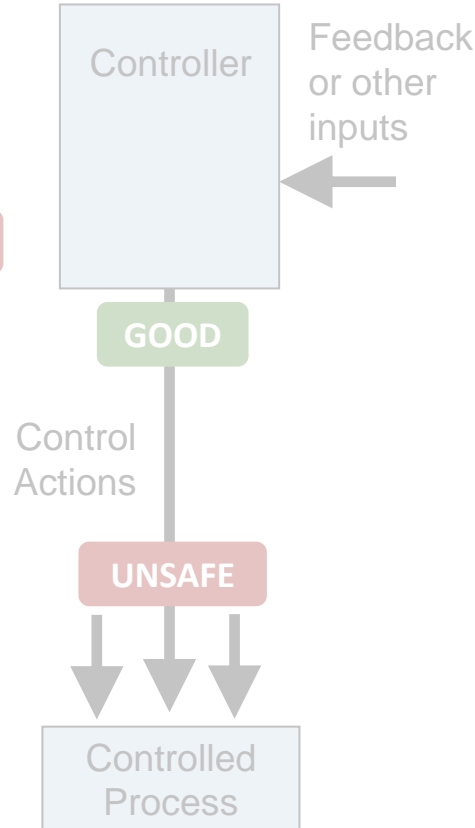
## Class 1



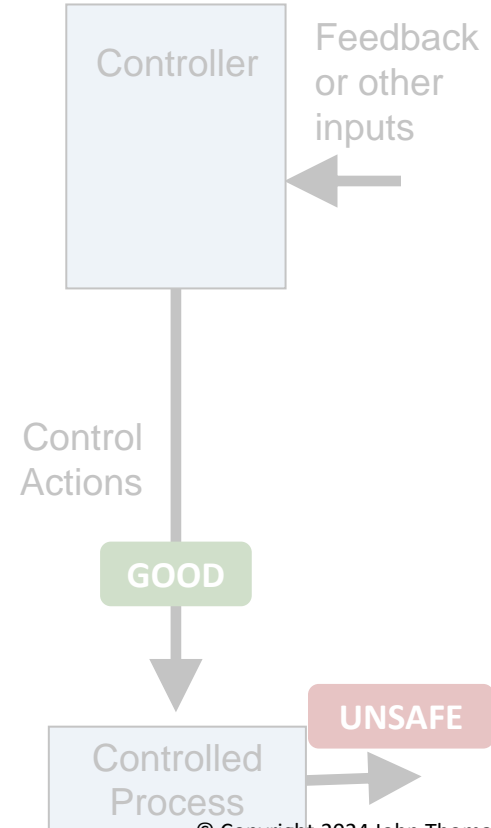
## Class 2



## Class 3

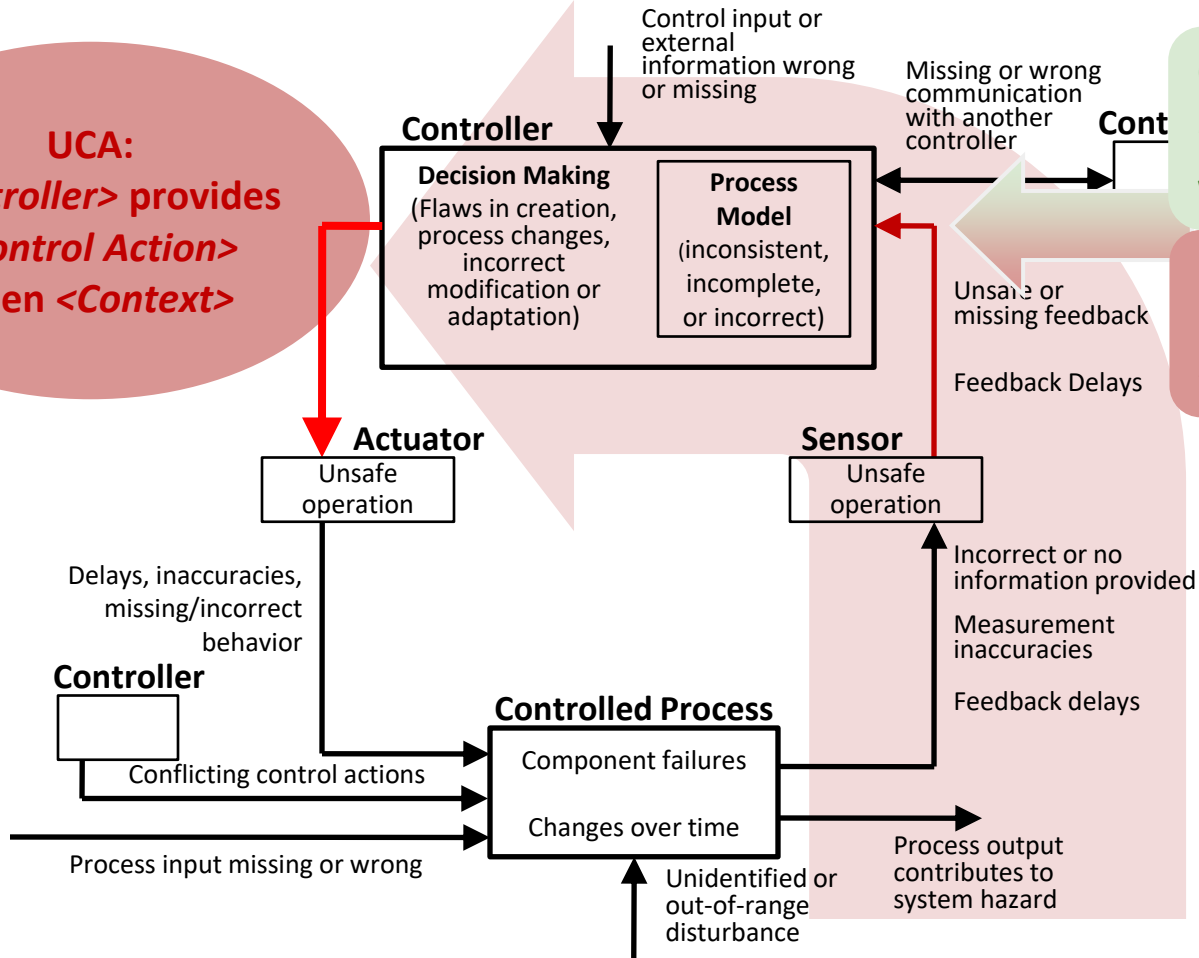


## Class 4



# STPA: Identify scenarios that cause UCAs

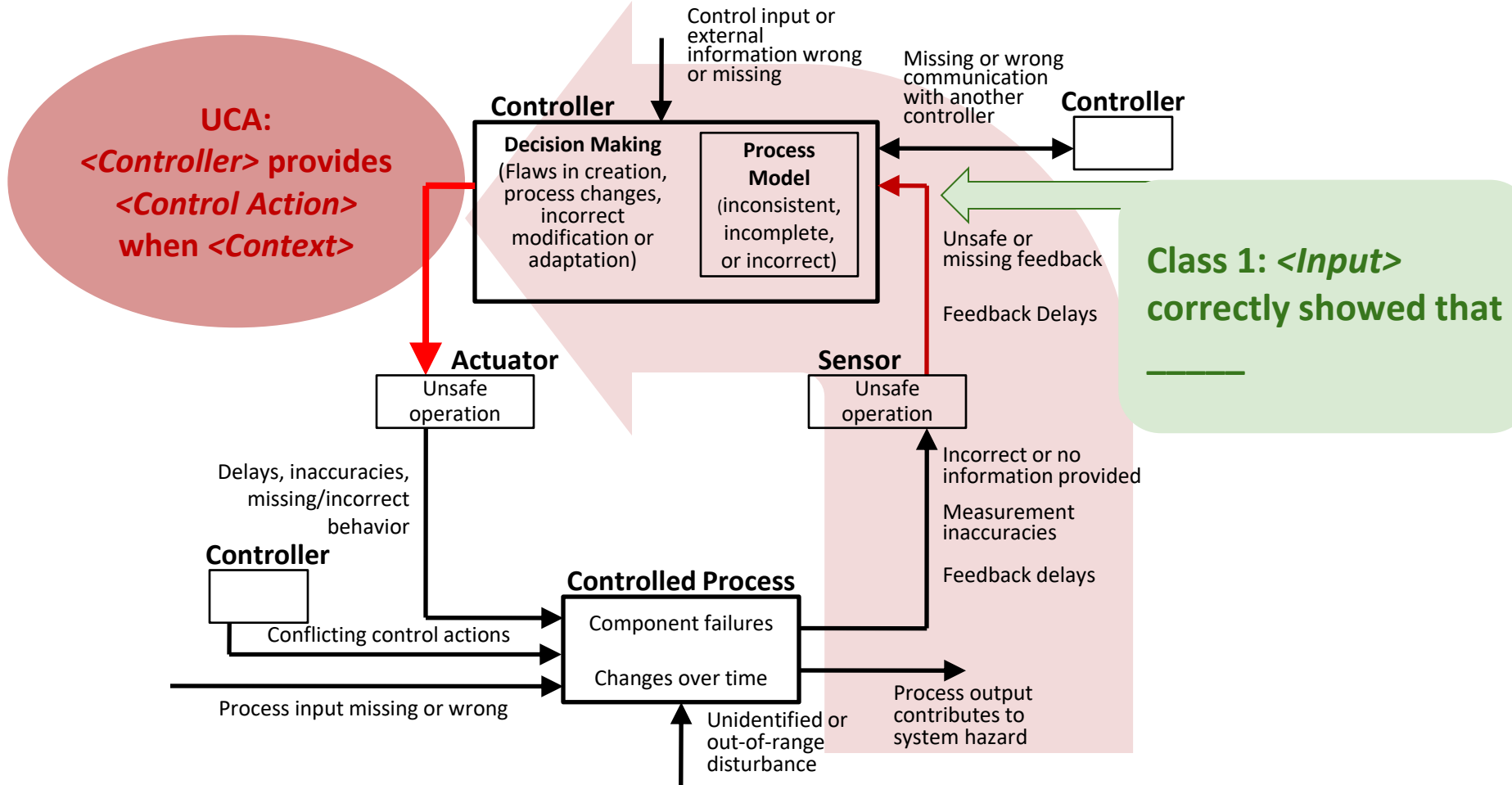
**UCA:**  
*<Controller>* provides  
*<Control Action>*  
when *<Context>*



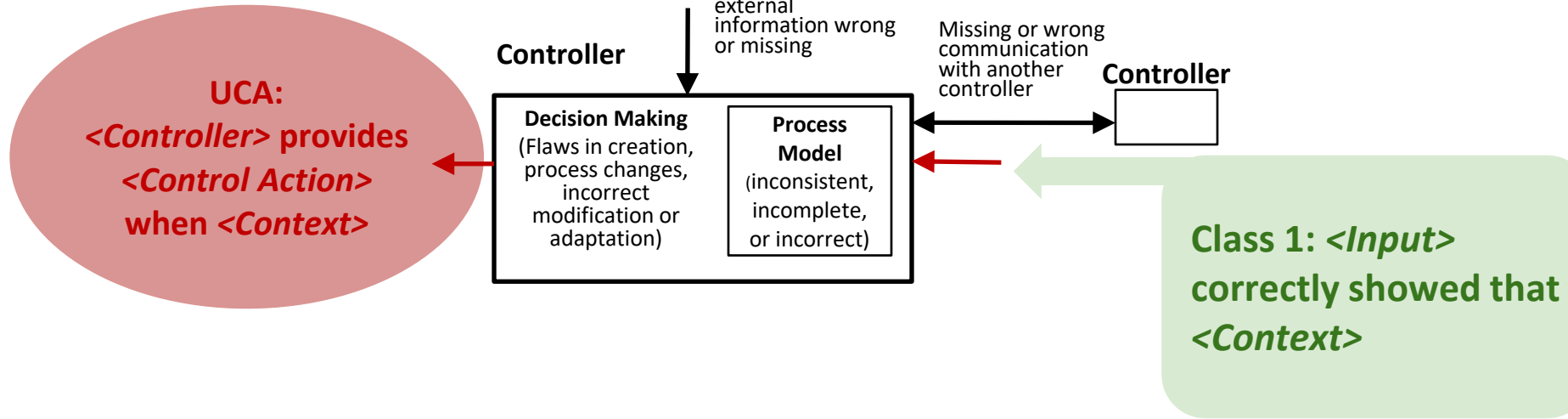
**Class 1:**  
*<Feedback/input>* \_\_\_  
was adequate

**Class 2:**  
*<Feedback/input>* \_\_\_  
was unsafe

# STPA: Class 1 Scenario Archetype



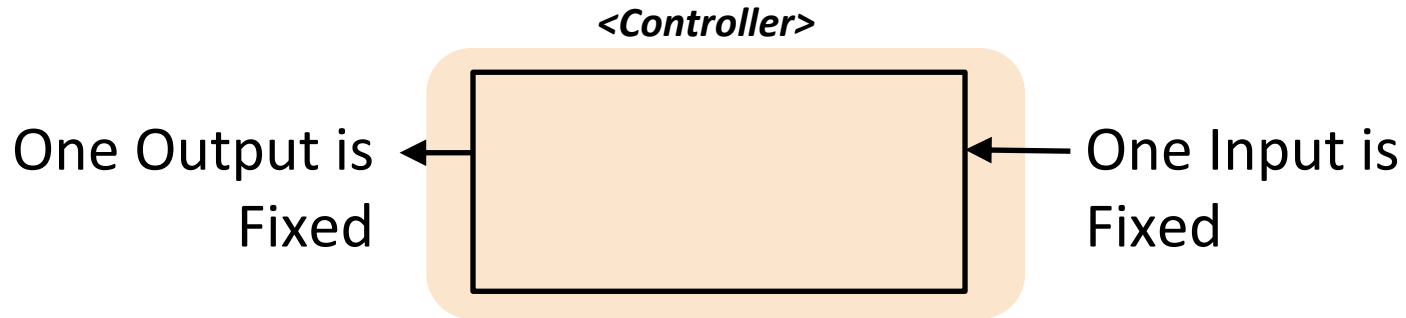
# STPA: Class 1 Scenario Archetype



## Class 1 Scenario Archetype:

- **Output:** UCA: <Controller> provides <Control Action> when <Context>
- **Input:** <Input> correctly showed that <Context>

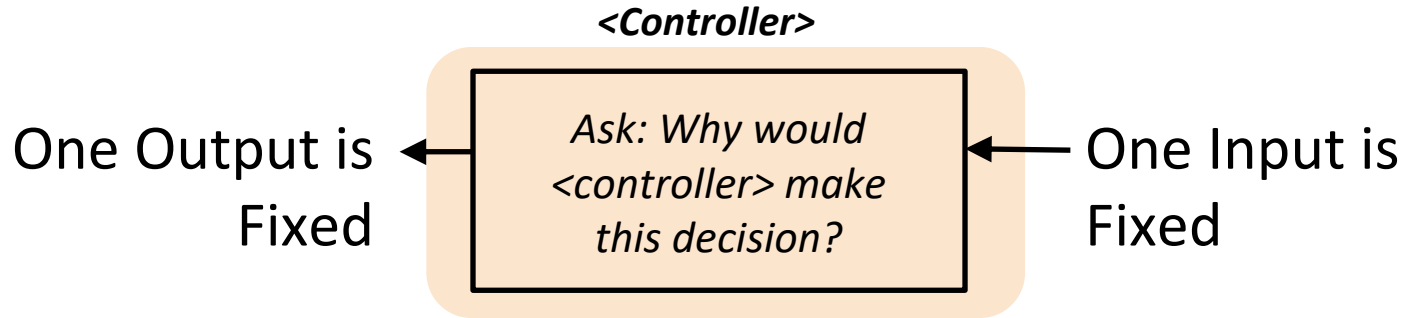
# General Transfer Function Concept



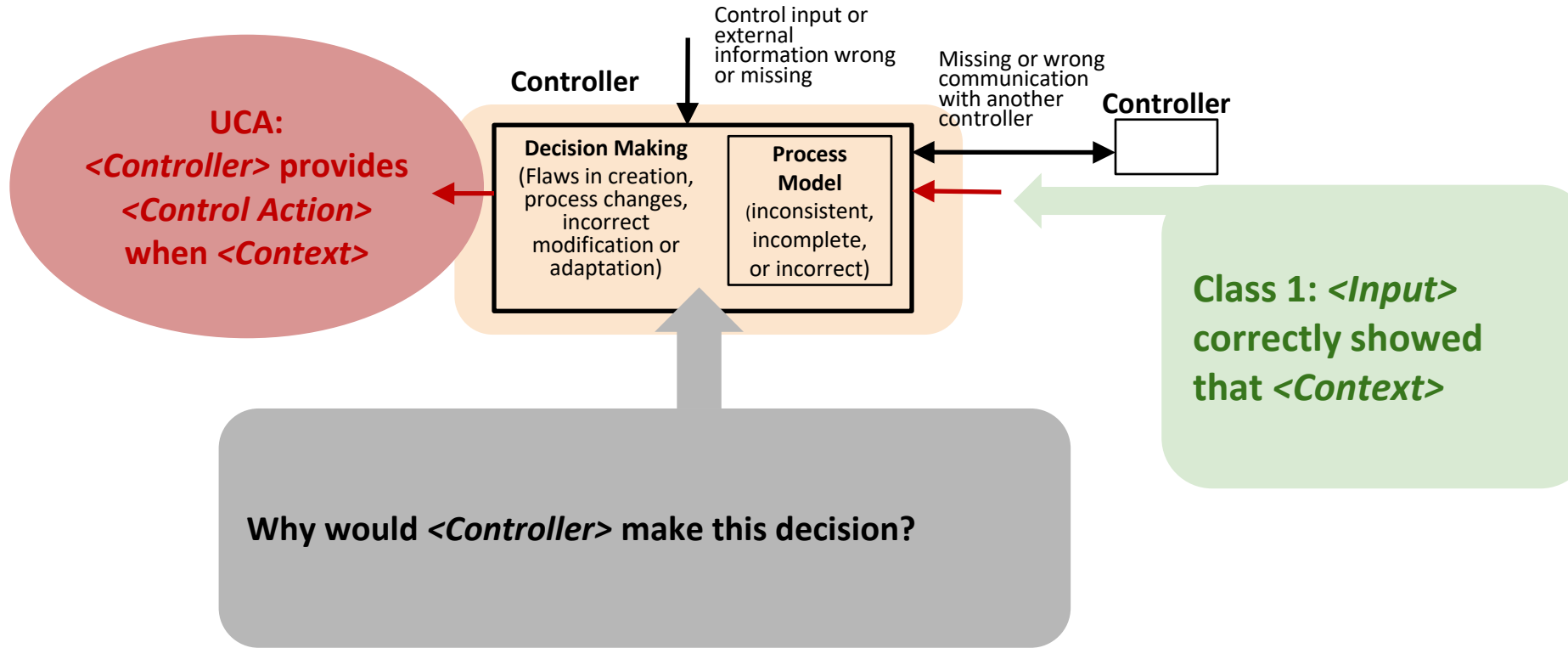
## Class 1 Scenario Archetype:

- **Output:** UCA: *<Controller>* provides *<Control Action>* when *<Context>*
- **Input:** *<Input>* correctly showed that *<Context>*

# General Transfer Function Concept



# STPA: Class 1 Scenario Archetype

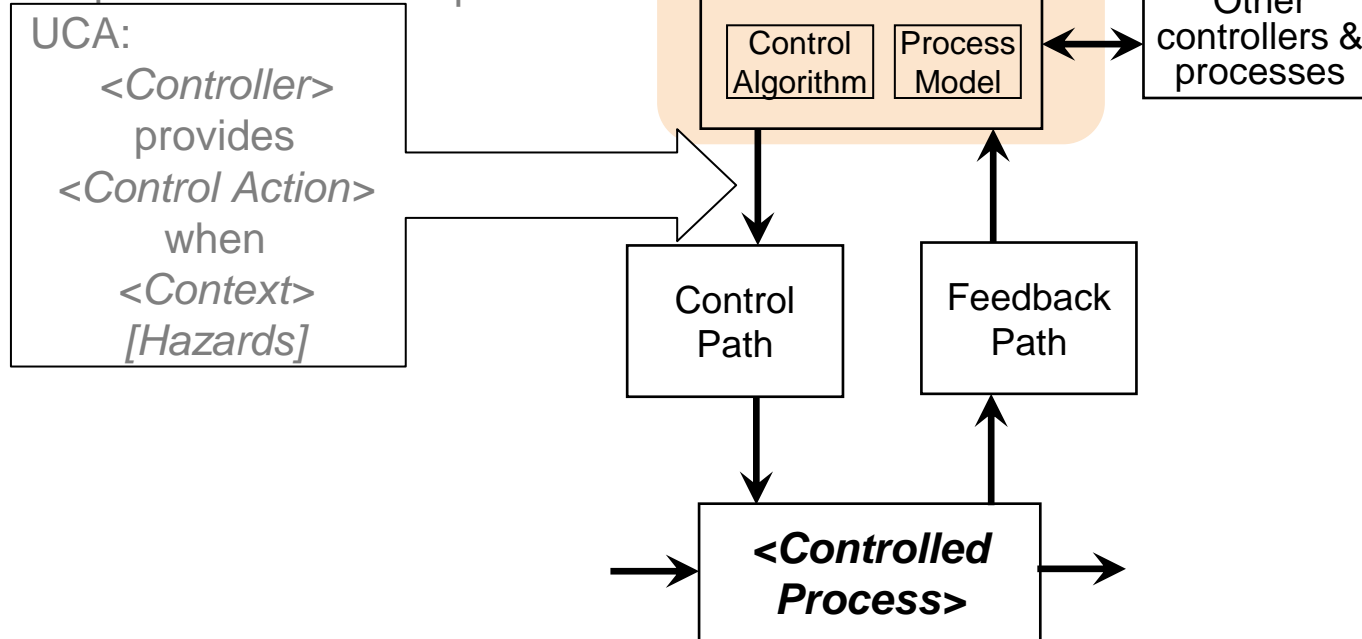


We'll refine these scenarios and identify causes in another step (4.3).

## Class 1 Scenario Archetype: **Unsafe Controller Behavior**

- UCA: *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

Result from previous STPA Step 3



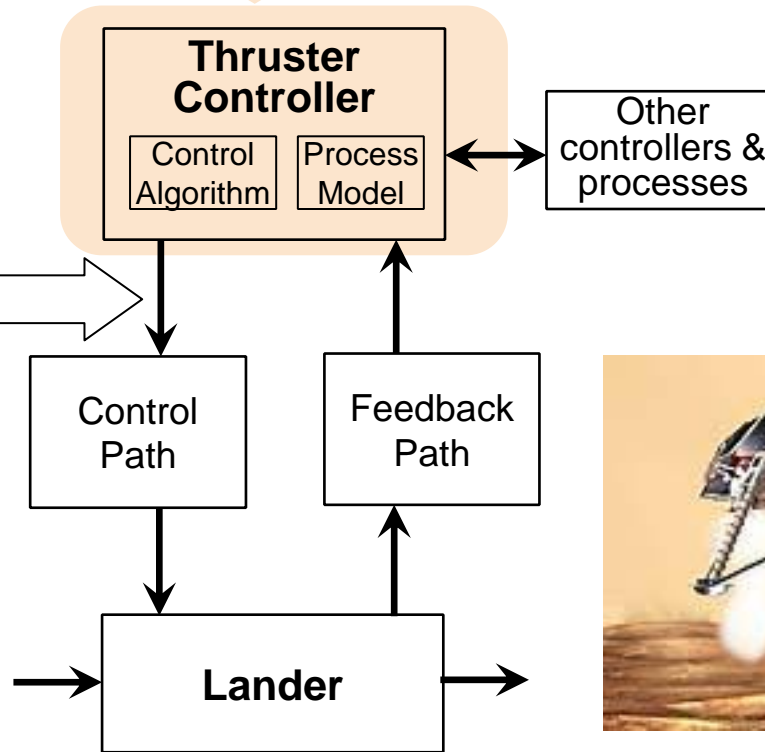
## Class 1 Scenario Archetype:

### Unsafe Controller Behavior

- UCA: *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

Result from previous STPA Step 3

<i>UCA:</i>	<i>UCA-2:</i>
<i>&lt;Controller&gt;</i>	<i>Thruster Controller</i>
provides	provides
<i>&lt;Control Action&gt;</i>	<i>Disable-Thruster Cmd</i>
when	when
<i>&lt;Context&gt;</i>	<i>lander is in the air</i>
<i>[Hazards]</i>	<i>[H-1]</i>



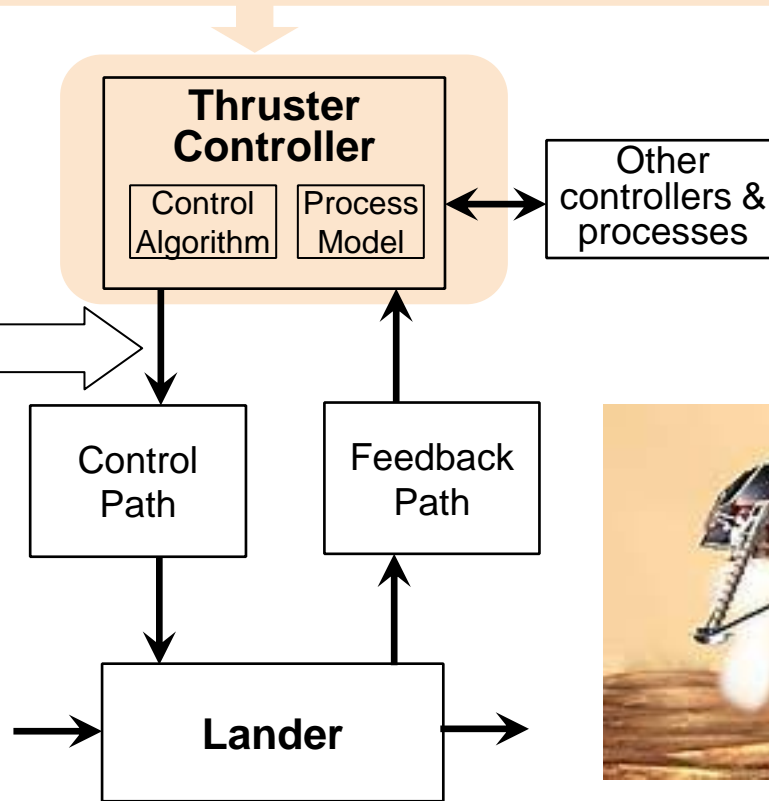
## Class 1 Scenario Archetype:

### Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- *<Input>* to *<Controller>* correctly indicates *<Context>*

Result from previous STPA Step 3

<i>UCA:</i>		<i>UCA-2:</i>
<i>&lt;Controller&gt;</i>	→	<i>Thruster Controller</i>
provides	→	provides
<i>&lt;Control Action&gt;</i>	→	<i>Disable-Thruster Cmd</i>
when	→	when
<i>&lt;Context&gt;</i>	→	<i>lander is in the air</i>
<i>[Hazards]</i>	→	<i>[H-1]</i>



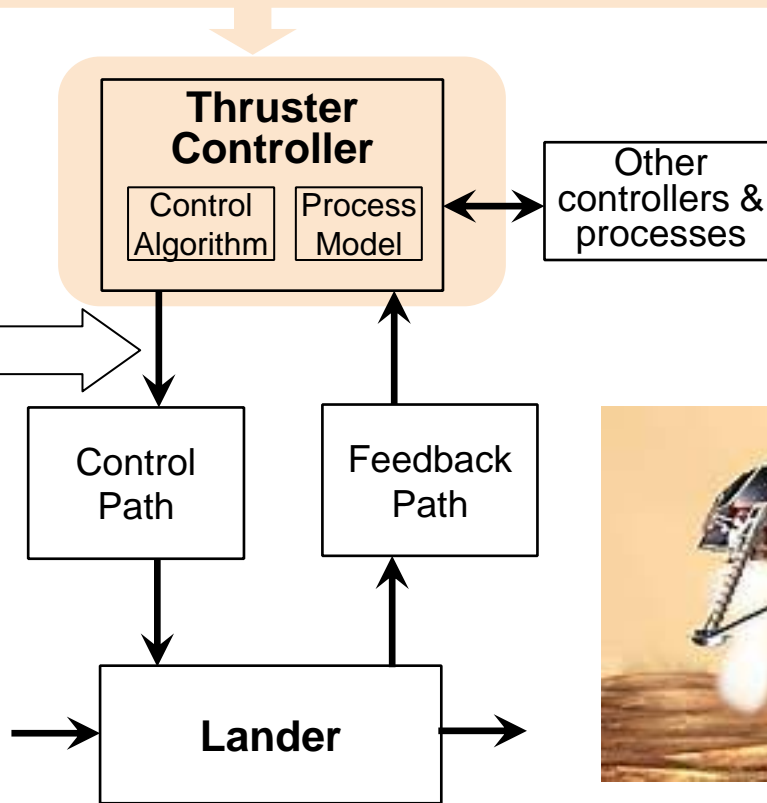
## Class 1 Scenario Archetype:

### Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- *<Input>* to Thruster Controller correctly indicates lander is in the air

Result from previous STPA Step 3

<u>UCA:</u>		<u>UCA-2:</u>
<u>&lt;Controller&gt;</u>	→	<u>Thruster Controller</u>
provides	→	provides
<u>&lt;Control Action&gt;</u>	→	<u>Disable-Thruster Cmd</u>
when	→	when
<u>&lt;Context&gt;</u>	→	<u>lander is in the air</u>
[Hazards]	→	[H-1]



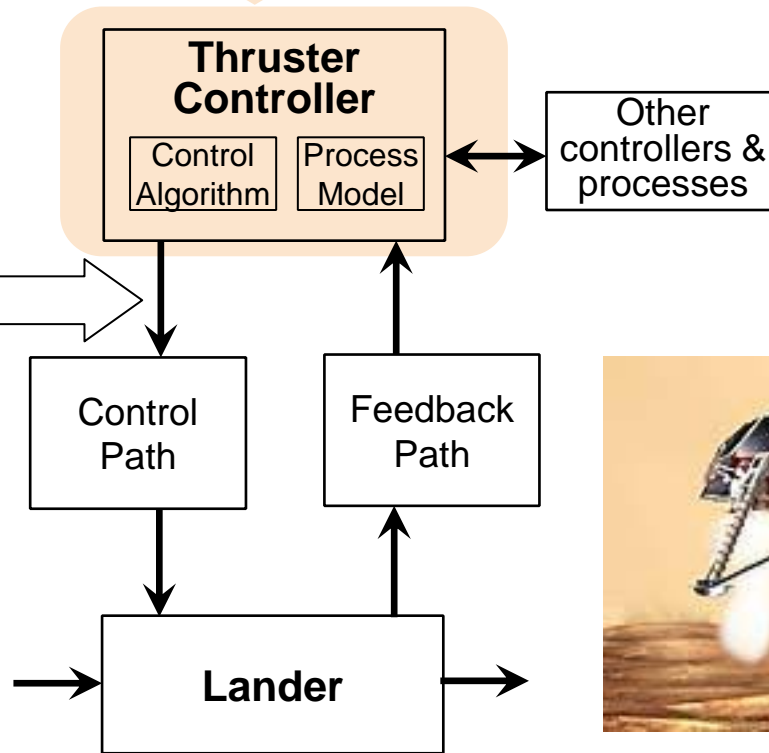
## Class 1 Scenario Archetype:

### Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air

Result from previous STPA Step 3

<u>UCA:</u>		<u>UCA-2:</u>
<u>&lt;Controller&gt;</u>	→	<u>Thruster Controller</u>
provides	→	provides
<u>&lt;Control Action&gt;</u>	→	<u>Disable-Thruster Cmd</u>
when	→	when
<u>&lt;Context&gt;</u>	→	<u>lander is in the air</u>
[Hazards]	→	[H-1]



# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

## 4.1) Identify high-level scenarios

- ✓ Class 1
- Class 2
- Class 3
- Class 4

## 4.3) Identify refined scenarios

- Class 1
- Class 2
- Class 3
- Class 4

**Solution Space:**  
What must be done to prevent losses?

## 4.2) Identify high-level solutions

- **Class 1**
- Class 2
- Class 3
- Class 4

## 4.4) Identify refined solutions

- Class 1
- Class 2
- Class 3
- Class 4

## 4.2) Identify High-level Solutions

What is needed from each part of the control structure to prevent the scenario? For example:

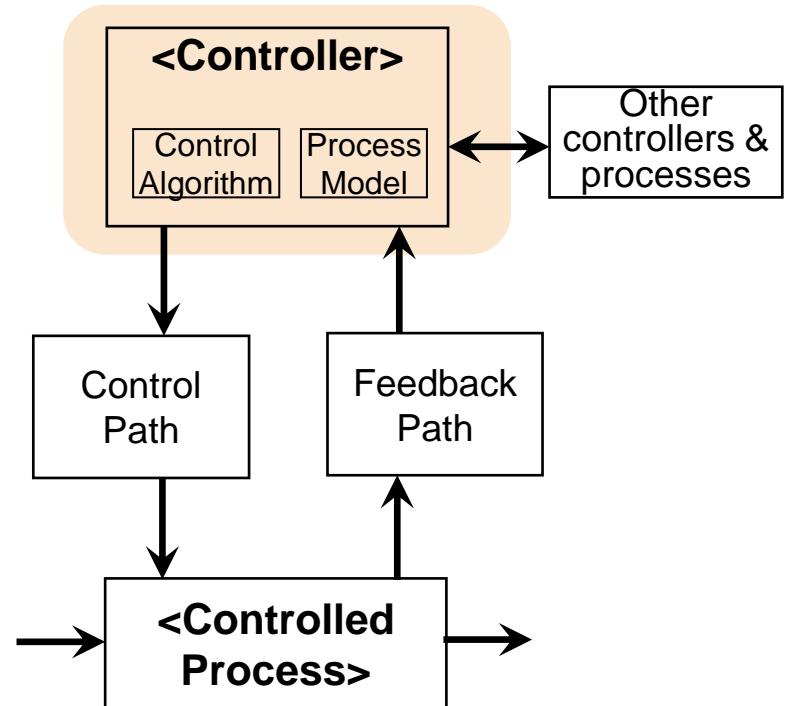
Structural mitigations

- Add/remove control actions?
- Add/remove controllers?
- Add/remove feedback?
- Add/remove processes?

Behavioral mitigations

- *<Controller>* must \_\_\_\_ when \_\_\_\_
- *<Feedback>* must \_\_\_\_ when \_\_\_\_
- *<Controller>* is/is not responsible for \_\_\_\_
- *<Controller>* Control Algorithm must \_\_\_\_
- *<Controller>* Process Model must include \_\_\_\_  
(e.g., make controller aware of some other state)
- *<Process>* must \_\_\_\_

Others?



**Discussion:** Solutions and decisions are guided by the scenarios that must be prevented.  
Can help minimize unnecessary complexity.

## 4.1)

### Class 1 Scenario Archetype:

#### Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air

## 4.2) Identify High-level Solutions

What can be done to prevent or mitigate this scenario?

- Process behavior: Add hardware / mechanical interlock
  - Mechanically prevent thruster cutoff before physical landing?
- Controller behavior: Include software interlock
  - Software must prevent thruster cutoff without simultaneous landing indication?
- Other solutions / mitigations?



## 4.1)

### Class 1 Scenario Archetype:

#### Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air

## 4.2) Identify High-level Solutions

What can be done to prevent or mitigate this scenario?

- Process behavior: Add hardware / mechanical interlock
  - Mechanically prevent thruster cutoff before physical landing?
- Controller behavior: Include software interlock
  - Software must prevent thruster cutoff without an active landing indication
- Other solutions / mitigations?



**Discussion:** These solutions can appear obvious when you're grounded in a scenario, which is a good thing. Imagine you don't have the scenario above, but you have a thousand pages of design info for MPL and a long list of failure-based safety assessments (e.g., software uses 3oo3 logic, so triple touchdown sensor failure is necessary to fool the software, but that likelihood is  $10^{-9}$ ). Missing functions like the one highlighted are not always obvious. Using structured scenario classes, like above, can make a missing requirement much more apparent. The actual MPL software was in fact missing the highlighted requirement, which contributed to the crash. It was overlooked by the safety assessment, which relied on failure-based methods like FTA and FMECA.

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

- 4.1) Identify high-level scenarios**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

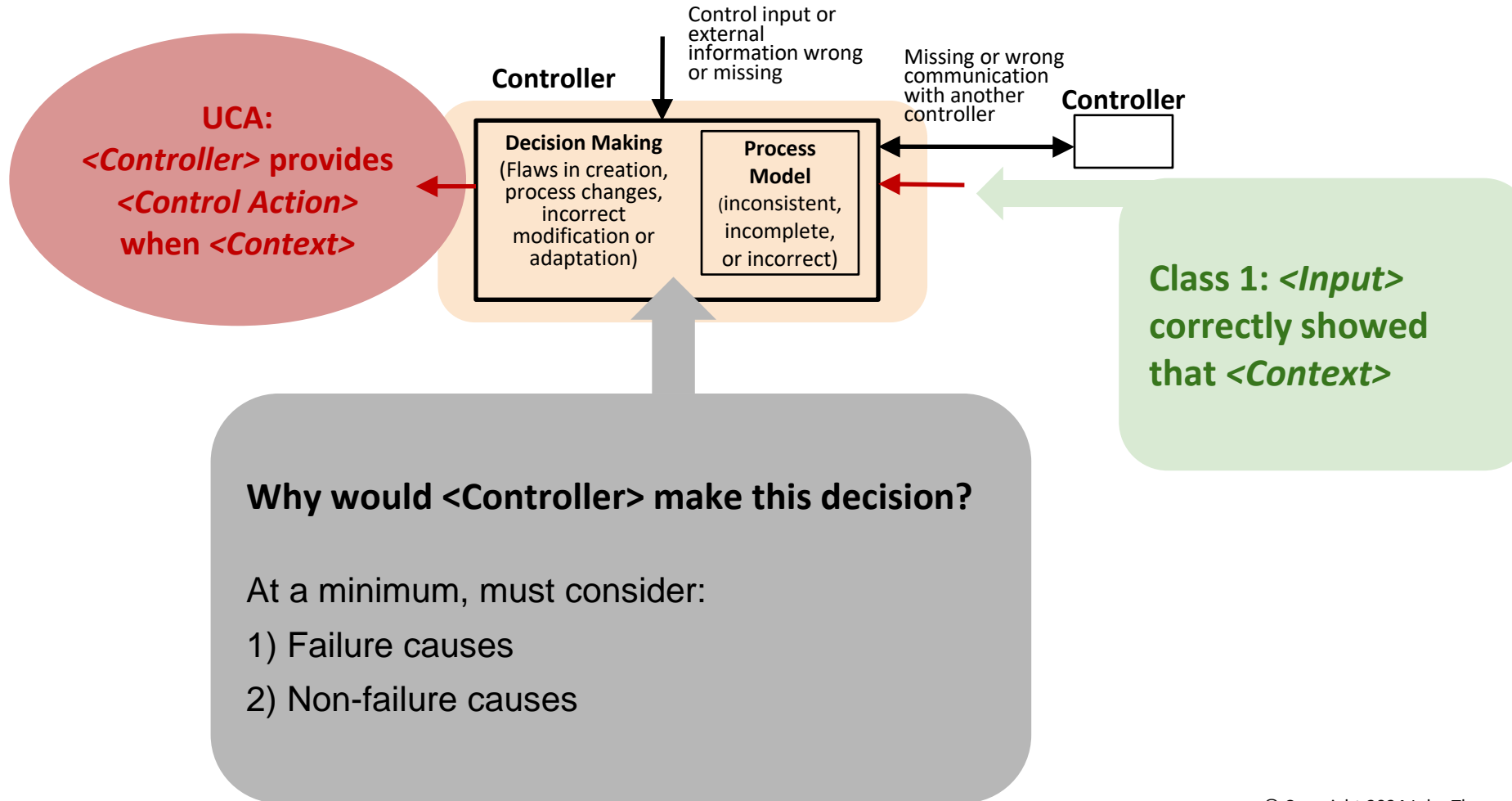
- 4.3) Identify refined scenarios**
- Class 1
  - Class 2
  - Class 3
  - Class 4

**Solution Space:**  
What must be done to prevent losses?

- 4.2) Identify high-level solutions**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

- 4.4) Identify refined solutions**
- Class 1
  - Class 2
  - Class 3
  - Class 4

# STPA: Class 1 Scenario Archetype



# Scenario Archetypes

UCA-2:

*<Controller> provides <Control Action> when <Context>*

## **Class 1 Scenario Archetype:** **Unsafe Controller Behavior**

- *<Controller> provides <Control Action> when <Context>*
- *<Input> to <Controller> correctly indicates <Context>*

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

# Refined Scenarios

*Why would <Controller> make this decision?*

- 1) Failure causes
- 2) Non-failure causes

A simple approach:  
Go ask the SMEs!

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

***Why would <Controller> make this decision?***

*Causes without failure:*

- Because <Controller> is designed to default to \_\_\_\_ if \_\_\_\_ (will ignore \_\_\_\_ and \_\_\_\_)
- Because <Controller> also received \_\_\_\_, so it appeared reasonable to <Control Action>
- Because there is no other \_\_\_\_, so other control actions were not feasible
- Etc.

Example SMEs responses from a real project.

Time required: under 15 minutes

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

***Refinement Option 1: Ask SMEs to help identify causes.***

***Is there a more rigorous option?***

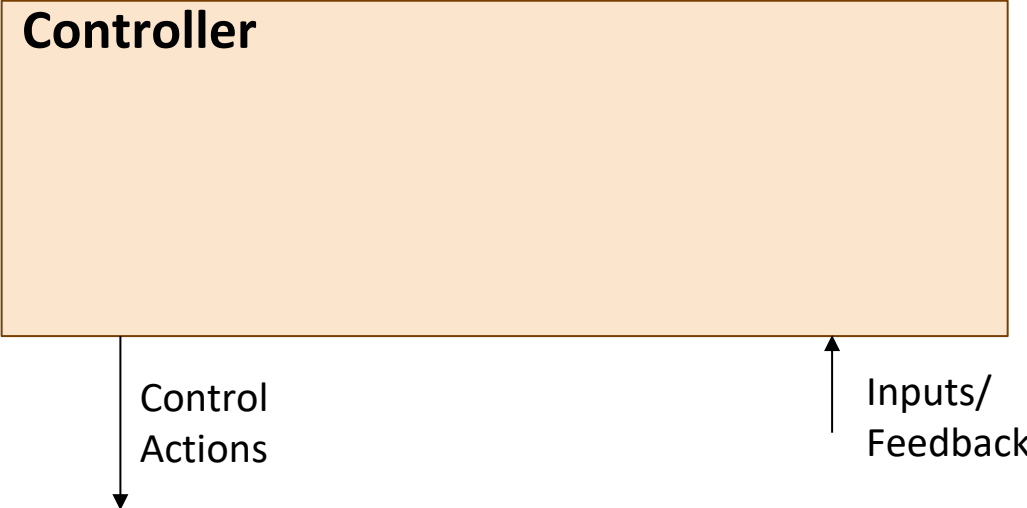
*For some projects, like early concept development or a “lightweight STPA”, option 1 may be enough.*

*If more rigor needed, a generic controller model can be used to rigorously refine the scenarios and find actionable causes.*

Refinement Option 2:

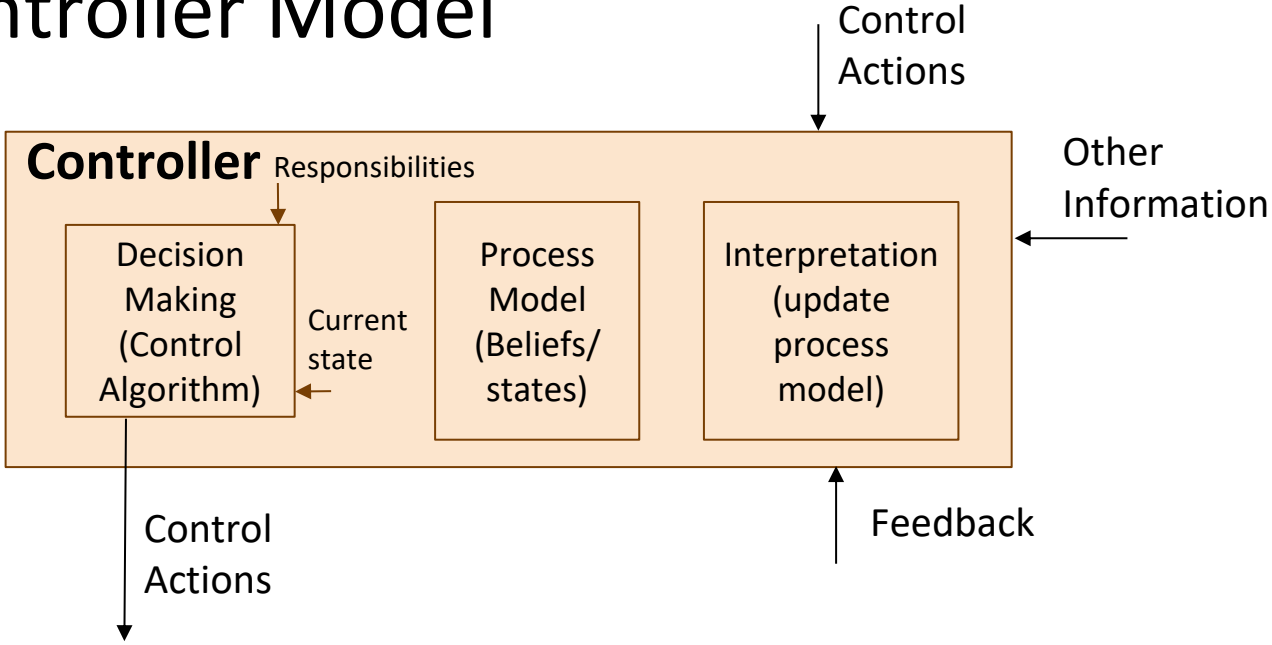
Rigorous (Model-Based)  
Scenario Refinement

# Generic Controller Model



**Class 1 Scenario Archetype:**                      **Output:** UCA                      **Input:** Correctly indicates UCA context (but other inputs may override or conflict)

# Generic Controller Model

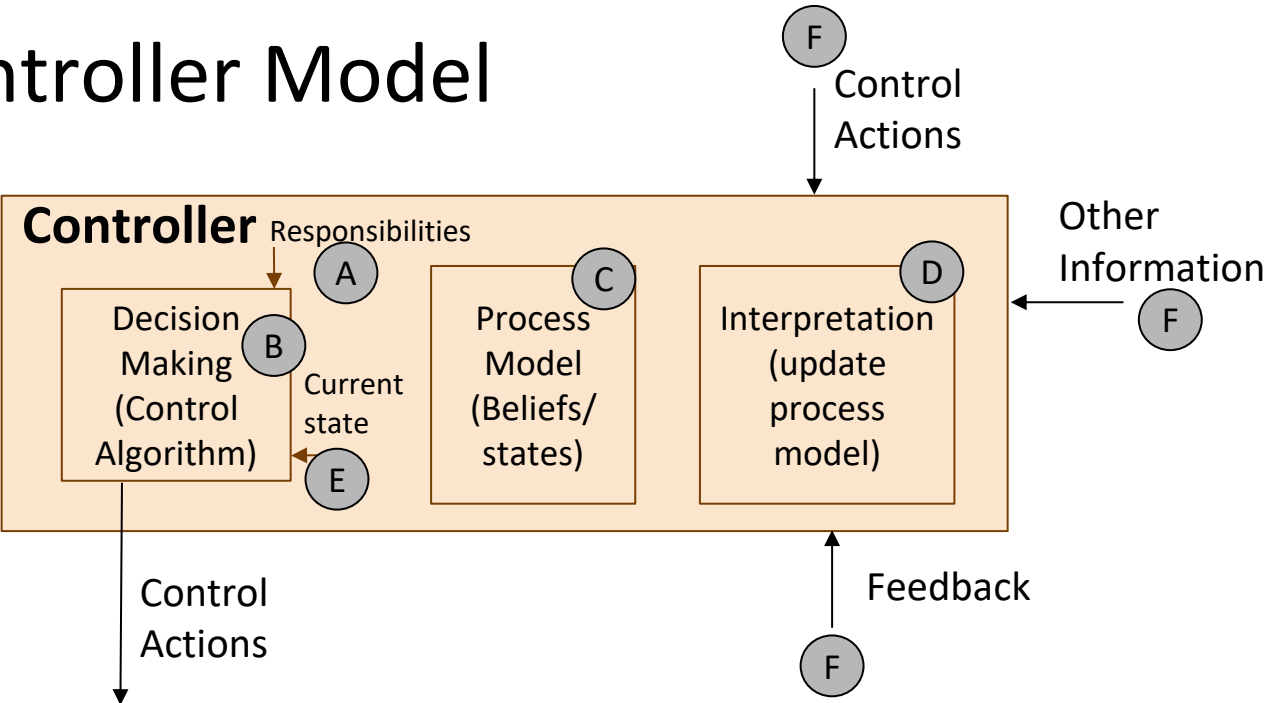


**Class 1 Scenario Archetype:**

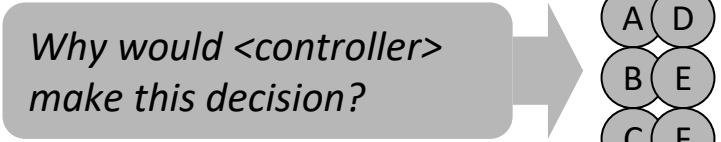
**Output:** UCA

**Input:** Correctly indicates UCA context (but other inputs may override or conflict)

# Generic Controller Model



**Class 1 Scenario Archetype:**      **Output:** UCA      **Input:** Correctly indicates UCA context (but other inputs may override or conflict)



# Refining Class 1 Scenario Archetype: Unsafe Controller Behavior

## Common causes of Scenario Archetype 1:

- A • Responsibilities (e.g., desired end states) that would produce this UCA
- B • Control algorithms or decision-making rationale that would explain the UCA
- C • Process models that would explain this UCA
- D • Interpretation or process model updates that would explain the UCA
- E • Internal controller states/modes that would explain the UCA (failure, lame duck mode, etc.)
- F • Controller inputs (control actions, feedback, or other inputs) that would explain the UCA
  - E.g., Conflicting/contradictory inputs, inputs from another controller, etc.
  - If the input is another UCA, then make sure the new UCA is recorded in STPA Step 3. The new UCA will be analyzed using the same process.
- G • Etc.

# Scenario Archetypes

# Refined Scenarios

UCA-2:

*<Controller> provides <Control Action> when <Context>*

**Class 1 Scenario Archetype:**  
**Unsafe Controller Behavior**

- *<Controller> provides <Control Action> when <Context>*
- *<Input> to <Controller> correctly indicates <Context>*

**Class 2 Scenario Archetype**

**Class 3 Scenario Archetype**

**Class 4 Scenario Archetype**

A

B

C

D

E

F

# Scenario Archetypes

UCA-2:

*<Controller>* provides *<Control Action>* when *<Context>*

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

Examples from real projects:

A

## Responsibilities

*<Controller>* by design is responsible for always assigning \_\_\_\_ to every \_\_\_\_ (even if \_\_\_\_.)

**Discussion:** Responsibilities can be:

- Fixed (hardcoded, embedded in design of algorithm)
- Dynamic (selected in real-time, e.g., an external control action)
- Adaptive (developed and changed by the controller as needed)

# Scenario Archetypes

UCA-2:

*<Controller>* provides *<Control Action>* when *<Context>*

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

Examples from real projects:

## Control Algorithms

*<Controller>* control algorithm is designed to trigger *<Control Action>* during a fallback to \_\_\_\_\_ strategy, which can happen if \_\_\_\_\_

## Decision Making

If \_\_\_\_\_, then *<Controller>* may select *<Control Action>* to \_\_\_\_\_

B

# Scenario Archetypes

UCA-2:

*<Controller> provides <Control Action> when <Context>*

## **Class 1 Scenario Archetype:** **Unsafe Controller Behavior**

- *<Controller> provides <Control Action> when <Context>*
- *<Input> to <Controller> correctly indicates <Context>*

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

# Refined Scenarios

C

## Process Models

PM-1: *<Controller>* incorrectly believes \_\_\_\_\_

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## **Class 1 Scenario Archetype: Unsafe Controller Behavior**

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

C

## Process Model

<Controller>  
believes \_\_\_\_\_

D

## Update Process Model

PM-1 is not updated when \_  
PM-1 is the initial (default)  
belief before feedback/input  
\_\_\_ received

PM-1 is updated incorrectly  
due to feedback/input \_\_\_  
that indicates \_\_\_

PM-1 is updated too late (or  
early) due to \_\_\_

PM-1 stops updating too  
soon before \_\_\_\_\_

PM-1 continues to be  
updated too long after \_\_\_\_\_

**Discussion:** Identifying a process model is usually not enough; must identify *why* it may happen so that it can be prevented or mitigated.

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## **Class 1 Scenario Archetype: Unsafe Controller Behavior**

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

D

## Update Process Model

PM-1 is not updated when \_  
PM-1 is the initial (default)  
belief before feedback/input  
\_\_\_ received

PM-1 is updated incorrectly  
due to feedback/input \_\_\_  
that indicates \_\_\_

PM-1 is updated too late (or  
early) due to \_\_\_

PM-1 stops updating too  
soon before \_\_\_  
PM-1 continues to be  
updated too long after \_\_\_

← Not Provided  
Causes Hazard

← Provided Causes  
Hazard

← Too early /  
too late

← Stopped too soon /  
Applied too long

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## **Class 1 Scenario Archetype: Unsafe Controller Behavior**

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

C

### Process Model

<Controller> believes \_\_\_\_\_

Why?

D

### Update Process Model

<Controller> does not update PM-1 when \_\_\_\_\_

Why?

F

### Other Feedback

<Controller> does not receive <other feedback> when <other context>

Why?

### Unsafe Feedback (Class 2 Scenario)

- <Controller> does not receive <other input/feedback>
- <Other context> is true

# Refined Scenarios

C

## Process Model

<Controller> believes \_\_\_\_\_

Why?

D

## Update Process Model

<Controller> does not update PM-1 when \_\_\_\_\_

Why?

F

## Other Feedback

<Controller> does not receive <other input/feedback> when <other context>

Why?

## Unsafe Feedback (Class 2 Scenario)

- <Controller> does not receive <other input/feedback>
- <Other context> is true

**Discussion:** A class 1 scenario means that an input correctly indicates <context>. No assumption is made about any other inputs. There may be another input/feedback that provides conflicting info or indicates some other overriding context (like an emergency condition).

In these cases, the class 1 scenario refinement should identify the other input(s) that, together with <UCA context>, would trigger the UCA.

**Stopping rule:** The class 1 scenario refinement stops at the boundary of the controller and identifying what feedback/input would cause the UCA. The sources of feedback/inputs are analyzed in class 2 if needed.

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

Examples from real projects:

D

## Interpretation of inputs

In some situations, <Controller> will interpret <feedback/input> as an indicator of \_\_\_\_\_. This interpretation can underestimate \_\_\_\_\_, causing \_\_\_\_\_.

When <feedback/input 1> conflicts with <feedback/input 2>, <Controller> may assume \_\_\_\_\_

When <feedback/input> is not available, <Controller> may assume \_\_\_\_\_

# Scenario Archetypes

UCA-2:

*<Controller>* provides *<Control Action>* when *<Context>*

## **Class 1 Scenario Archetype: Unsafe Controller Behavior**

- *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

# Refined Scenarios

Examples from real projects:

## Controller States / Modes

- If *<Controller>* is in \_\_\_\_ mode/state, it will continue to *<Control Action>* using alternate input \_\_\_\_.
- If \_\_\_\_ is disabled, then *<Controller>* can \_\_\_\_.

E

# Scenario Archetypes

UCA-2:

*<Controller>* provides *<Control Action>* when *<Context>*

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- *<Controller>* provides *<Control Action>* when *<Context>*
- *<Input>* to *<Controller>* correctly indicates *<Context>*

## Class 2 Scenario Archetype

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

Examples from real projects:

## Other Inputs

*<Controller>* does not prevent *<Control Action>* when alternate input \_\_\_ is received.

F

# Scenario Archetypes

UCA-2:

*<Controller> provides <Control Action> when <Context>*

## **Class 1 Scenario Archetype:** **Unsafe Controller Behavior**

- *<Controller> provides <Control Action> when <Context>*
- *<Input> to <Controller> correctly indicates <Context>*

## **Class 2 Scenario Archetype**

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

# Refined Scenarios

Examples from real projects:

## Other inputs

- Although *<Input>* is correct, the feedback from \_\_\_ may be incorrect and may cause *<Controller>* to \_\_\_\_.

F

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

- 4.1) Identify high-level scenarios**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

- 4.3) Identify refined scenarios**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

**Solution Space:**  
What must be done to prevent losses?

- 4.2) Identify high-level solutions**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

- 4.4) Identify refined solutions**
- **Class 1**
  - Class 2
  - Class 3
  - Class 4

#### **4.1) Class 1 Scenario Archetype: Unsafe Controller Behavior**

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air



#### **4.3) Refined Scenario: Update Process Model**

- By design, the Thruster Controller cannot listen to Touchdown Input if a fault is flagged at any time during descent. Will use \_\_\_ instead.

#### **4.4) Identify Refined Solutions**

What can be done to prevent or mitigate this scenario?

#### 4.1) Class 1 Scenario Archetype: Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air

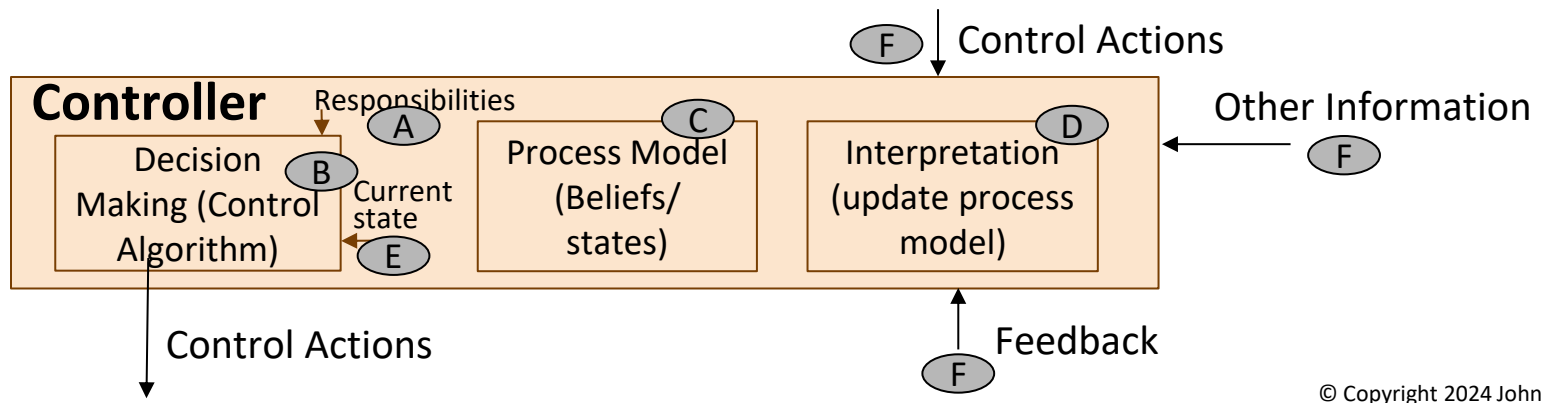


#### 4.3) Refined Scenario: Update Process Model

- By design, the Thruster Controller cannot listen to Touchdown Input if a fault is flagged at any time during descent. Will use \_\_\_ instead.

#### 4.4) Identify Refined Solutions

What can be done to prevent or mitigate this scenario? Consider structural and behavioral mitigations.





#### **4.1) Class 1 Scenario Archetype: Unsafe Controller Behavior**

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air



#### **4.3) Refined Scenario: Update Process Model**

- By design, the Thruster Controller cannot listen to Touchdown Input if a fault is flagged at any time during descent. Will use \_\_\_ instead.

#### **4.4) Identify Refined Solutions**

What can be done to prevent or mitigate this scenario? Consider structural and behavioral mitigations.

- Examples of controller behavior mitigations:
  - Responsibilities: Provide a way to clear/reset/recover from the fault?
  - Update Process Model: Ignore faults when providing Disable Thruster Cmd?
  - Control Algorithm: Instead of ignoring all Touchdown Inputs, drop to 2oo3 logic?
  - Other Inputs: Use Radar Altimeter as a back up if \_\_\_\_\_?
- Other solutions / mitigations?

**Discussion**: Some solutions may be obvious. Others may require revisiting the concept, control structure, or other decisions.

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

- 4.1) Identify high-level scenarios**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

- 4.3) Identify refined scenarios**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

**Solution Space:**  
What must be done to prevent losses?

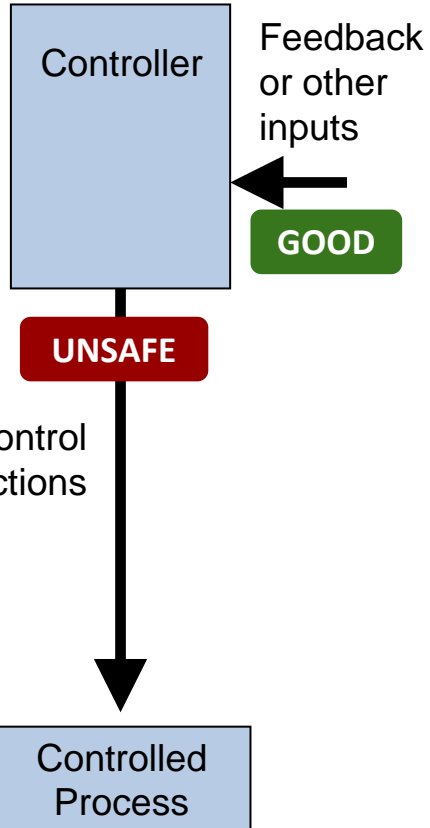
- 4.2) Identify high-level solutions**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

- 4.4) Identify refined solutions**
- ✓ Class 1
  - Class 2
  - Class 3
  - Class 4

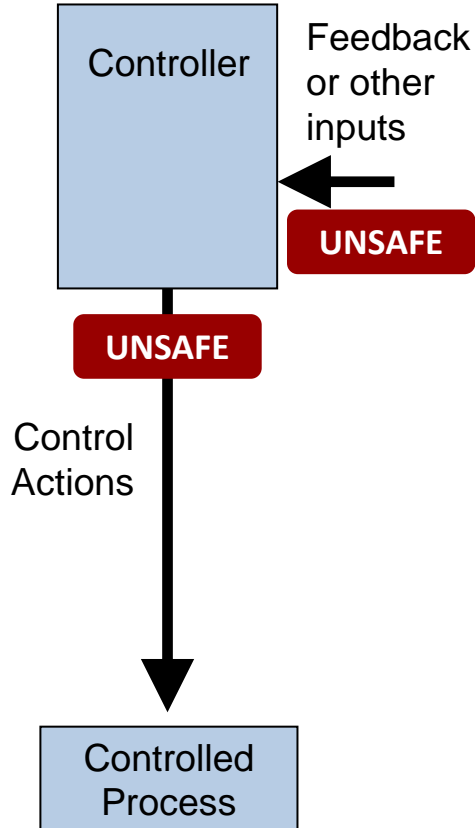
**Done! Let's Do Scenario Class 2!**

# Four Classes of Formal Scenarios

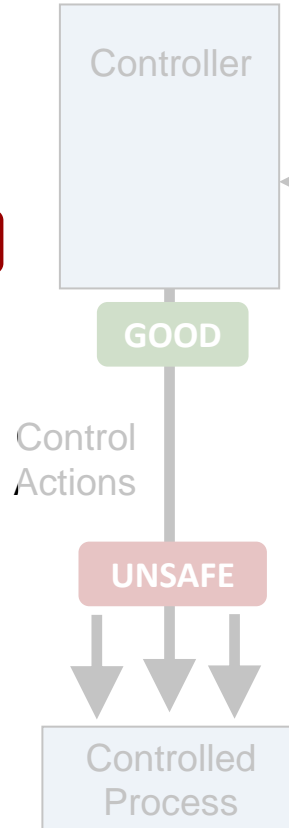
## Class 1



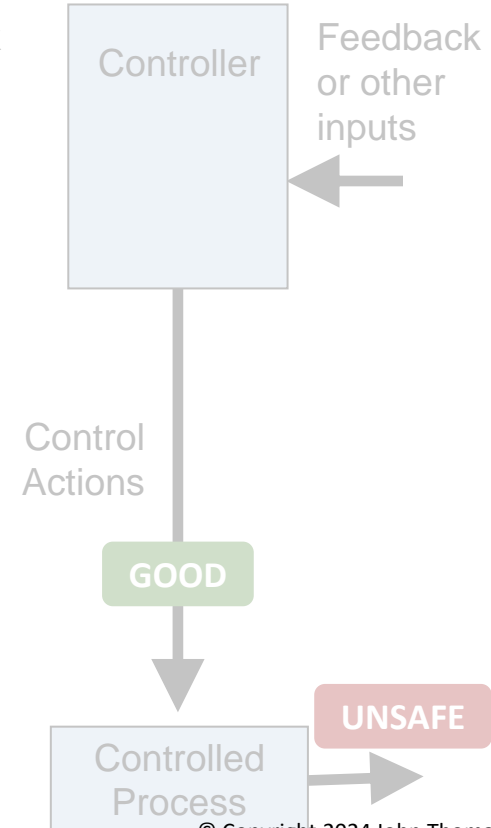
## Class 2



## Class 3

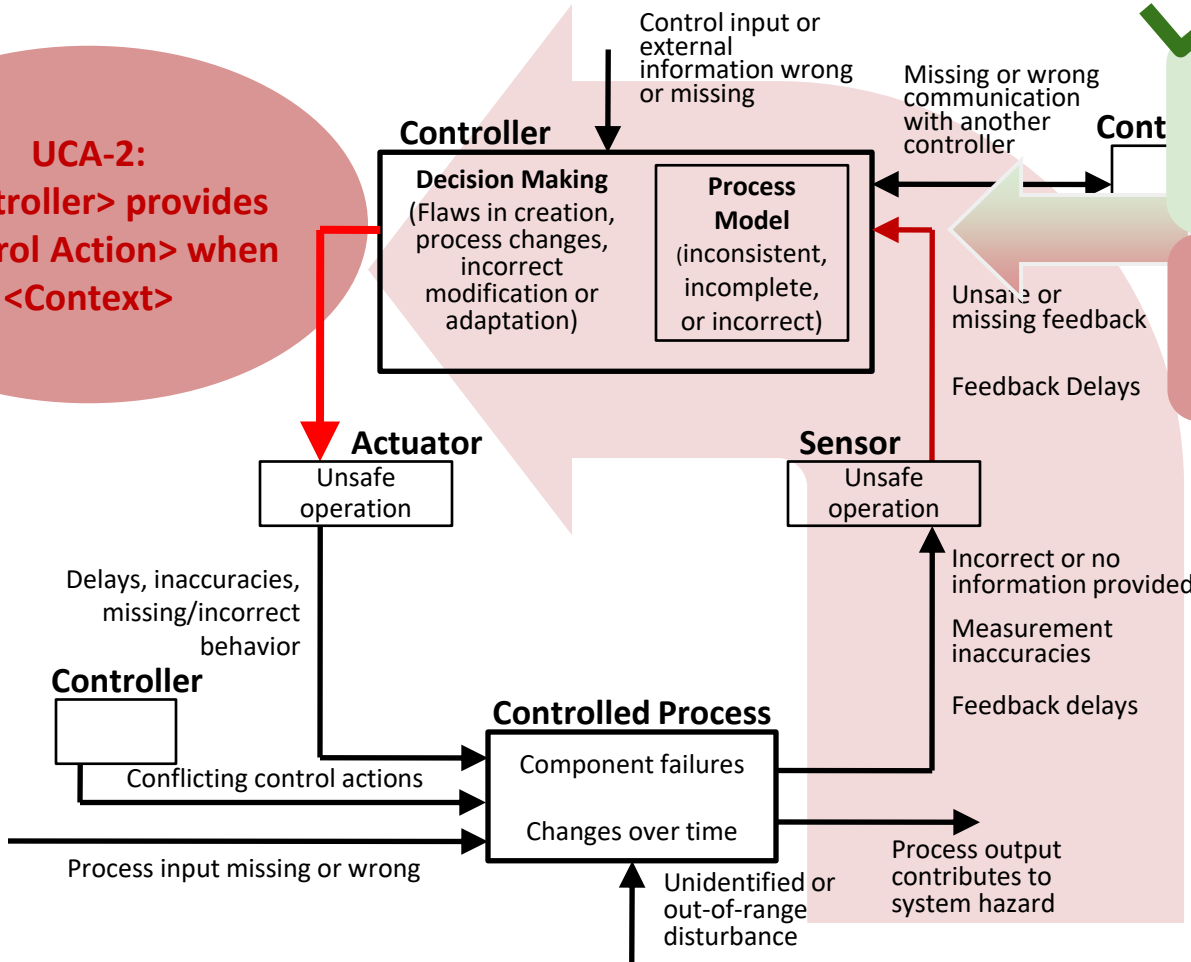


## Class 4



# STPA Step 4A: Identify scenarios that cause UCAs

**UCA-2:**  
**<Controller> provides**  
**<Control Action> when**  
**<Context>**

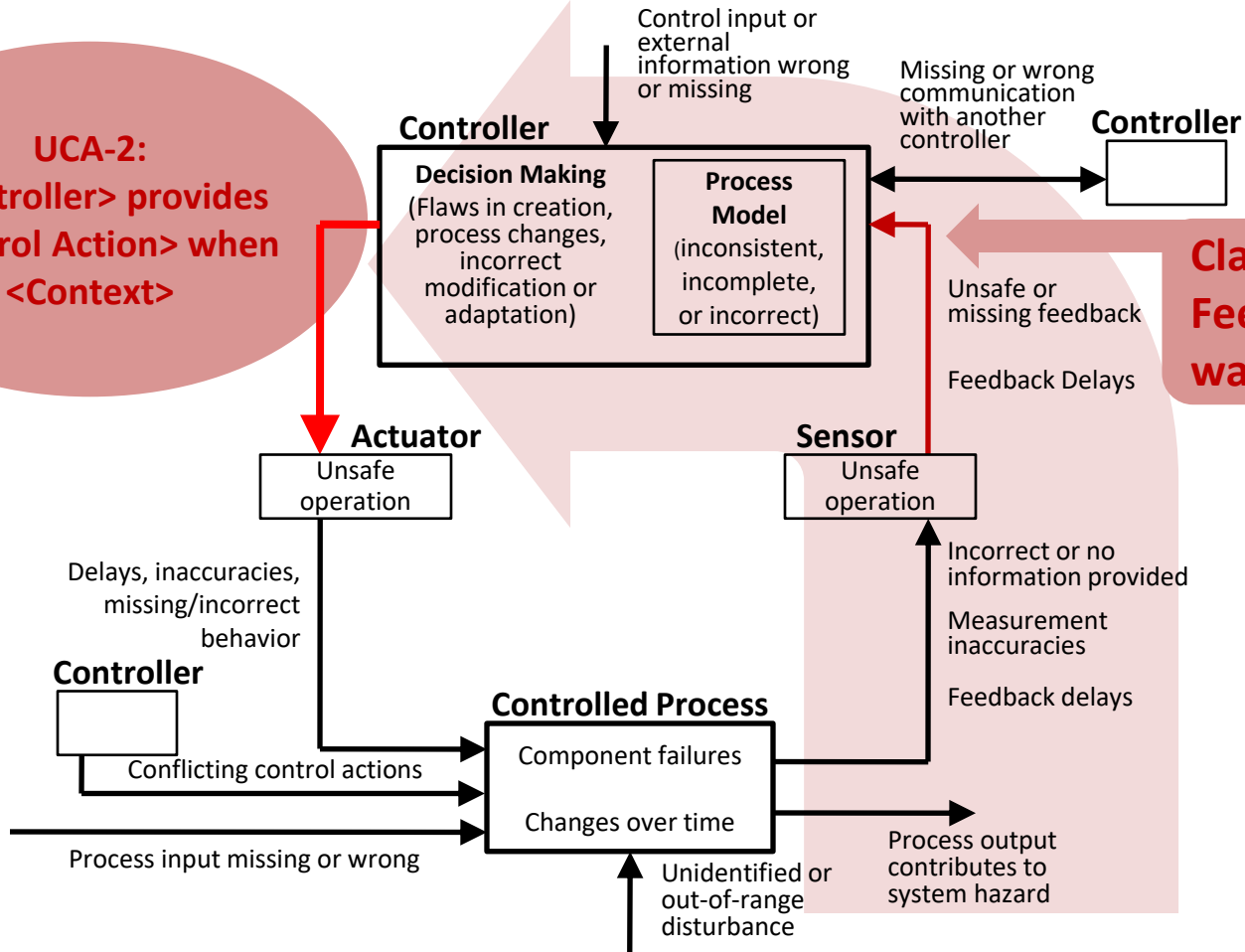


✓  
**Class 1:**  
**Feedback/input \_\_\_\_\_**  
**was adequate**

**Class 2:**  
**Feedback/input \_\_\_\_\_**  
**was unsafe**

# STPA Step 4: Class 2 Scenario Archetype

**UCA-2:**  
**<Controller> provides  
<Control Action> when  
<Context>**



**Class 2:**  
**Feedback/input \_\_\_\_  
was unsafe**

# STPA Step 4: Class 2 Scenario Archetype

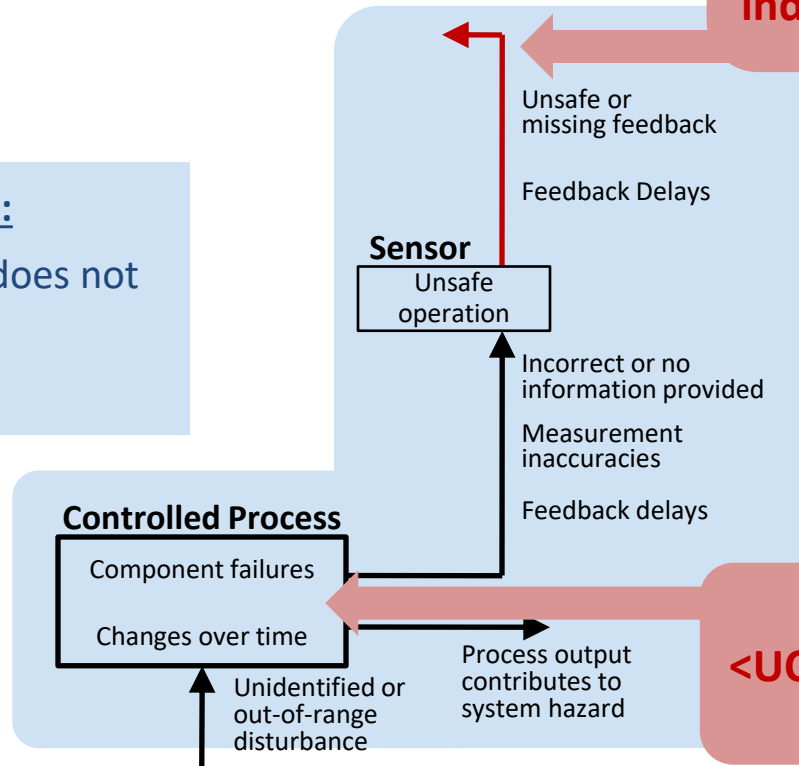
UCA-2:

<Controller> provides  
<Control Action> when  
<Context>

## Class 2 Scenario Archetype:

- Feedback/Input to <Controller> does not adequately indicate <Context>
- <Context> is true

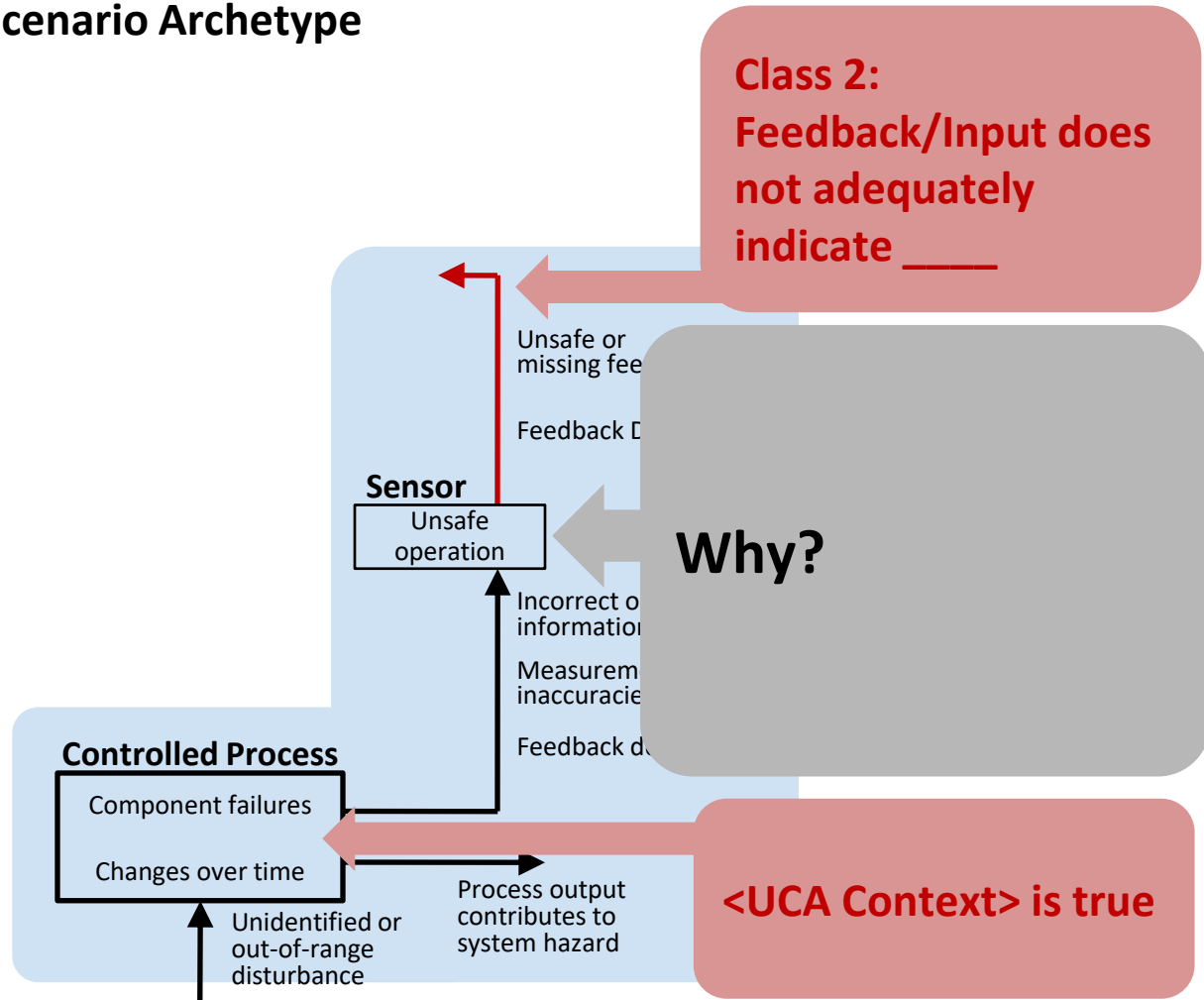
Class 2:  
Feedback/Input does  
not adequately  
indicate \_\_\_\_



<UCA Context> is true

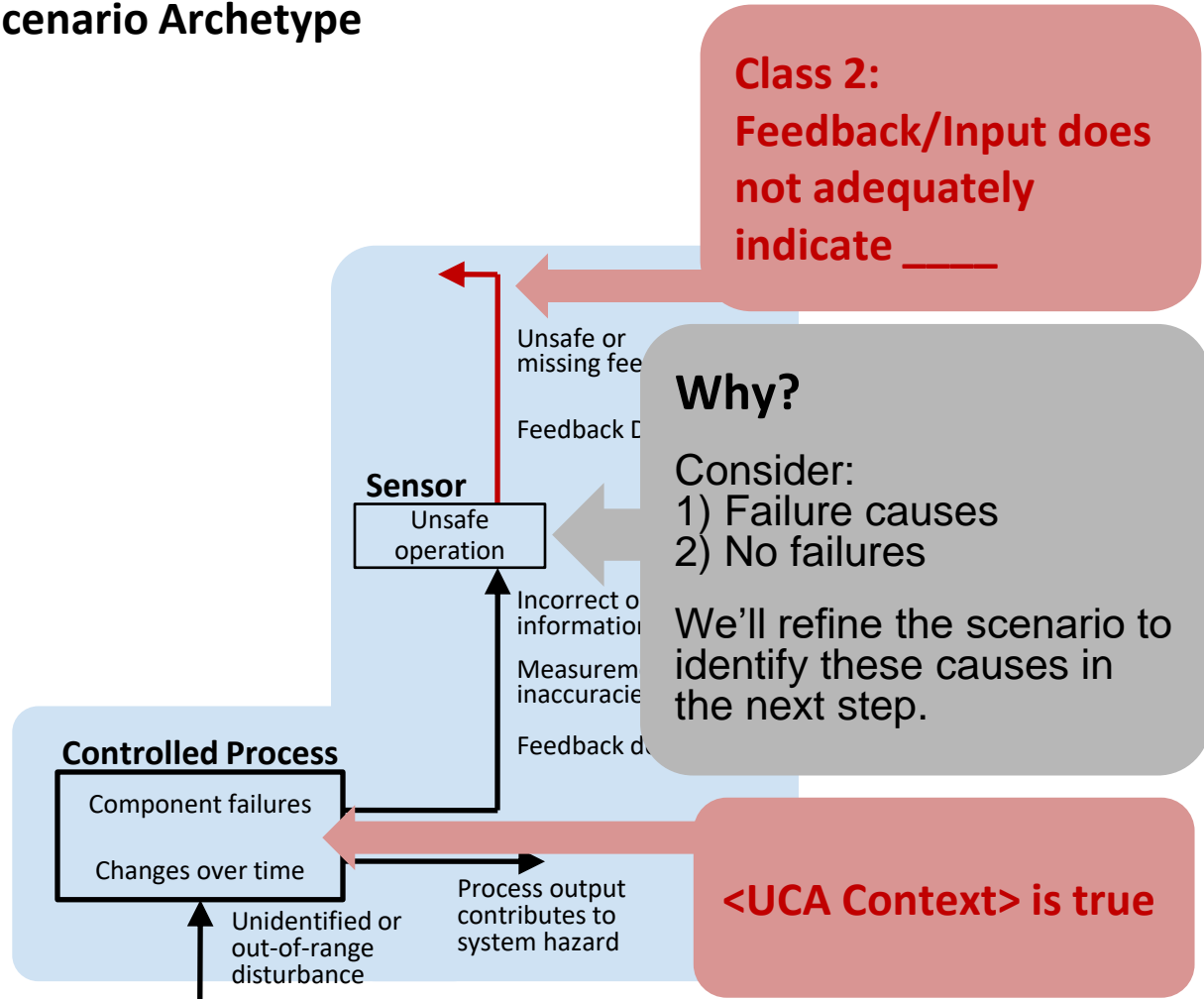
# STPA Step 4: Class 2 Scenario Archetype

**UCA-2:**  
<Controller> provides  
<Control Action> when  
<Context>



# STPA Step 4: Class 2 Scenario Archetype

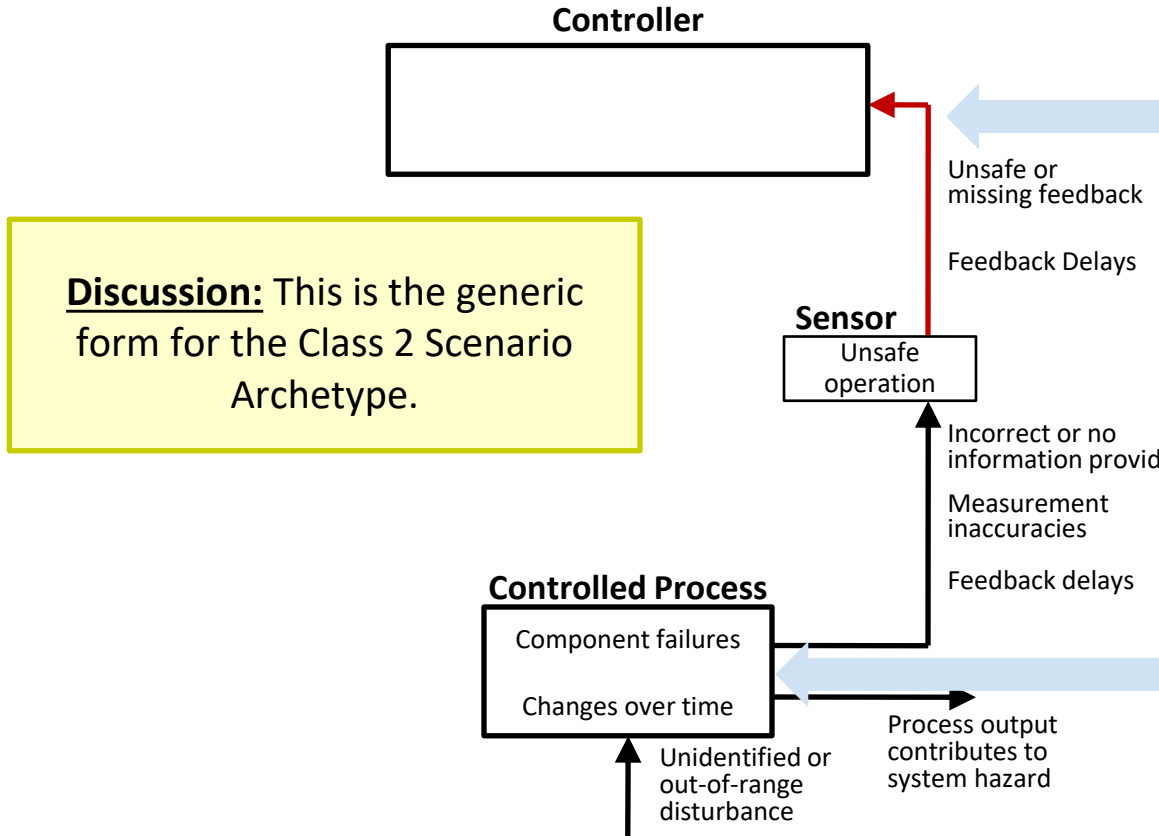
**UCA-2:**  
<Controller> provides  
<Control Action> when  
<Context>



# STPA Step 4: Class 2 Scenario Archetype

UCA-2:

<Controller> provides <Control Action> when <Context>



**Discussion:** This is the generic form for the Class 2 Scenario Archetype.

## Class 2 Scenario Archetype Unsafe Feedback

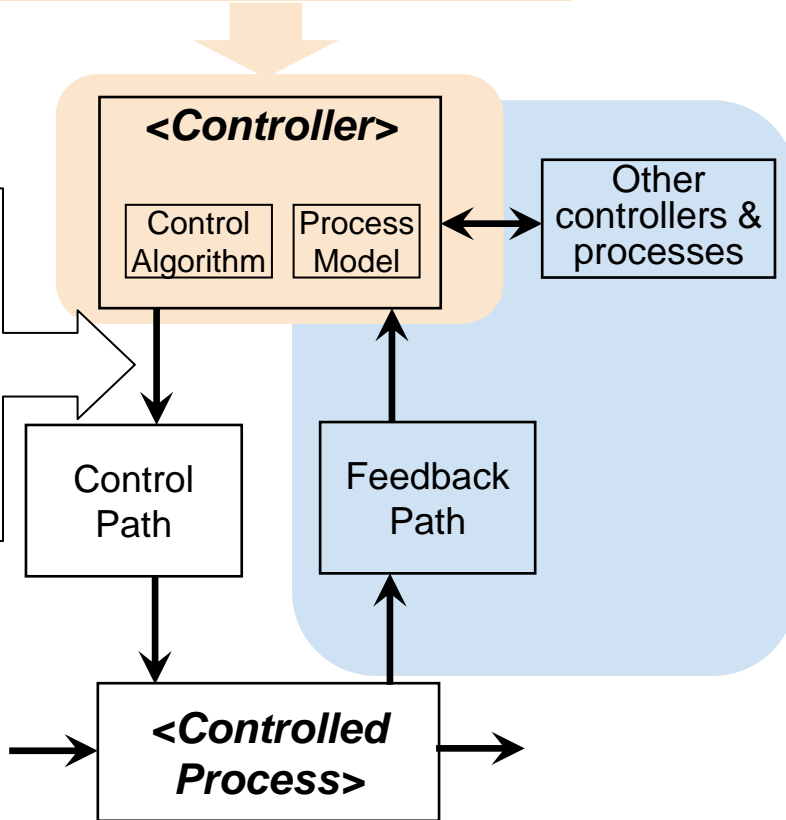
**Output:**  
<Feedback/Input> to <Controller> does not adequately indicate <Context>

**Input:** <Process> is actually <Context>

Class 1 Scenario Archetype:  
Unsafe Controller Behavior

Result from  
previous STPA  
Step 3

UCA:  
<Controller>  
provides  
<Control Action>  
when  
<Context>  
[Hazards]



Class 2 Scenario Archetype:  
Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>



Result from previous STPA Step 3

UCA:

<Controller>

provides

<Control Action>

when

<Context>

[Hazards]

UCA-2:

Thruster Controller

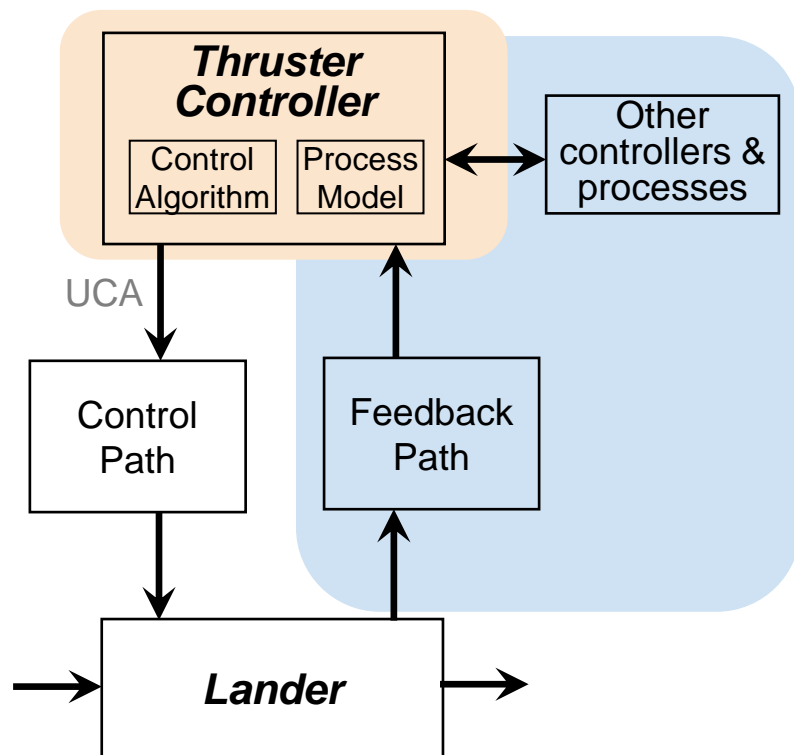
provides

Disable-Thruster Cmd

when

lander is in the air

[H-1]



### Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>



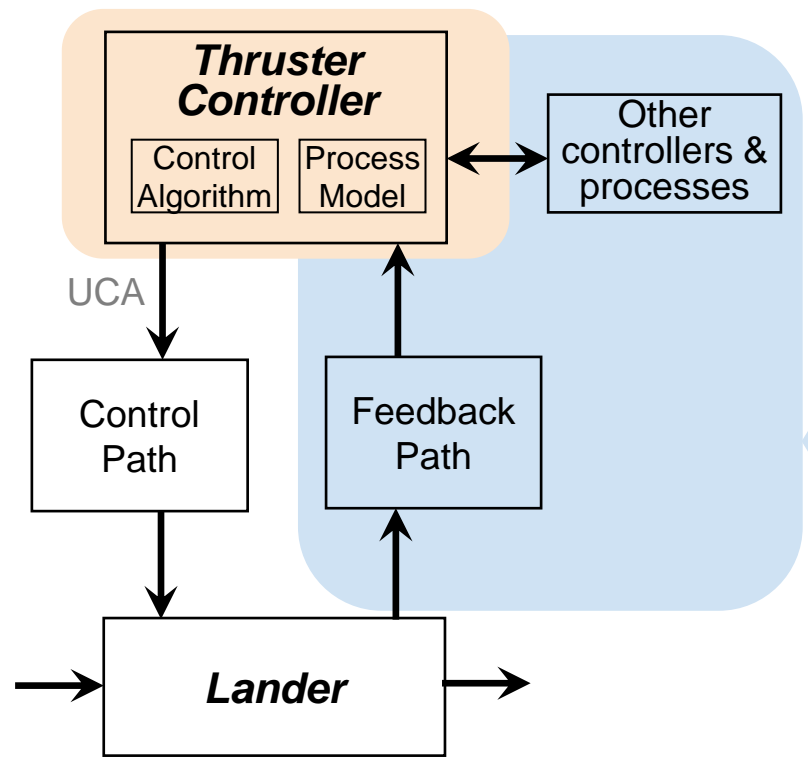
Result from previous STPA Step 3

UCA:

<Controller>  
 provides  
<Control Action>  
 when  
<Context>  
[Hazards]

UCA-2:

Thruster Controller  
 provides  
Disable-Thruster Cmd  
 when  
lander is in the air  
[H-1]



**Class 2 Scenario Archetype:**  
**Unsafe Feedback/Information**

- Touchdown feedback does not indicate lander in air
- <Process> is actually <Context>



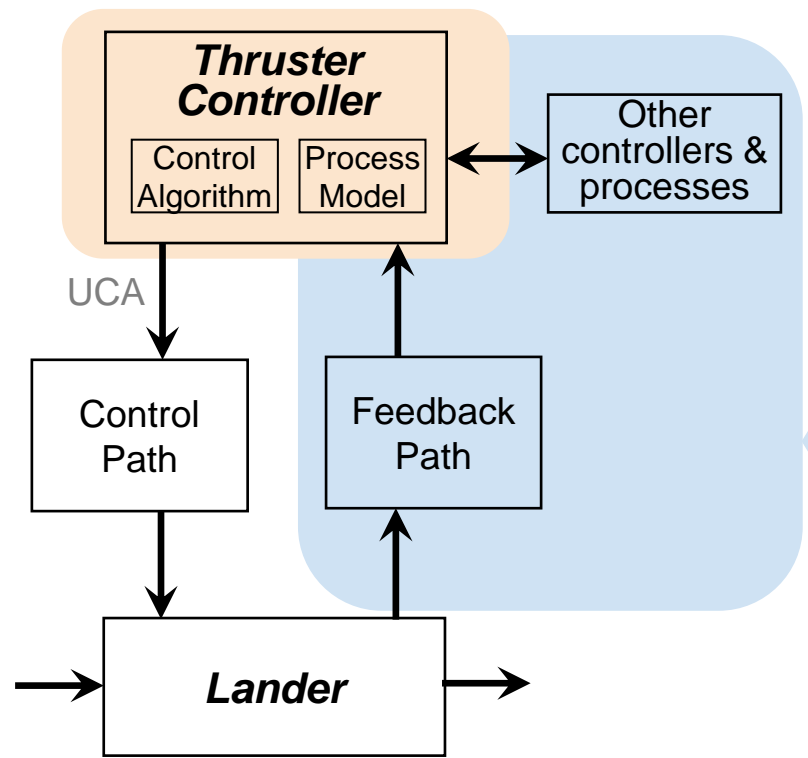
Result from previous STPA Step 3

UCA:

<Controller>  
 provides  
<Control Action>  
 when  
<Context>  
[Hazards]

UCA-2:

Thruster Controller  
 provides  
Disable-Thruster Cmd  
 when  
lander is in the air  
[H-1]



**Class 2 Scenario Archetype:**  
**Unsafe Feedback/Information**

- Touchdown feedback does not indicate lander is in the air
- Lander is actually in the air

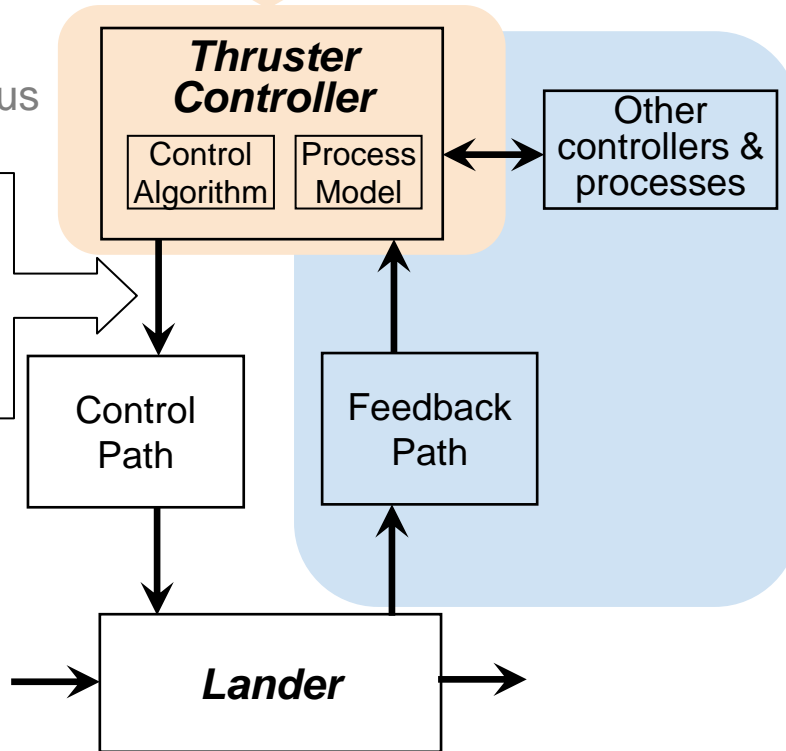


## Class 1 Scenario Archetype: Unsafe Controller Behavior

- UCA: Thruster Controller provides Disable-Thruster Cmd when lander is in the air
- Touchdown Input to Thruster Controller correctly indicates lander is in the air

Result from previous  
STPA Step 3

UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]



## Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in the air
- Lander is actually in the air

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

## 4.1) Identify high-level scenarios

- Class 1
- ✓ - Class 2
- Class 3
- Class 4

## 4.3) Identify refined scenarios

- Class 1
- Class 2
- Class 3
- Class 4

**Solution Space:**  
What must be done to prevent losses?

## 4.2) Identify high-level solutions

- Class 1
- **Class 2**
- Class 3
- Class 4

## 4.4) Identify refined solutions

- Class 1
- Class 2
- Class 3
- Class 4

## 4.2) Identify High-level Solutions

What is needed from each part of the control structure to prevent the scenario? For example:

Structural mitigations

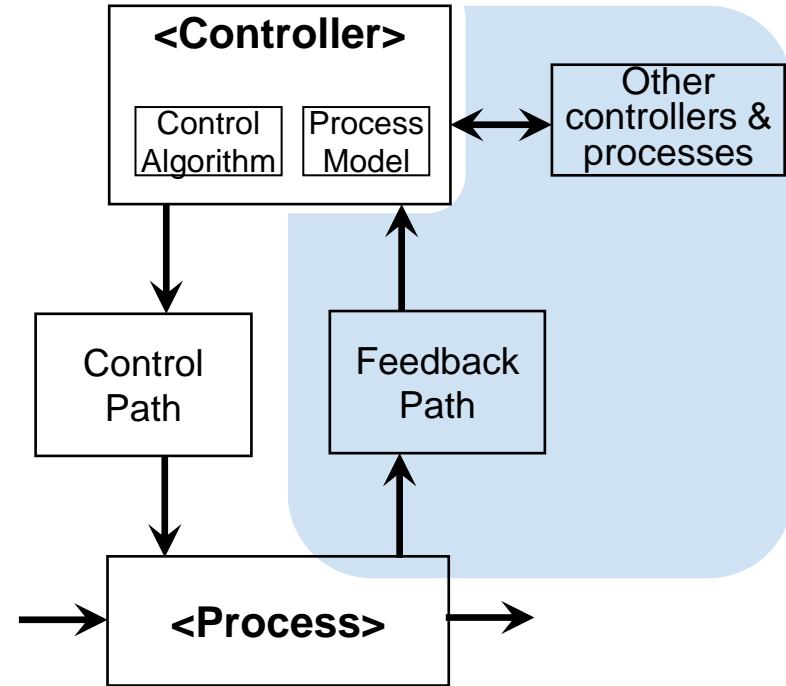
- Add/remove control actions?
- Add/remove controllers?
- Add/remove feedback?
- Add/remove processes?

Behavioral mitigations

- <Controller> must \_\_\_\_ when \_\_\_\_
- <Feedback> must \_\_\_\_ when \_\_\_\_
- <Controller> is/is not responsible for \_\_\_\_
- <Controller> Control Algorithm must \_\_\_\_
- <Controller> Process Model must include \_\_\_\_  
(e.g., make controller aware of some other state)
- <Process> must \_\_\_\_

Others?

**Discussion:** Solutions are not necessarily limited to any one part of the control structure, like the feedback path.



## 4.1)

### Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in air
- Lander is actually in the air

## 4.2) Identify High-level Solutions

What can be done to prevent or mitigate this scenario?

- Add Feedback:
  - Use alternate indication, such as the radar altimeter ?
- Add Controller Responsibility:
  - Software must filter out touchdown indications <TBD ms ?
- Other solutions / mitigations?



# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

- 4.1) Identify high-level scenarios**
- Class 1
  - ✓ - Class 2
  - Class 3
  - Class 4

- 4.3) Identify refined scenarios**
- Class 1
  - **Class 2**
  - Class 3
  - Class 4

**Solution Space:**  
What must be done to prevent losses?

- 4.2) Identify high-level solutions**
- Class 1
  - ✓ - Class 2
  - Class 3
  - Class 4

- 4.4) Identify refined solutions**
- Class 1
  - Class 2
  - Class 3
  - Class 4

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

*Ask: What can cause this?*

Consider:

- 1) Failure causes
- 2) Causes without failure

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control  
Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype

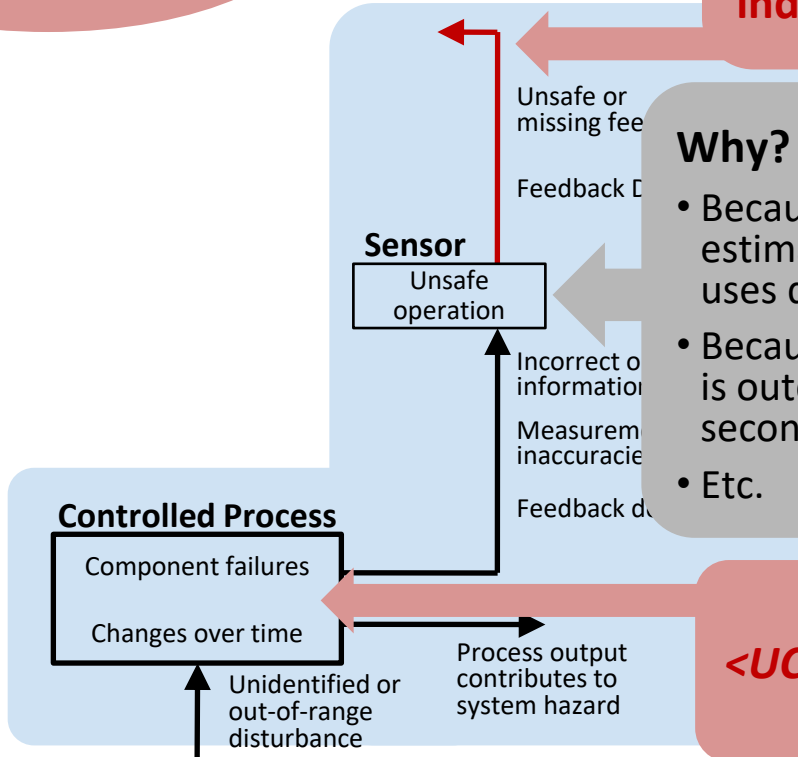
## Class 4 Scenario Archetype

***Refinement Option 1: Ask  
SMEs to help identify causes.***

# STPA Step 4: Class 2 Scenario Archetype

**UCA-2:**  
<Controller> provides  
<Control Action> when  
<Context>

**Class 2:**  
<Feedback/Input>  
does not adequately  
indicate <Context>



**Why? (non-failures)**

- Because the \_\_\_ metric is estimated from \_\_\_ that uses different \_\_\_
- Because the \_\_\_ feedback is outdated due to \_\_\_ second feedback delay
- Etc.

**<UCA Context> is true**

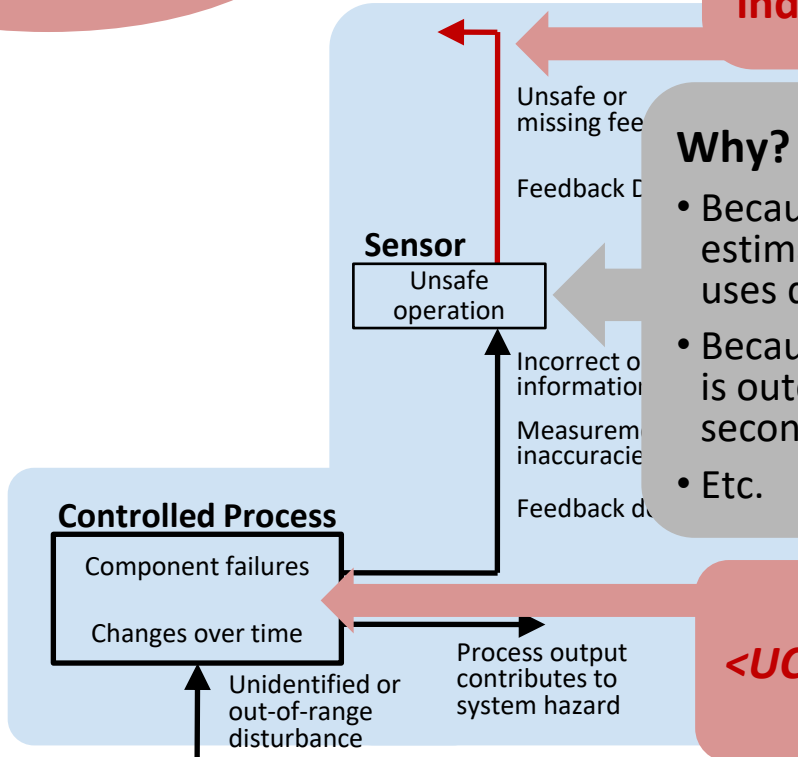
Example SMEs responses from a real project.

Time required: under 15 minutes

# STPA Step 4: Class 2 Scenario Archetype

**UCA-2:**  
<Controller> provides  
<Control Action> when  
<Context>

**Class 2:**  
<Feedback/Input>  
does not adequately  
indicate <Context>



**Why? (non-failures)**

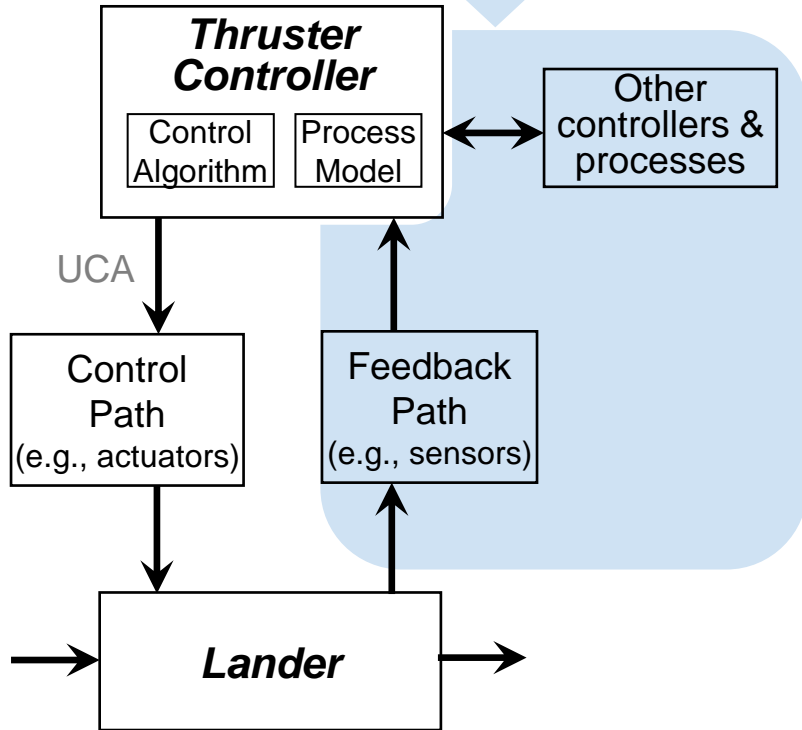
- Because the \_\_\_ metric is estimated from \_\_\_ that uses different \_\_\_
- Because the \_\_\_ feedback is outdated due to \_\_\_ second feedback delay
- Etc.

**Discussion:** The “Why” answers may come from SMEs, not necessarily the STPA practitioner. You may not be an expert in the system. The point is for you to use this framework to ask the right questions for the team or SMEs to answer.

**<UCA Context> is true**

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in air
- Lander is actually in the air



## Refined Scenarios

What can cause this?

- 1) Failure causes
- 2) **Causes without failure**



To refine non-failure scenarios, it can help to think through the feedback transfer function without a failure.

What does “without failure” mean?

- It means the sensors, etc., technically worked as specified

What does “touchdown sensor worked” mean?

- It means the sensor output was as specified for the sensor input

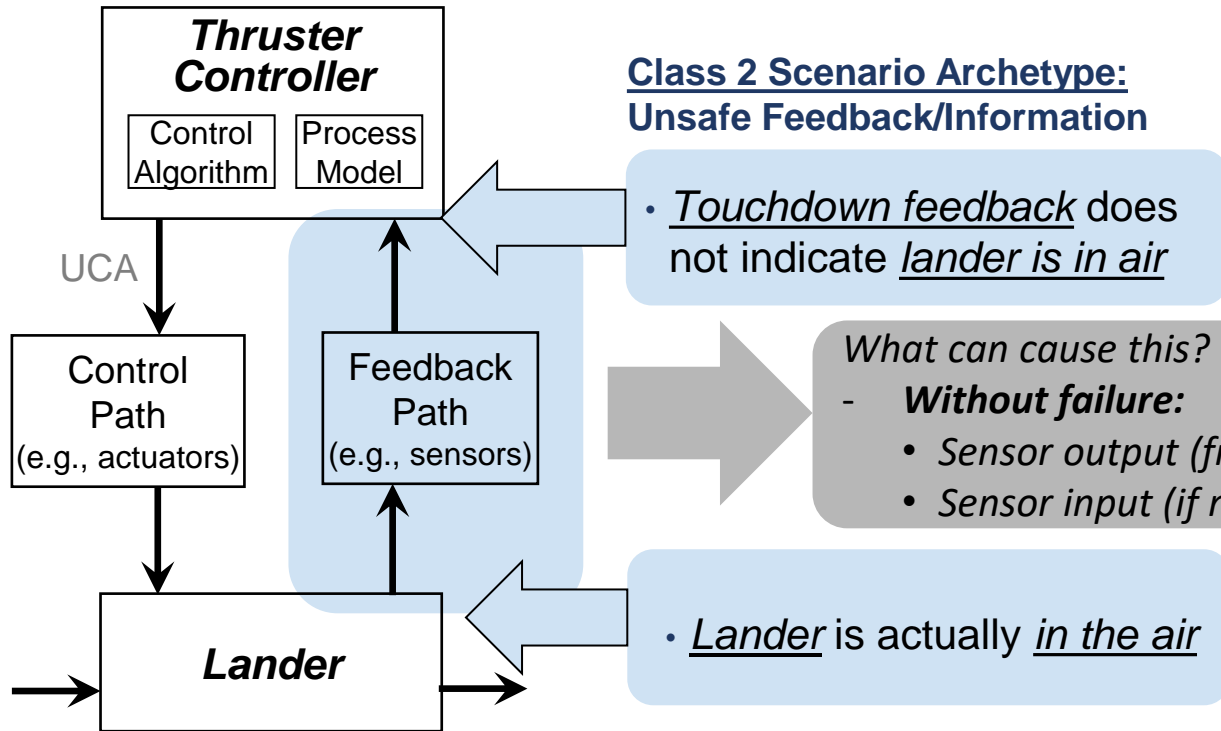
From the class 2 scenario archetype, we know the feedback to controller (sensor output) was NOT [IN AIR].

What is the input to the feedback path if there is technically no feedback failure?

- If output = [NOT IN AIR],  
then input = [VIBRATION > X]



UCA-2: Thruster Controller provides  
Disable-Thruster Cmd when lander is in the air [H-1]



## Refined Scenarios

*What can cause this?*

- **Without failure:**

- Sensor output (from scenario) = TOUCHDOWN TRUE
- Sensor input (if not failed) = VIBRATION > X



UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]

**Class 2 Scenario Archetype:  
Unsafe Feedback/Information**

• Touchdown feedback does not indicate lander is in air

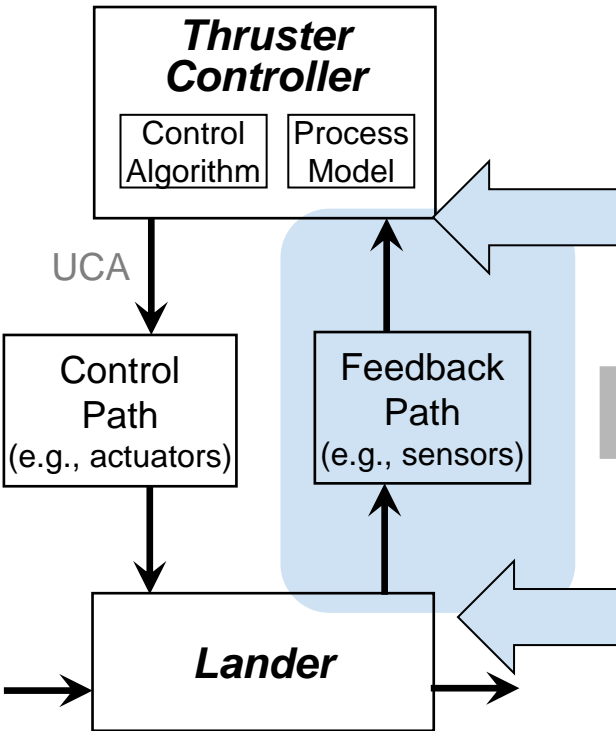
## Refined Scenarios

What can cause this?

- **Without failure:**

- Sensor output (from scenario) = TOUCHDOWN TRUE
- Sensor input (if not failed) = VIBRATION > X

• Lander is actually in the air



AHA! If Lander is actually in the air AND VIBRATION > X, it will cause UCA-2:  
Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]

UCA-2: Thruster Controller provides  
Disable-Thruster Cmd when lander is in the air [H-1]



### Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in air

*What can cause this?*

- **Without failure:**

- *Sensor output (from scenario) = TOUCHDOWN TRUE*
- *Sensor input (if not failed) = VIBRATION > X*

## Refined Scenarios

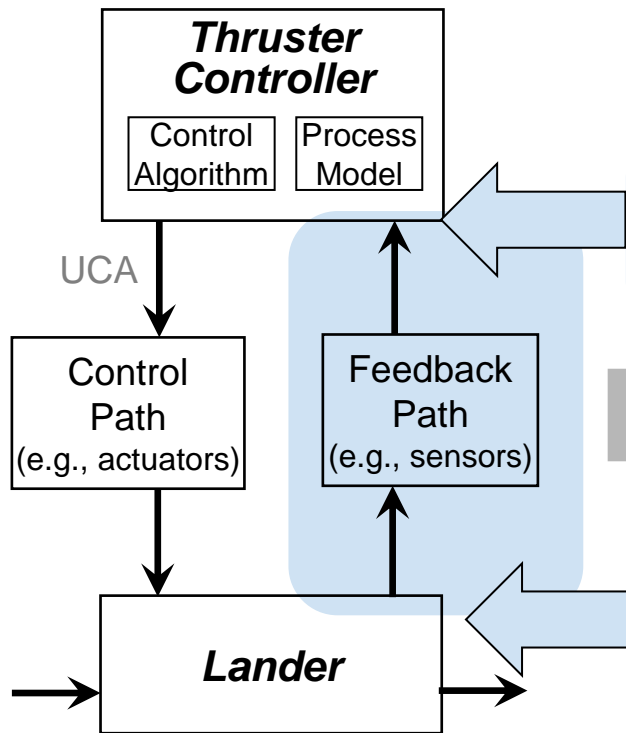
- Lander is actually in the air

### **Question to SMEs:**

How can we get  
VIBRATION > X when  
Lander is actually in the air?

Consider:

- 1) Failure causes
- 2) Causes without failure



AHA! If  AND , then it will cause UCA-X and  $[H-1]$ !

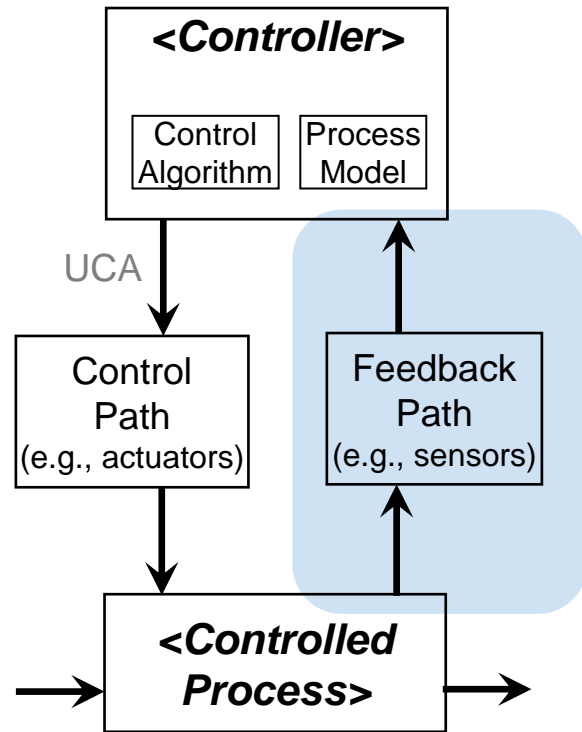
Question for SMES:

What could cause  when  ?

Consider:

- 1) *Failure Causes*
- 2) *Causes without failure*

What solutions can prevent these scenarios?



**Discussion:** In industry trials, this approach during STPA Step 4 has been extremely effective. It identified many real-world catastrophic non-failure scenarios missed by standard approaches, including:

- Catastrophic flaws in real aircraft control systems (caught by this STPA process after certification but just before operation)
- Safety-related deficiencies in real nuclear power plant control systems overlooked during the safety assessment and licensing
- Flaws in real automotive L2, L3, and L4 systems
- Flaws that would trigger large-scale outages in production online/IT software systems (found by STPA ~1 week prior)

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

# Refined Scenarios

***Refinement Option 1: Ask SMEs to help identify causes.***

***Is there a more rigorous option?***

*For some projects, like early concept development or a “lightweight STPA”, option 1 may be enough.*

*If more rigor needed, a generic feedback model can be used to rigorously refine the scenarios and find causes.*

## STPA Step 4: Class 2 Scenario Archetype

## Refined Scenarios

**UCA-2:**

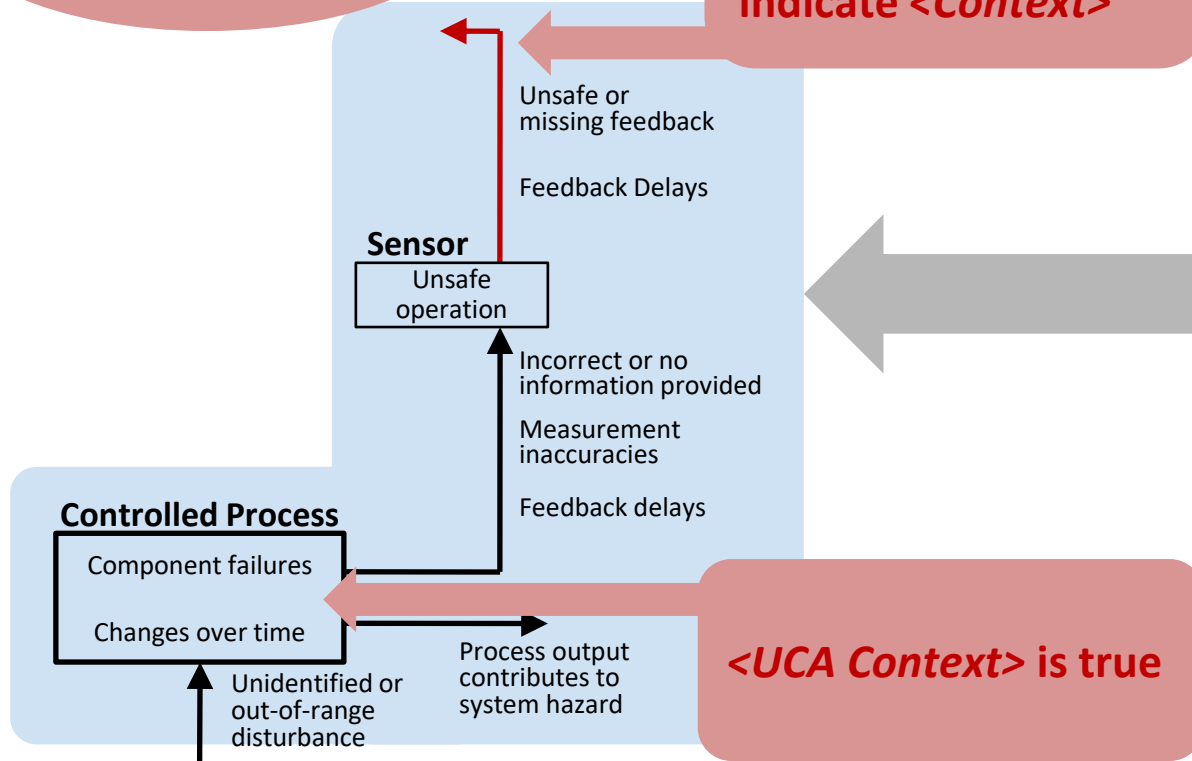
*<Controller> provides  
<Control Action> when  
<Context>*

**Class 2:**

*<Feedback/Input>  
does not adequately  
indicate <Context>*

**Why? Common Causes:**

- Feedback/info missing from design/concept
- Feedback/info not provided
- Conflicting feedback/info
- Incorrect feedback/info provided
- Too early or too late (delayed) feedback/info
- Measurement inaccuracies
- Dropouts
- Corruption
- Content incomplete
- Feedback/info provided in a way the controller can't use
- Overloaded or too much feedback/info
- Etc.



# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control  
Action> when <Context>

## **Class 1 Scenario Archetype: Unsafe Controller Behavior**

## **Class 2 Scenario Archetype: Unsafe Feedback/Information**

- <Feedback/Input> to <Controller>  
does not adequately indicate  
<Context>
- <Process> is actually <Context>

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

### Unsafe Feedback or Inputs

Feedback/input \_\_\_ is (or  
is not) provided when \_\_\_

Feedback/input \_\_\_ is  
provided when \_\_\_

Feedback/input \_\_\_ is  
delayed (or too early) when  
\_\_\_

Feedback/input \_\_\_ is  
applied too long after  
(stopped too soon before)  
\_\_\_\_\_

Not Provided  
Causes Hazard

Provided Causes  
Hazard

Too early /  
too late

Stopped too soon /  
Applied too long

# Scenario Archetypes

UCA-2:

<Controller> provides <Control  
Action> when <Context>

## **Class 1 Scenario Archetype:** **Unsafe Controller Behavior**

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## **Class 2 Scenario Archetype:** **Unsafe Feedback/Information**

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## **Class 3 Scenario Archetype**

## **Class 4 Scenario Archetype**

# Refined Scenarios

## Too early or too late info

<Feedback> may be sent too early  
before \_\_\_\_ has occurred.

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control  
Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

No info provided

<Feedback> is not provided  
when \_\_\_\_ is initialized, reset, or  
on power up.

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype

## Class 4 Scenario Archetype

### Overloaded or too much info

If <Process> is overloaded, then there might not be any indication that <Context>

**Discussion:** These scenarios are used to help us think about mitigations. E.g., <Controller> currently has no feedback to indicate <process> is overloading, so the design doesn't currently have any way to correct it.

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

## 4.1) Identify high-level scenarios

- Class 1
- ✓ Class 2
- Class 3
- Class 4

## 4.3) Identify refined scenarios

- Class 1
- ✓ **Class 2**
- Class 3
- Class 4

**Solution Space:**  
What must be done to prevent losses?

## 4.2) Identify high-level solutions

- Class 1
- ✓ Class 2
- Class 3
- Class 4

## 4.4) Identify refined solutions

- Class 1
- **Class 2**
- Class 3
- Class 4

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in air
- Lander is actually in the air



### **Refined Scenario:**

If Lander is actually in the air and VIBRATION > X, then it will cause UCA-2: Thruster Controller provides Disable-Thruster Cmd [H-1].

What can cause footpad vibration in the air (non-failures):

- Leg deployment during descent
- Mechanical forces from parachute deployment
- Etc.

### **4.4) Identify Refined Solutions**

What can be done to prevent or mitigate this scenario?

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- Touchdown feedback does not indicate lander is in air
- Lander is actually in the air



### **Refined Scenario:**

If Lander is actually in the air and VIBRATION > X, then it will cause UCA-2: Thruster Controller provides Disable-Thruster Cmd [*H-1*].

What can cause footpad vibration in the air (non-failures):

- Leg deployment during descent
- Mechanical forces from parachute deployment
- Etc.

### **4.4) Identify Refined Solutions**

What can be done to prevent or mitigate this scenario? Consider structural and behavioral mitigations.

- Controller mitigations:
  - Responsibilities: Thrust Controller must not monitor touchdown indications prior to TBD ?
  - Control Algorithm: Thrust Controller must filter transient touchdown indications <TBD ms ?
- Feedback mitigations:
  - Sensors: Touchdown sensors must rely on a method other than vibration for touchdown indication ?
- Others?

# STPA: Building Scenarios

**Problem Space:**  
What can go wrong?

- 4.1) Identify high-level scenarios**
- ✓ Class 1
  - ✓ Class 2
  - Class 3
  - Class 4

- 4.3) Identify refined scenarios**
- ✓ Class 1
  - ✓ Class 2
  - Class 3
  - Class 4

**Solution Space:**  
What must be done to prevent losses?

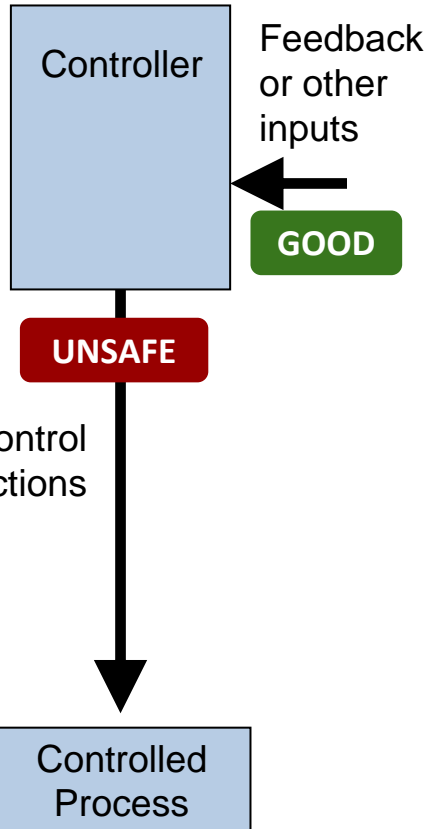
- 4.2) Identify high-level solutions**
- ✓ Class 1
  - ✓ Class 2
  - Class 3
  - Class 4

- 4.4) Identify refined solutions**
- ✓ Class 1
  - ✓ Class 2
  - Class 3
  - Class 4

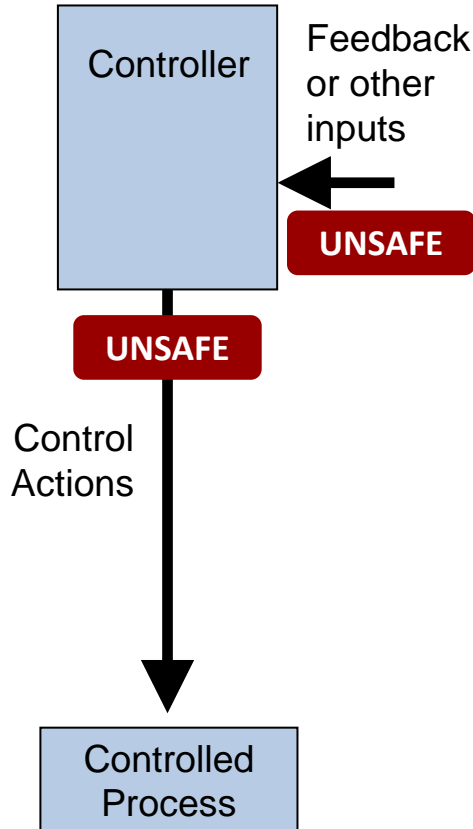
**Done! Same pattern for classes 3-4...**

# Four Classes of Formal Scenarios

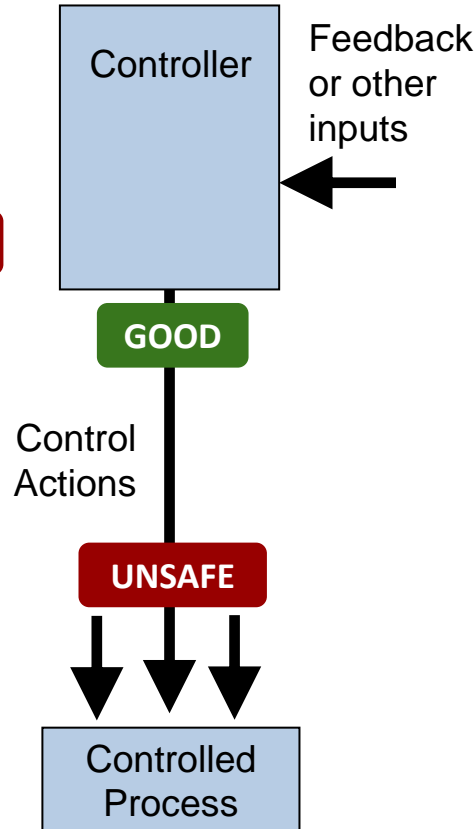
## Class 1



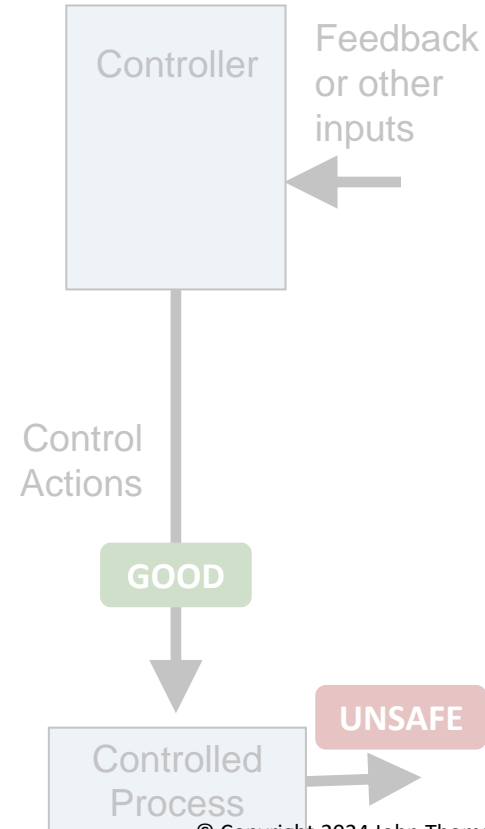
## Class 2



## Class 3



## Class 4



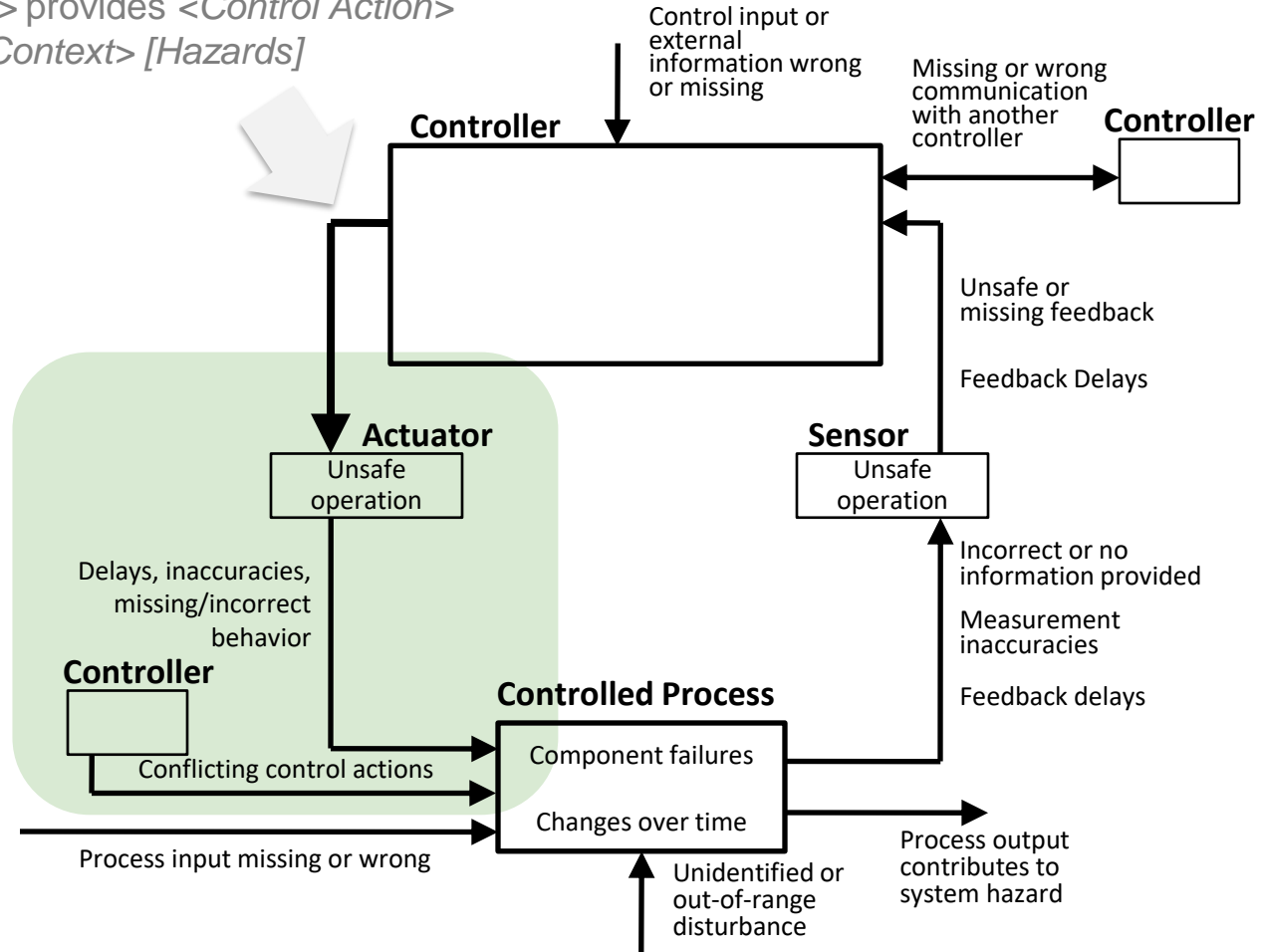
**Result from previous STPA Step 3**  
UCA: <Controller> provides <Control Action>  
when <Context> [Hazards]

### Class 3 Scenario Archetype: Unsafe Control Execution

We need to look at how the UCA can be *emulated* due to interactions involving the process's control path.

Constructing Scenario Archetype 3:

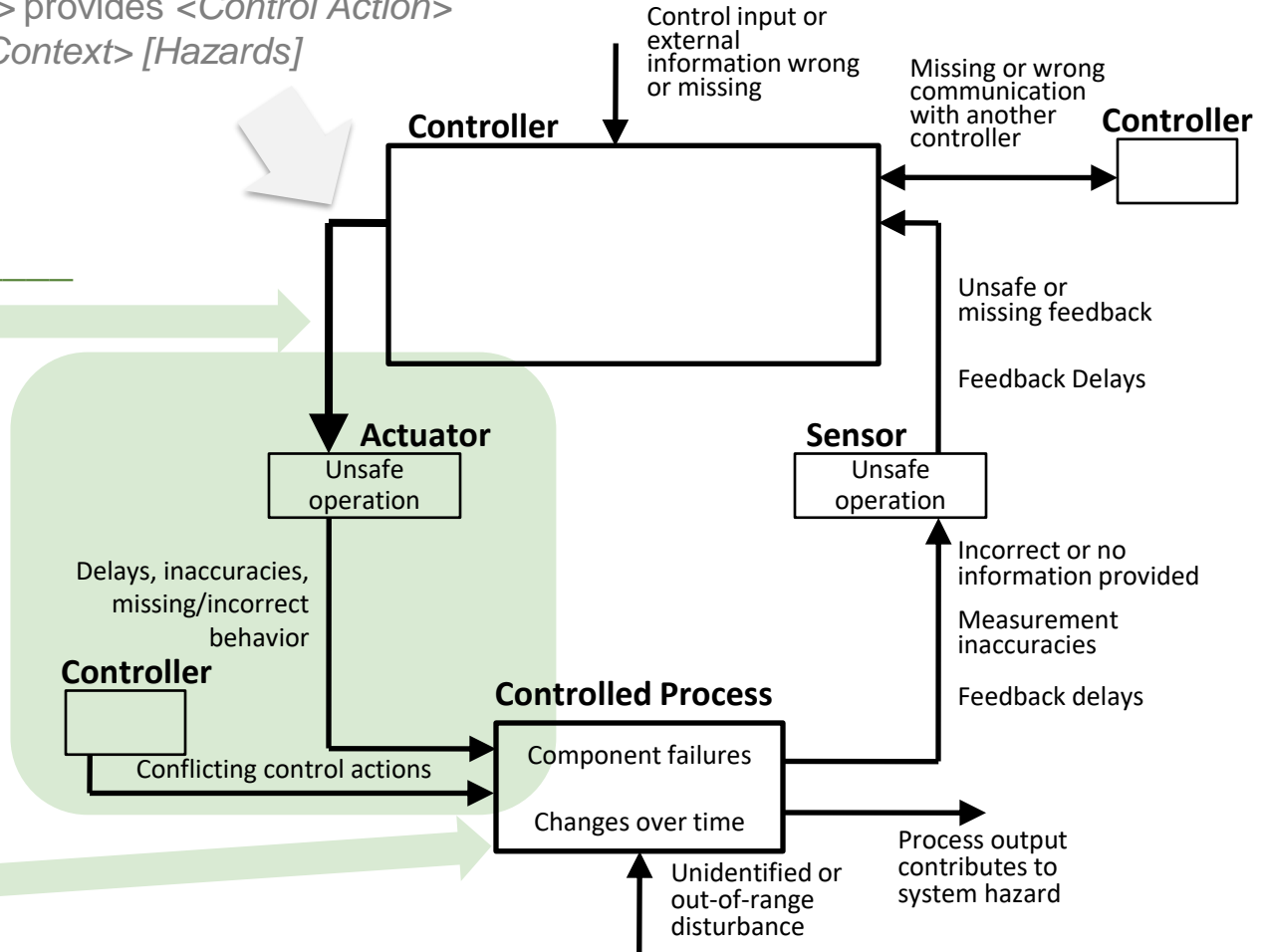
- Suppose the UCA does not happen (invert the UCA). The controller provided a “safe” control action.
- BUT... something happened on the control path making it as if the UCA had occurred



Result from previous STPA Step 3  
UCA: <Controller> provides <Control Action>  
when <Context> [Hazards]

**Class 3 Scenario Archetype:  
Unsafe Control Execution**

- <Controller> does not provide \_\_\_\_\_



- <Process> receives \_\_\_\_\_

UCA: *<Controller>* provides *<Control Action>* when *<Context>* [Hazards]

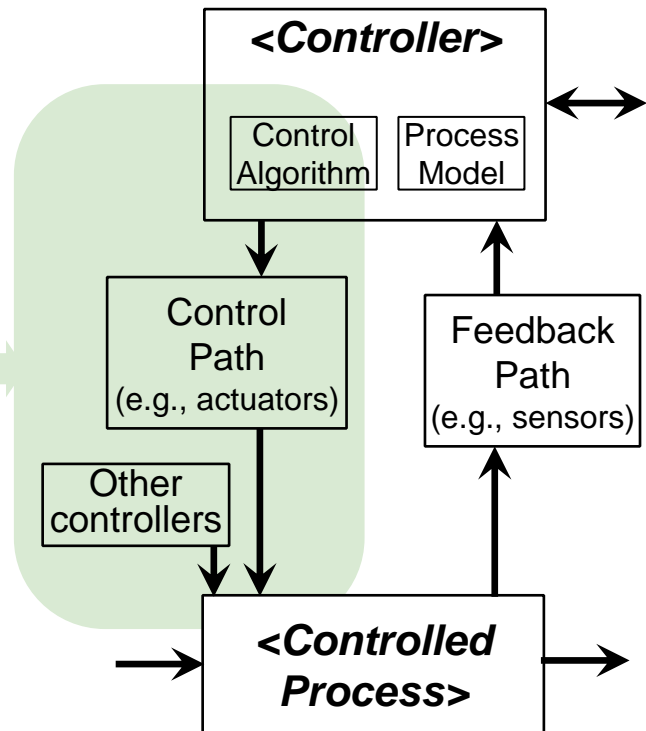
**Note the inversion!**

**Class 3 Scenario Archetype:**  
**Unsafe Control Execution**

- *<Controller>* does not provide *<Control Action>* when *<Context>*
- *<Process>* receives *<Control Action>* when *<Context>*

**Intuition behind Class 3:**

*Even if the controller does the right thing (invert the UCA), how can the control path still make it unsafe?*



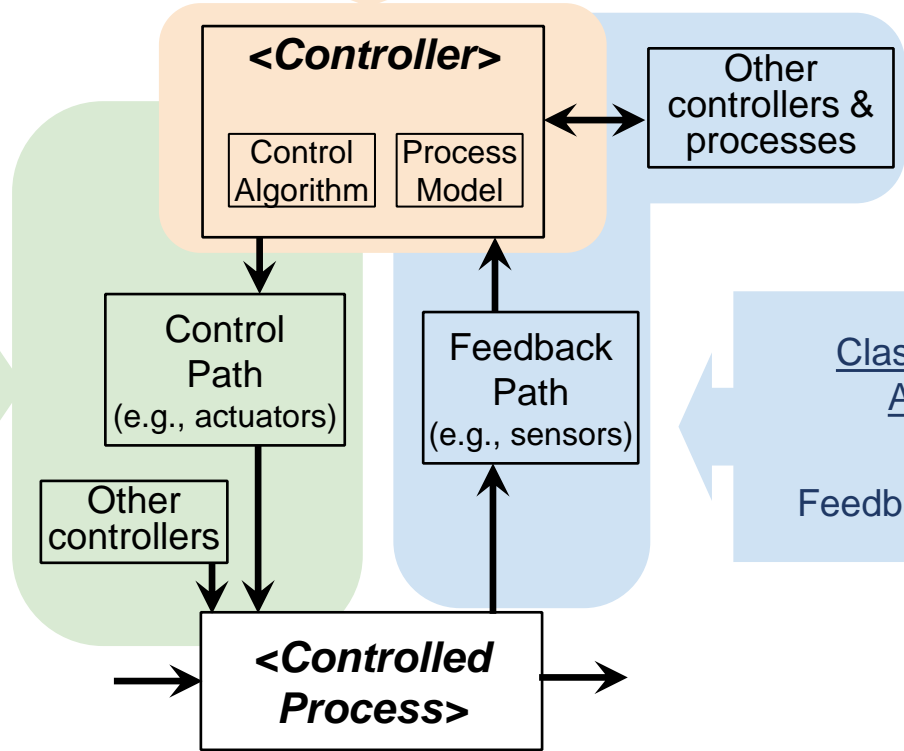
# UCA result from previous STPA Step 3

UCA: <Controller> provides <Control Action> when <Context> [Hazards]

**Class 1 Scenario Archetype:**  
Unsafe Controller Behavior

**Class 3 Scenario Archetype:**  
Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives <Control Action> when <Context>



**Class 2 Scenario Archetype:**  
Unsafe Feedback/Information

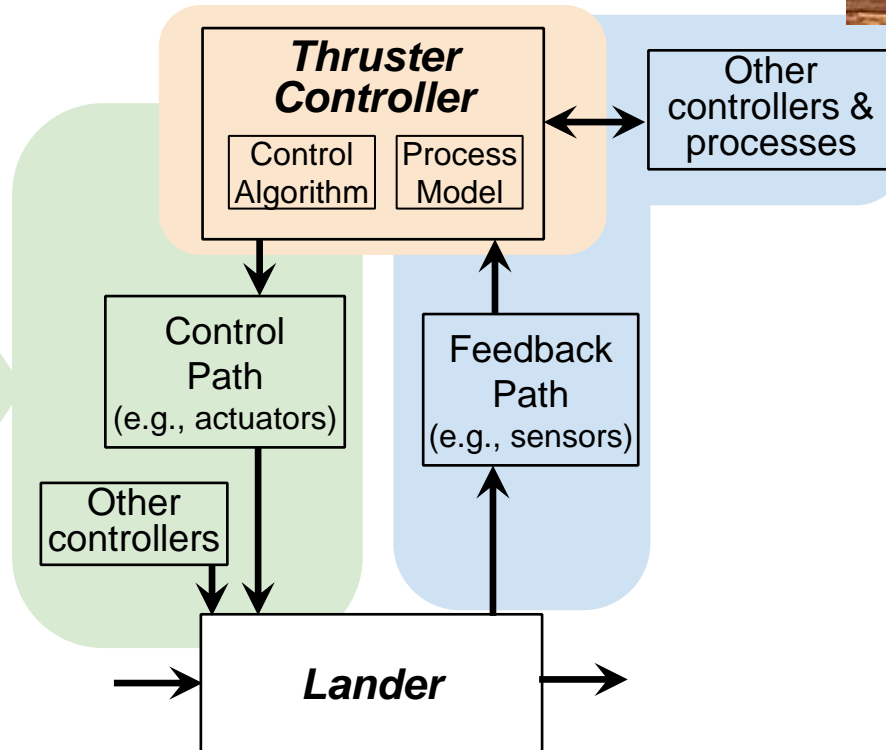
### Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*



### **Class 3 Scenario Archetype: Unsafe Control Execution**

- *<Controller>* does not provide *<Control Action>* when *<Context>*
- *<Process>* receives *<Control Action>* when *<Context>*



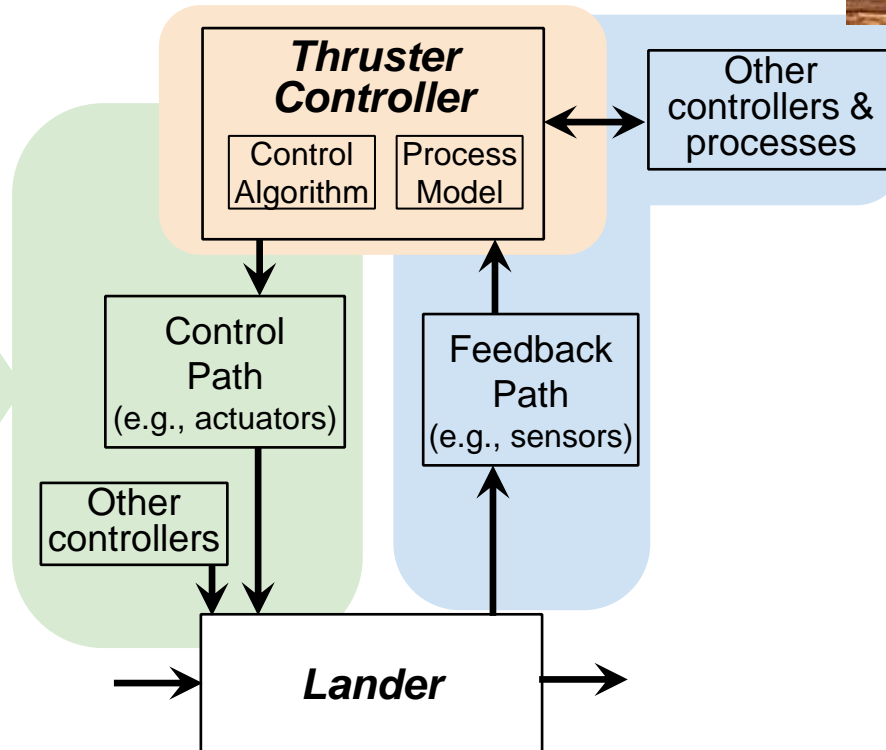
### Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*



### Class 3 Scenario Archetype: Unsafe Control Execution

- *Thruster Controller* does not provide *Disable-Thruster Cmd* when *lander is in the air*
- *<Process>* receives *<Control Action>* when *<Context>*



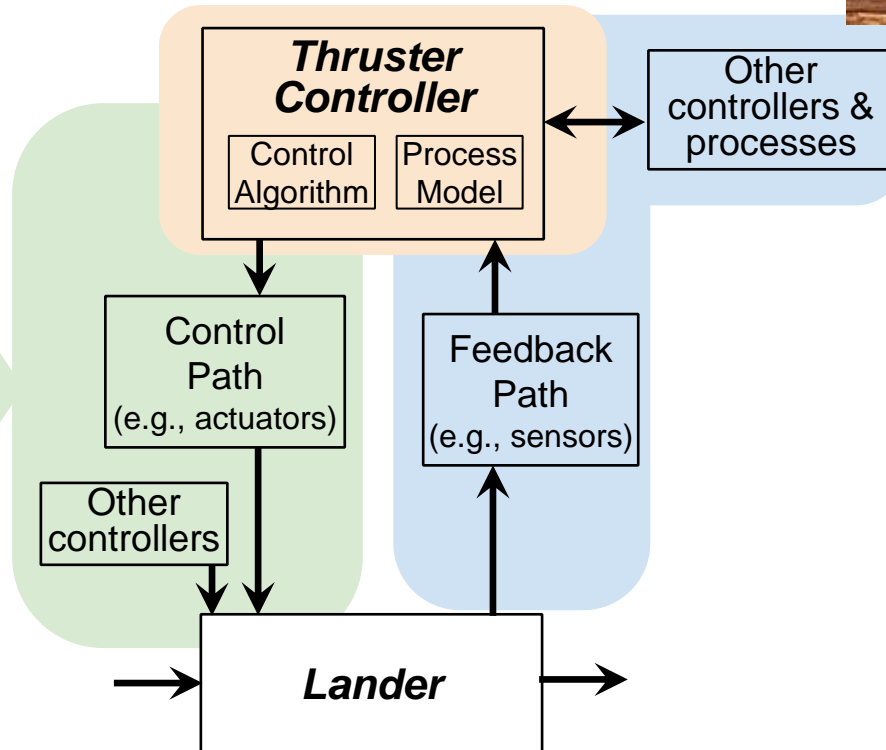
## Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*



### **Class 3 Scenario Archetype: Unsafe Control Execution**

- Thruster Controller does not provide Disable-Thruster Cmd when lander is in the air
- Lander receives Disable-Thruster Cmd when lander is in the air



# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives a <Control Action> when <Context>

## Class 4 Scenario Archetype

Ask: What can cause this Scenario Archetype?

Consider:

- 1) Failure causes
- 2) Causes without failure

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives a <Control Action> when <Context>

## Class 4 Scenario Archetype

# Refined Scenarios

Examples from real projects (not a checklist):

## Why? Examples:

- <Other Controller> may generate <Control Action> and send it to <Process>
- <Controller> sends <Control Action> with Ignore bit set, but \_\_\_\_\_.
- Previous <Control Action> is buffered, delayed, or executed out of order
- <Controller> sends <Control Action> but it isn't received on time by <Process> due to \_\_\_\_\_
- <Control Action> may be spoofed, tampered, repudiated, denied, etc.
- Etc.

**Example SME Discussion:** Wait, how would <Controlled Process> know to ignore other \_\_\_\_\_? It might not know.

# Scenario Archetypes

# Refined Scenarios

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives a <Control Action> when <Context>

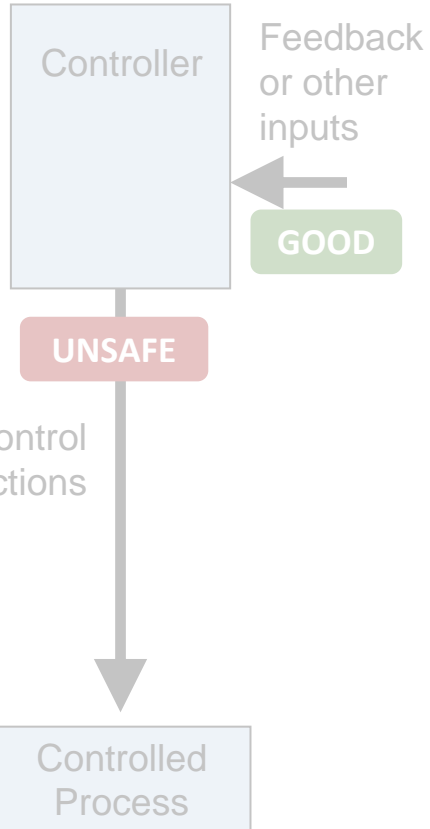
## Class 4 Scenario Archetype

## Why? Common Causes:

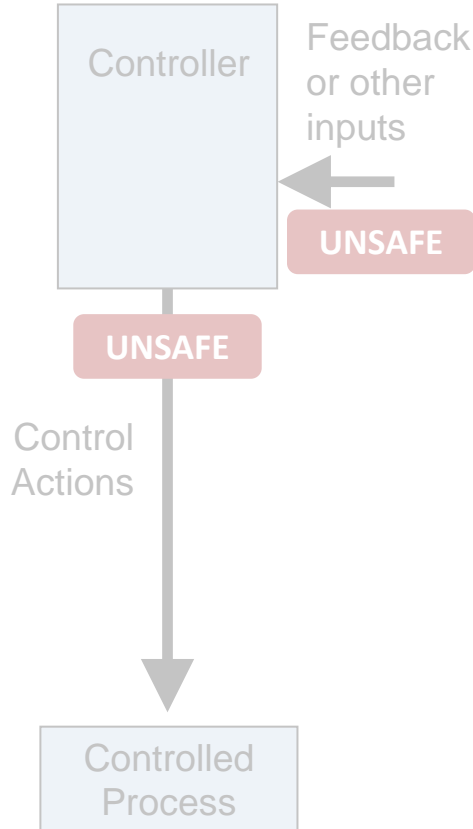
- Conflicting control action
- Previous control actions
- Delays
- Inadequate resolution / granularity
- Dropouts
- Corruption
- Content incomplete
- Control action provided in a way the controller can't use
- Overloaded control path or too much info
- Etc.

# Four Classes of Formal Scenarios

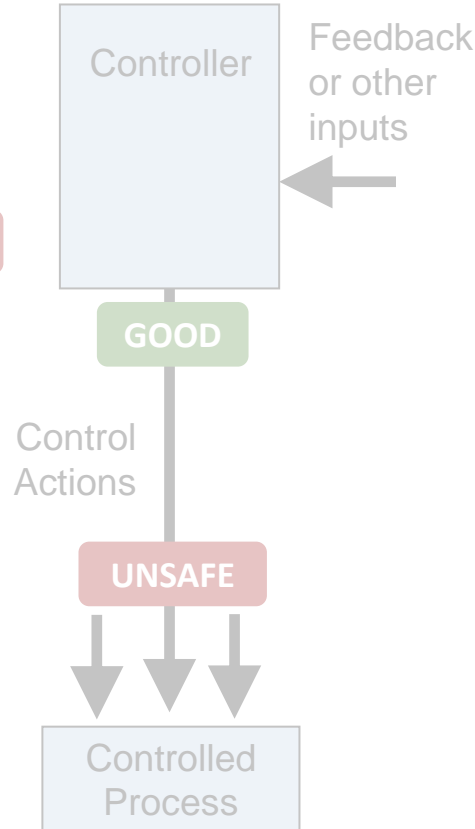
## Class 1



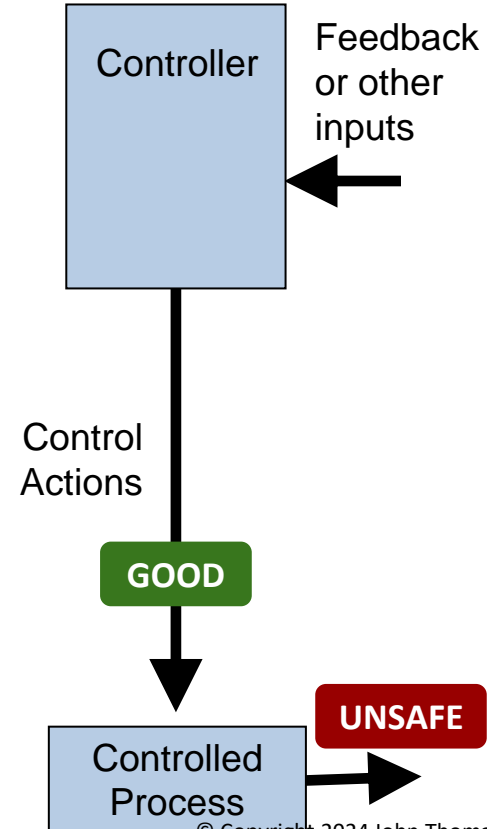
## Class 2



## Class 3



## Class 4



### Result from previous STPA Step 3

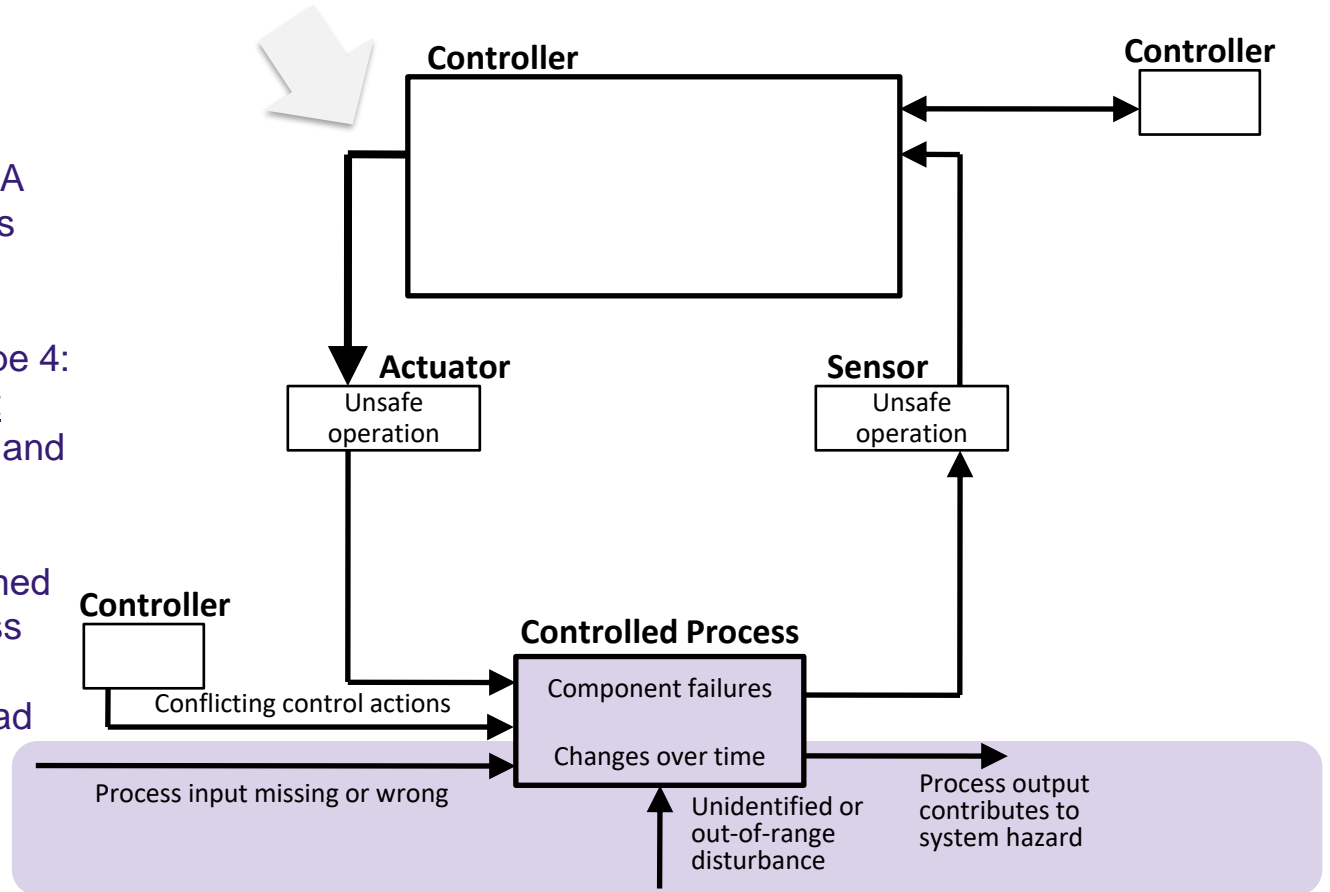
UCA: <Controller> provides <Control Action>  
when <Context> [Hazards]

### Class 4 Scenario Archetype: Unsafe Process Behavior

We need to look at how the UCA can be *emulated* due to process interactions.

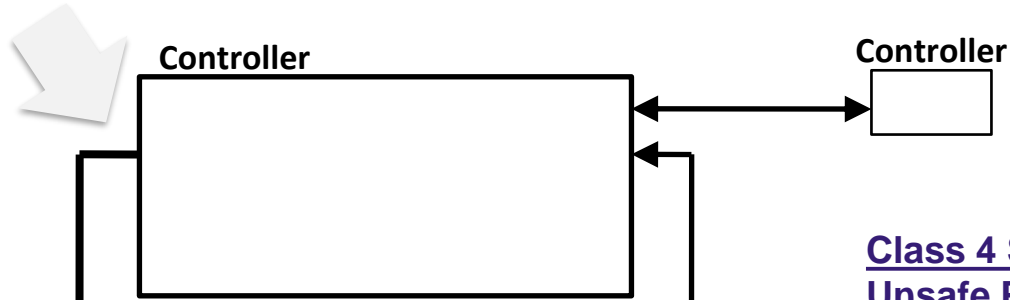
#### Constructing Scenario Archetype 4:

- Suppose the UCA did not happen (invert the UCA), and it was not received by the controlled process.
- BUT... something happened with the controlled process and related interactions making it as if the UCA had been provided.



### Result from previous STPA Step 3

UCA: <Controller> provides <Control Action>  
when <Context> [Hazards]

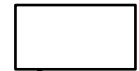


### Class 4 Scenario Archetype: Unsafe Process Behavior

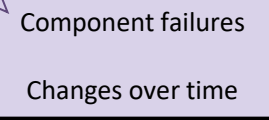


- <Process> does not receive \_\_\_\_\_
- <Process> does \_\_\_\_\_ anyway

Controller



Controlled Process



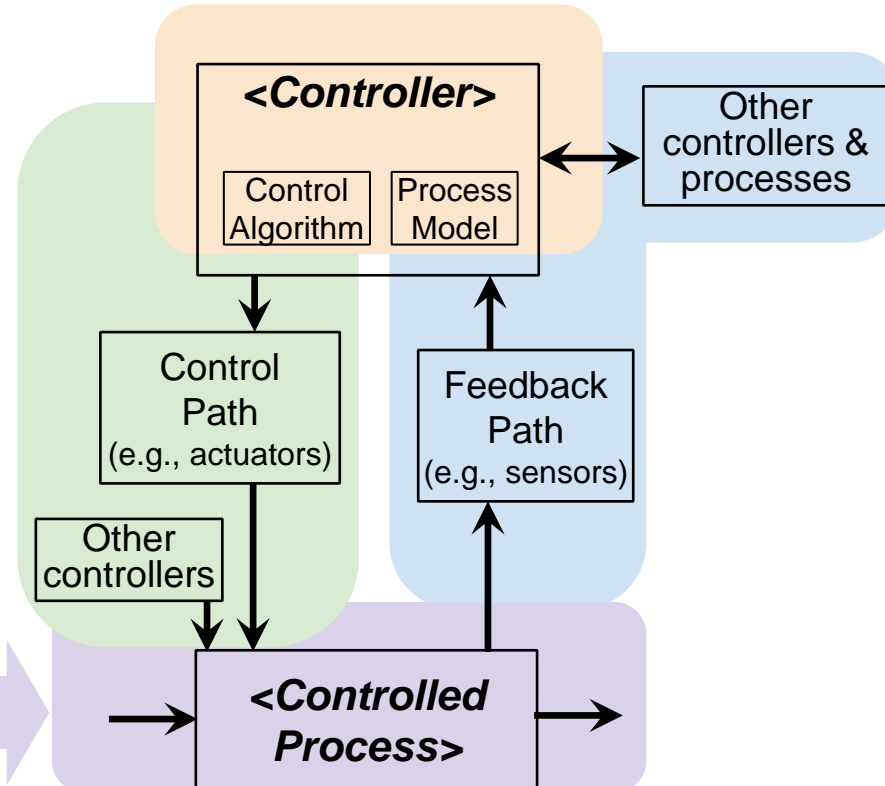
Process input missing or wrong

Unidentified or out-of-range disturbance

Process output contributes to system hazard

## Result from previous STPA Step 3

UCA: *<Controller>* provides *<Control Action>* when *<Context>* [*Hazards*]

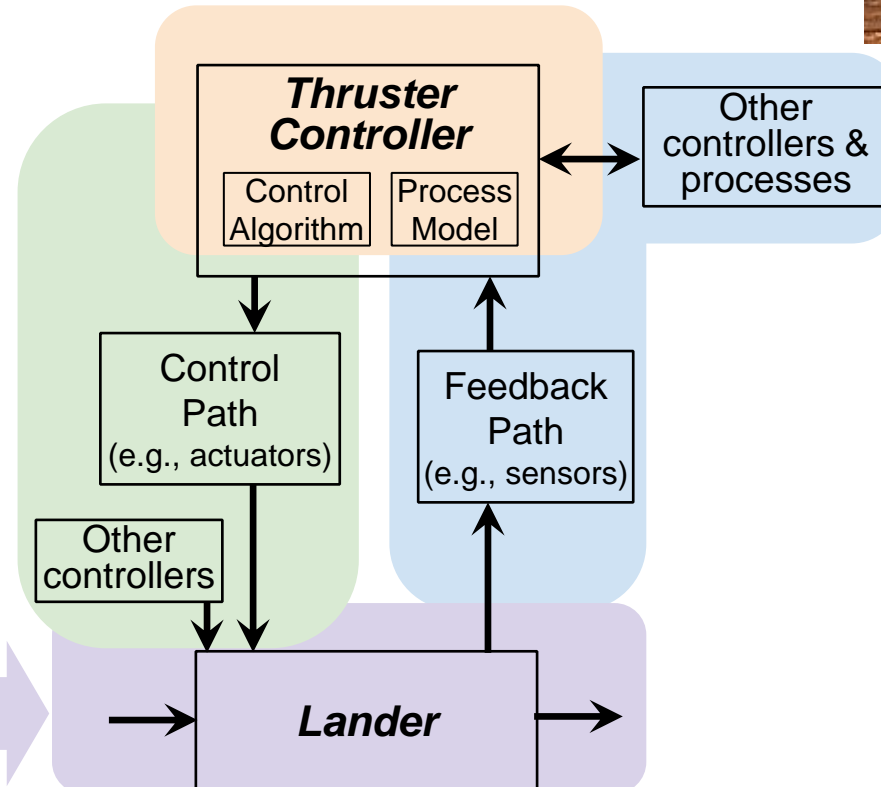


### Class 4 Scenario Archetype: Unsafe Process Behavior

- *<Process>* does not receive *<Control Action>* when *<Context>*
- *<Process>* effectively performs *<Control Action>* when *<Context>*

Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*

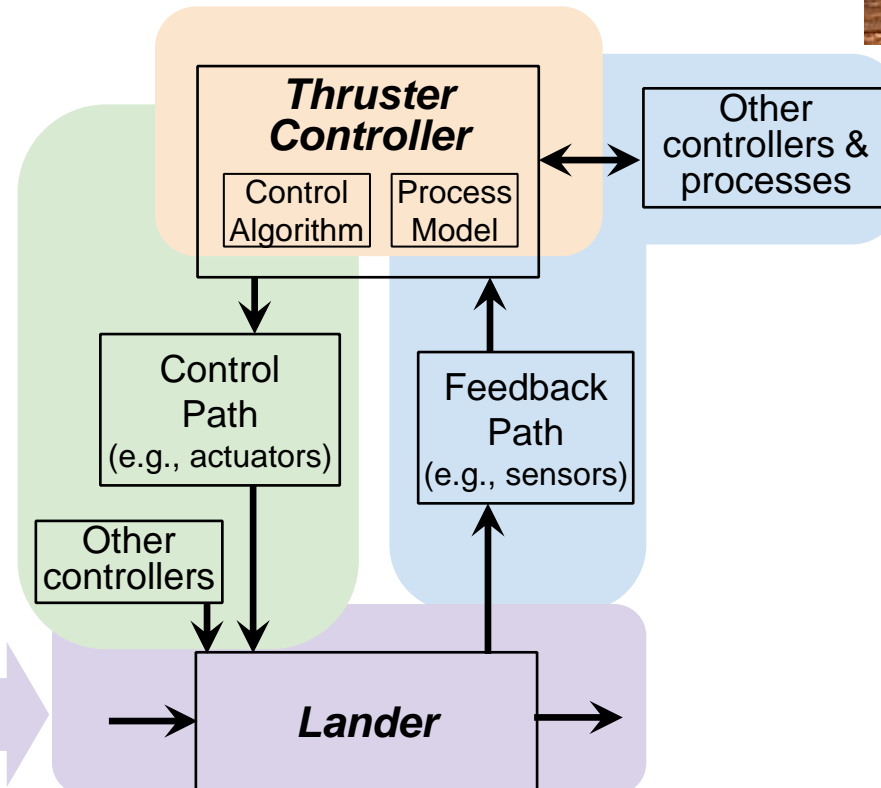


### Class 4 Scenario Archetype: Unsafe Process Behavior

- *<Process>* does not receive *<Control Action>* when *<Context>*
- *<Process>* effectively performs *<Control Action>* when *<Context>*

## Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*



### **Class 4 Scenario Archetype: Unsafe Process Behavior**

- *Lander/Thrusters* do not receive *Disable-Thruster Cmd* when *lander is in the air*
- *<Process>* effectively performs *<Control Action>* when *<Context>*

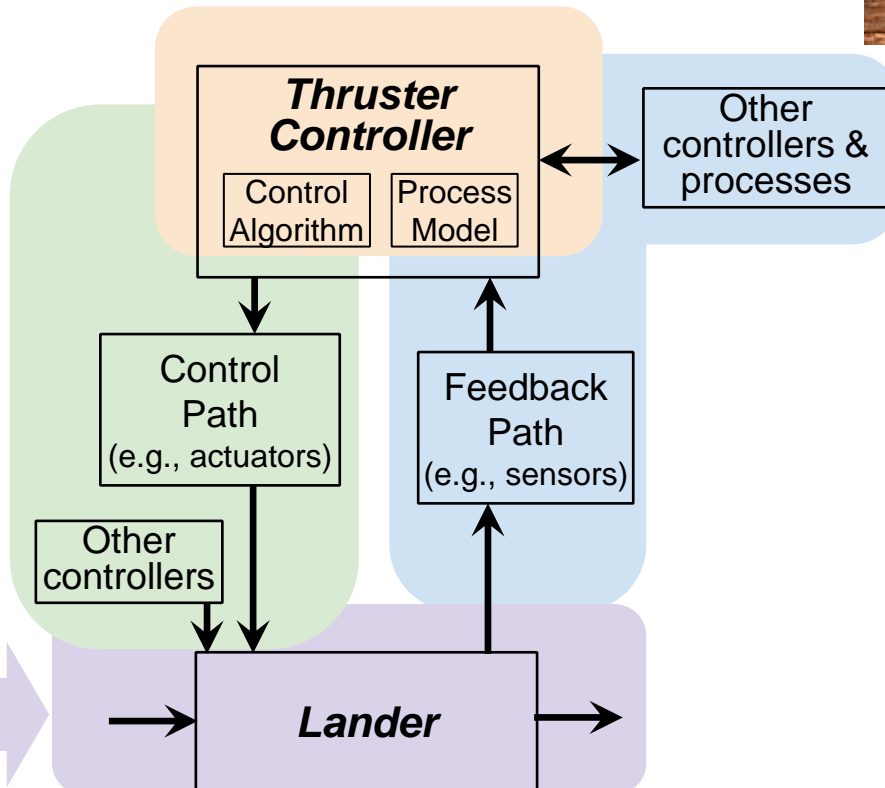
## Result from previous STPA Step 3

*UCA-2: Thruster Controller provides Disable-Thruster Cmd when lander is in the air [H-1]*



### **Class 4 Scenario Archetype: Unsafe Process Behavior**

- *Lander/Thrusters* do not receive Disable-Thruster Cmd when lander is in the air
- *Thrusters* are effectively Disabled when lander is in the air



# Scenario Archetypes

# Refined Scenarios

UCA-2:

*<Controller> provides <Control Action> when <Context>*

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- *<Controller> provides <Control Action> when <Context>*
- *<Input> to <Controller> correctly indicates <Context>*

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- *<Feedback/Input> to <Controller> does not adequately indicate <Context>*
- *<Process> is actually <Context>*

## Class 3 Scenario Archetype: Unsafe Control Execution

- *<Controller> does not provide <Control Action> when <Context>*
- *<Process> receives a <Control Action> when <Context>*

## Class 4 Scenario Archetype: Unsafe Process Behavior

- *<Process> does not receive a <Control Action> when <Context>*
- *<Process> applies <Control Action> when <Context>*

*Ask: What can cause this?*

Consider:

- 1) Failure causes
- 2) Causes without failure

# Scenario Archetypes

UCA-2:

<Controller> provides <Control Action> when <Context>

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives a <Control Action> when <Context>

## Class 4 Scenario Archetype: Unsafe Process Behavior

- <Process> does not receive a <Control Action> when <Context>
- <Process> applies <Control Action> when <Context>

# Refined Scenarios

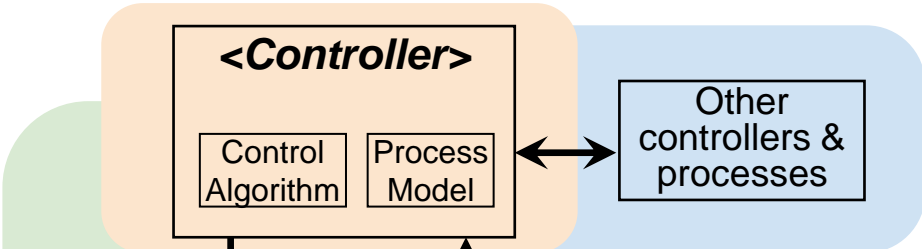
Examples from real projects (not a checklist):

## Why?

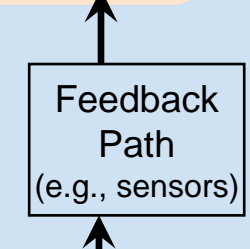
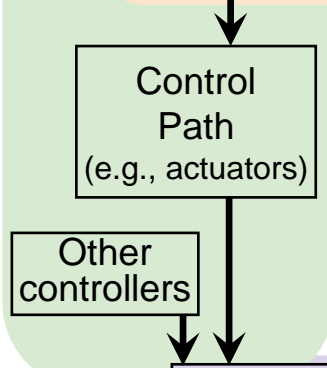
- <Process> may mechanically \_\_\_\_
- <Process> may run out of \_\_\_\_
- <Process> responds too early before \_\_\_\_ (or too late after \_\_\_\_)
- <Process> may see that \_\_\_\_ is full, in which case <Process> will automatically \_\_\_\_, which results in \_\_\_\_.
- If <Process> is in \_\_\_\_ mode, then all \_\_\_\_ will be ignored.

# Evaluating Scenario Coverage

Class 1 Scenario Archetype:  
Unsafe Controller Behavior

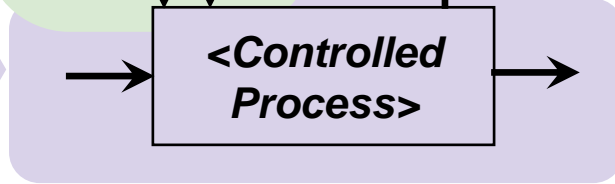


Class 3 Scenario Archetype:  
Unsafe Control Execution



Class 2 Scenario Archetype:  
Unsafe Feedback/Information

Class 4 Scenario Archetype:  
Unsafe Process Behavior



# Scenario Coverage

## Class 1 Scenario Archetype: Unsafe Controller Behavior

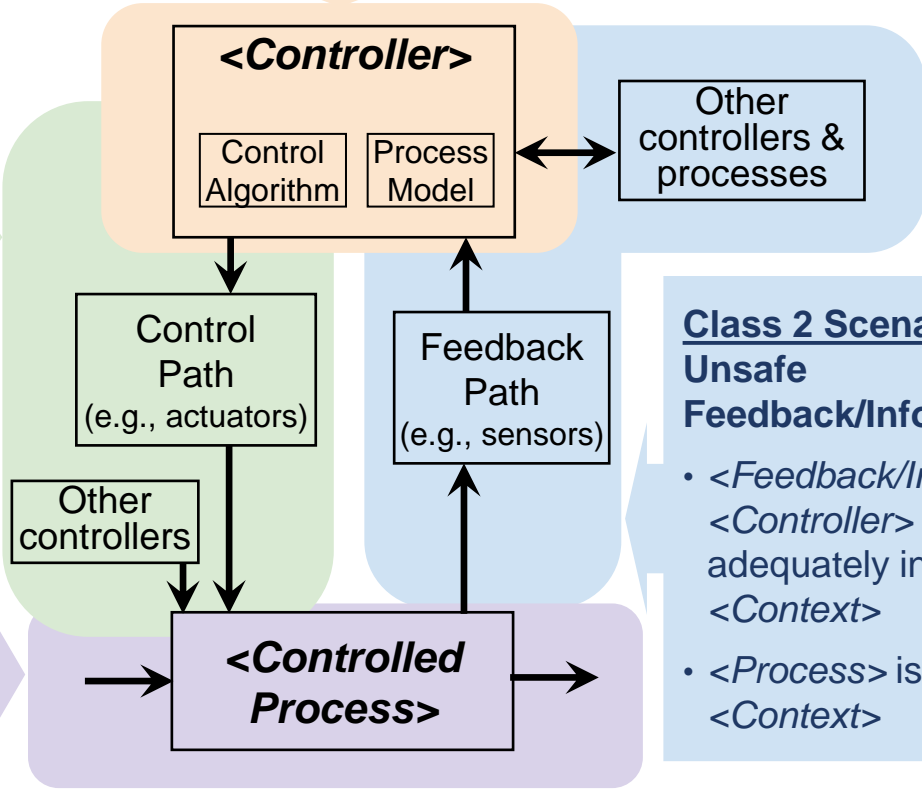
- UCA: <Controller> provides <Control Action> when <Context>
- <Feedback/Input> to <Controller> correctly indicates <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives <Control Action> when <Context>

## Class 4 Scenario Archetype: Unsafe Process Behavior

- <Process> does not receive <Control Action> when <Context>
- <Process> effectively applies <Control Action> when <Context>



## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

UCA-2:

<Controller> provides <Control Action> when <Context>

# STPA Formal Scenario Classes

## Class 1 Scenario Archetype: Unsafe Controller Behavior

- <Controller> provides <Control Action> when <Context>
- <Input> to <Controller> correctly indicates <Context>

## Class 2 Scenario Archetype: Unsafe Feedback/Information

- <Feedback/Input> to <Controller> does not adequately indicate <Context>
- <Process> is actually <Context>

## Class 3 Scenario Archetype: Unsafe Control Execution

- <Controller> does not provide <Control Action> when <Context>
- <Process> receives a <Control Action> when <Context>

## Class 4 Scenario Archetype: Unsafe Process Behavior

- <Process> does not receive a <Control Action> when <Context>
- <Process> applies <Control Action> when <Context>

## Refined scenarios

- 1) Failure causes
- 2) Causes without failure

## Discussion

These formal scenario classes can be used to:

- Establish contractual / regulatory compliance
- Improve consistency, inform reviews
- Generate the classes from UCAs
- Compare coverage/scope between different safety & reliability methods

# STPA Scenario Archetype Table

	Not providing	Providing	Timing	Duration
Control Action	UCA-1	UCA-2	UCA-3	UCA-4
	↓	↓	↓	↓
<b>Scenario Class 1:</b> <b>Unsafe Controller Behavior</b>	1)<controller> doesn't provide <cmd> when <context> 2)<controller> received feedback (or other inputs) that indicates <context>	1)<controller> provides <cmd> when <context> 2)<controller> received feedback (or other inputs) that indicates <context>	1)<controller> provides <cmd> too late/early after/before <context> 2)<controller> received feedback (or other inputs) that indicates <context> on time / in order	1)<controller> stops/continues providing <cmd> too soon/long before/after <context> 2)<controller> received feedback (or other inputs) that indicates <context> on time
<b>Scenario Class 2:</b> <b>Unsafe Feedback Path</b>	1)feedback (or other inputs) received by <controller> does not adequately indicate <context> 2)<context> is true	1)feedback (or other inputs) received by <controller> does not adequately indicate <context> 2)<context> is true	1)feedback (or other inputs) received by <controller> does not indicate <context> (too late/early/out of order) 2)<context> is true	1)feedback (or other inputs) received by <controller> does not indicate <context> (inappropriate duration) 2)<context> is true
<b>Scenario Class 3:</b> <b>Unsafe Control Path</b>	1)<controller> does provide <cmd> when <context> 2)<cmd> is not received by <controlled process> when <context>	1)<controller> does not provide <cmd> when <context> 2)<controlled process> receives <cmd> when <context>	1)<controller> does not provide <cmd> <context> (not too late/early/out of order) 2)<cmd> is received by <controlled process> <context> (too late/early/out of order)	1)<controller> provides <cmd> with appropriate duration 2)<cmd> is received by <controlled process> with <context> (inappropriate duration)
<b>Scenario Class 4:</b> <b>Unsafe Controlled Process Behavior</b>	1)<cmd> is received by <controlled process> when <context> 2)<controlled process> does not respond by <...>	1)<cmd> is not received by <controlled process> when <context> 2)<controlled process> responds by <...>	1)<cmd> is not received by <controlled process> <context> (not too late/early/out of order) 2)<controlled process> responds by <...> <context> (too late/early/out of order)	1)<cmd> is received by <controlled process> with appropriate duration 2)<controlled process> does not respond by <...> with <context> (inappropriate duration)

# Classes 1-4 are already in the STPA Handbook

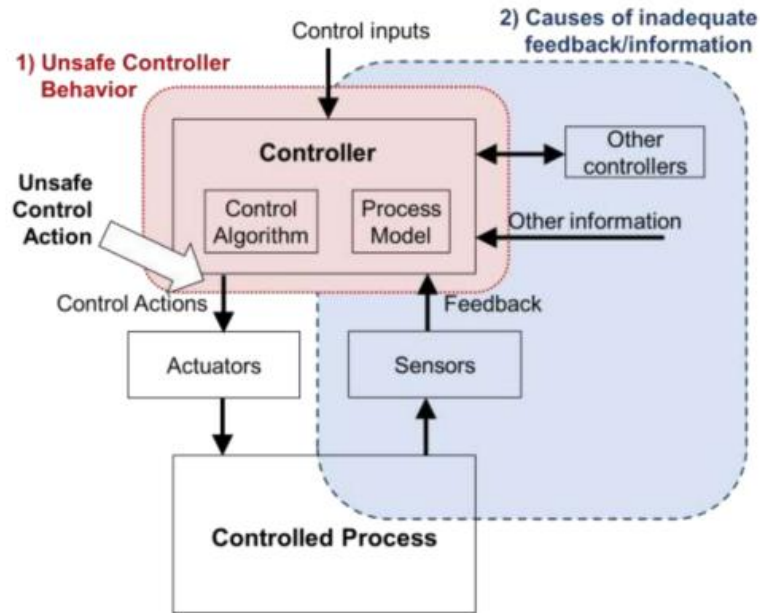


Figure 2.18: Unsafe Control Actions can be caused by (1) unsafe controller behavior and (2) inadequate feedback and other inputs

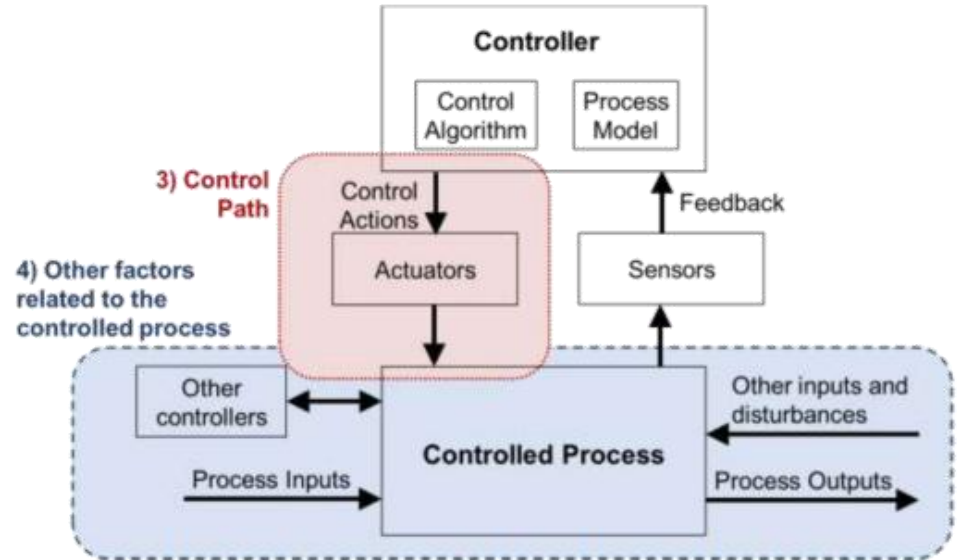


Figure 2.19: Generic control loop illustrating 1) the control path and 2) other factors that can affect the controlled process

**Discussion:** The new scenario process is consistent with the STPA Handbook but provides a more formal process with more guidance.

# Testing the New Approach: Results

# Many Real-World Evaluations Completed

## Examples:

- (Nuclear) A New Process for Building STPA Causal Scenarios, John Thomas, 2016 MIT STAMP Workshop
- (Space) A Process for STPA: STAMP Accident Model of HITOMI and Expansion to Future Safety Culture, John Thomas and Nancy Leveson (MIT), Masa Katahira and Naoki Ishimama (**JAXA**), Nobuyuki Hoshino (JAMSS), 2017 MIT STAMP Workshop
- (Aircraft) Systems Theoretic Process Analysis Applied to **Air Force** Acquisition Technical Requirements Development, Sarah Summers, MIT Thesis, 2017
- (Aircraft) STPA Applied to Manned-Unmanned Teaming, Jeremiah Robertson, MIT Thesis, 2019
- (Auto) STPA Applied to Autonomous Vehicles, Jeff Stafford (**Renesas**), John Thomas (MIT), 2019 MIT STAMP Workshop
- (Software) STPA Applied to AV Software, Shaun Mooney (**Codethink**), John Thomas (MIT), 2019
- (Auto) Application of Hierarchy to STPA: A Human Factors Study on Vehicle Automation, Rachel Cabosky, 2020, MIT Thesis (collaboration with **GM**)
- (Military Aviation) Evaluation of STPA for Aircraft Safety Assessment, **US Army**, 2021
- (Aviation) Rotary-Wing Aircraft Development: Cybersecurity and Safety STPA Status Report, MIT & MIT-LL, 2021
- (Aviation) A Top-Down, Safety-Driven Approach to Architecture Development for Complex Systems, Justin Poh, MIT Thesis, 2022
- (Aviation) System-Theoretic Safety Analysis for Teams of Collaborative Controllers, Andrew N. Kopeikin, MIT Dissertation, 2024
- (Communications) System-Theoretic Process Analysis of a Novel Airborne Laser Communication System, Brittany Bishop, MIT Thesis, 2024
- (Software) OEM & Supplier Use of STPA for Advanced Driver-Assistance Systems, **Qualcomm**, 2024
- (Software) Collaboration with Tim Falzone, Ruben Barroso, Garrett Holthaus, **Google**, 2024

In every application, the new scenario process has identified critical loss scenarios and causes that were previously overlooked

# New Scenario Approach: Observations

## Disadvantages

- More rules and structure
- Takes longer to teach & learn
- Unclear if it takes longer to perform (less time in some cases)

## Advantages

- The rules and structure provide more guidance
- Enables a more directed search, less ad-hoc and informal
- The rationale for the scenarios and how you found them is clearer.
- So far, the new process has always captured additional cases that were previously overlooked
- The top-down approach was more scalable to extremely complex systems compared to previous STPA attempts
- Less repetition in the results (shorter documentation, higher information density)
- Provides clear exit criteria to rigorously review and find gaps
- Enables automation of some parts of Step 4
- The formal structure can enable mathematical proofs for properties like scenario coverage
- Improved consistency and repeatability. Less dependence on who is doing the analysis.

# How to run an STPA project

Who would do this and how they would  
coordinate with others?

# STPA Project Participants

