NTNU

SafeScrum

SINTEF

Stepwise refinement of Requirements and Safety in SafeScrum

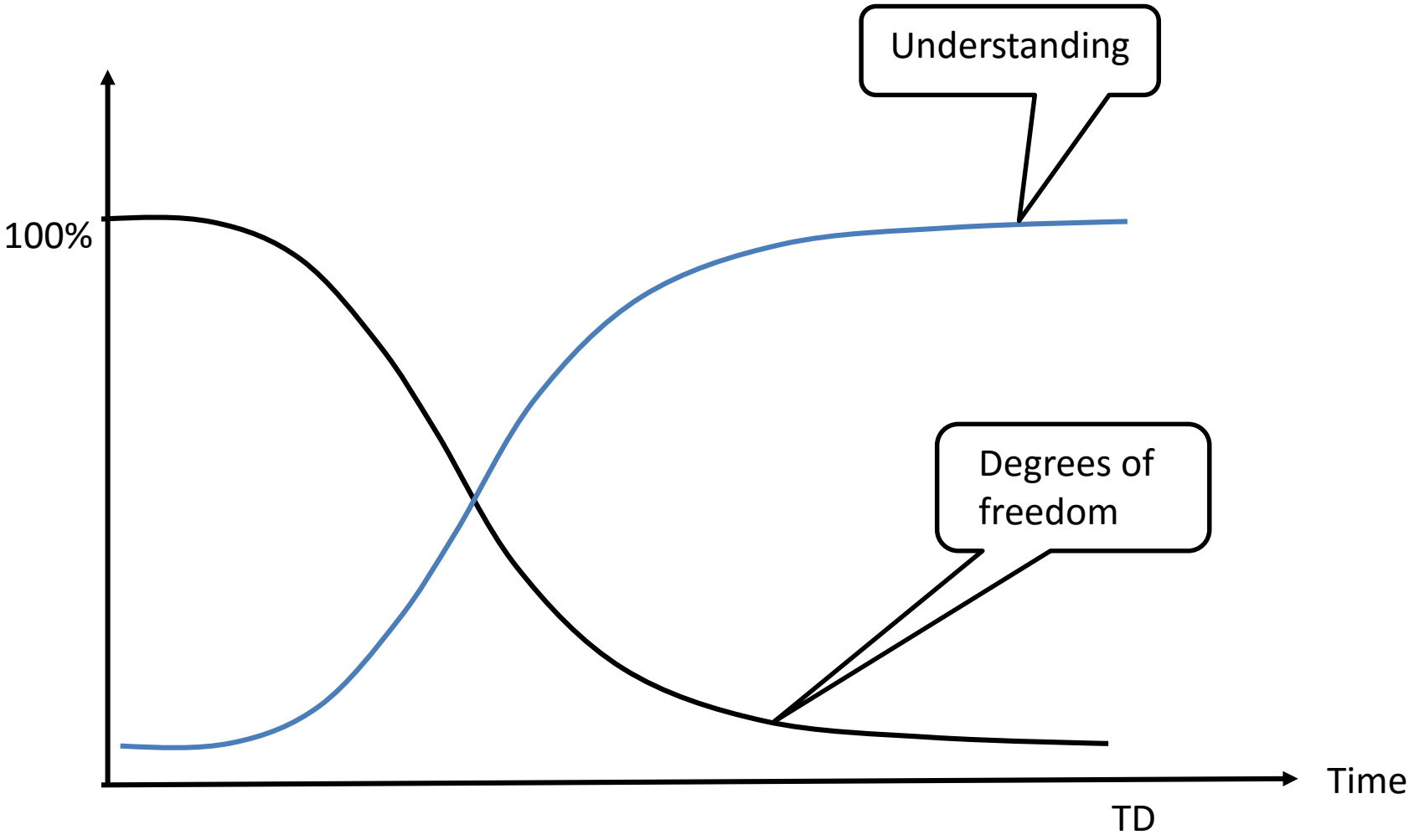
Tor Stålhane, IDI / NTNU

Thor Myklebust, ICT / SINTEF

Geir K. Hanssen, ICT / SINTEF

Børge Haugset, ICT / SINTEF

Challenge for all development



Challenges for safety-critical software

Architecture

- Important decisions have to be made early in the project when we have little information

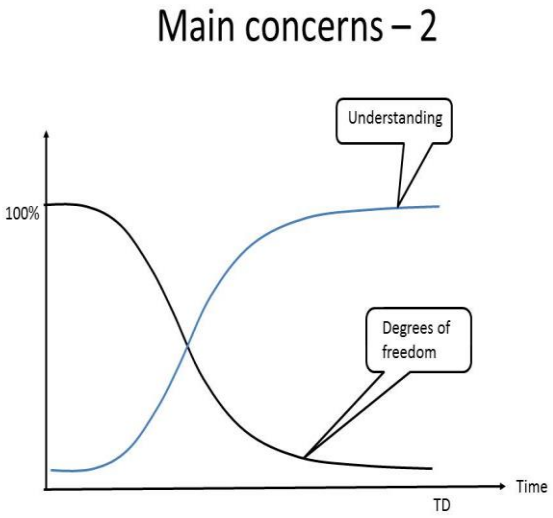
Safety analysis

- Must start as early as possible in a project
- Will generate new requirements due to the need to
  - make required functionality more safe
  - add barriers to handle unsafe situations

An early start – 1

Safety must be

- Considered from day 1 => safety considerations must be part of all decisions
- Based on
  1. epics and architectural patterns
  2. user stories and high level design
  3. generic system components



Important challenge:

Many important decisions are made early, when we have little knowledge of the final system

An early start – 2

The following well-known concepts should be used

- Architectural patterns – several exist for most application areas
- Generic
  - Hazard lists – environment and domain specific – e.g., FAA for aerospace
  - Failure modes – from just a few (e.g. 2) to quite many (e.g. 10)
  - Fault trees – environment and domain specific – e.g. IMO, building standards

Early safety analysis – 1

1. Write theme and epic – get an overview of what we want to achieve
2. Select an architectural pattern
   a. Allows us to identify generic components
   b. Starting-point for next level safety analysis and barriers
3. Apply FMEA to generic components to identify barriers
4. Write detailed system requirements

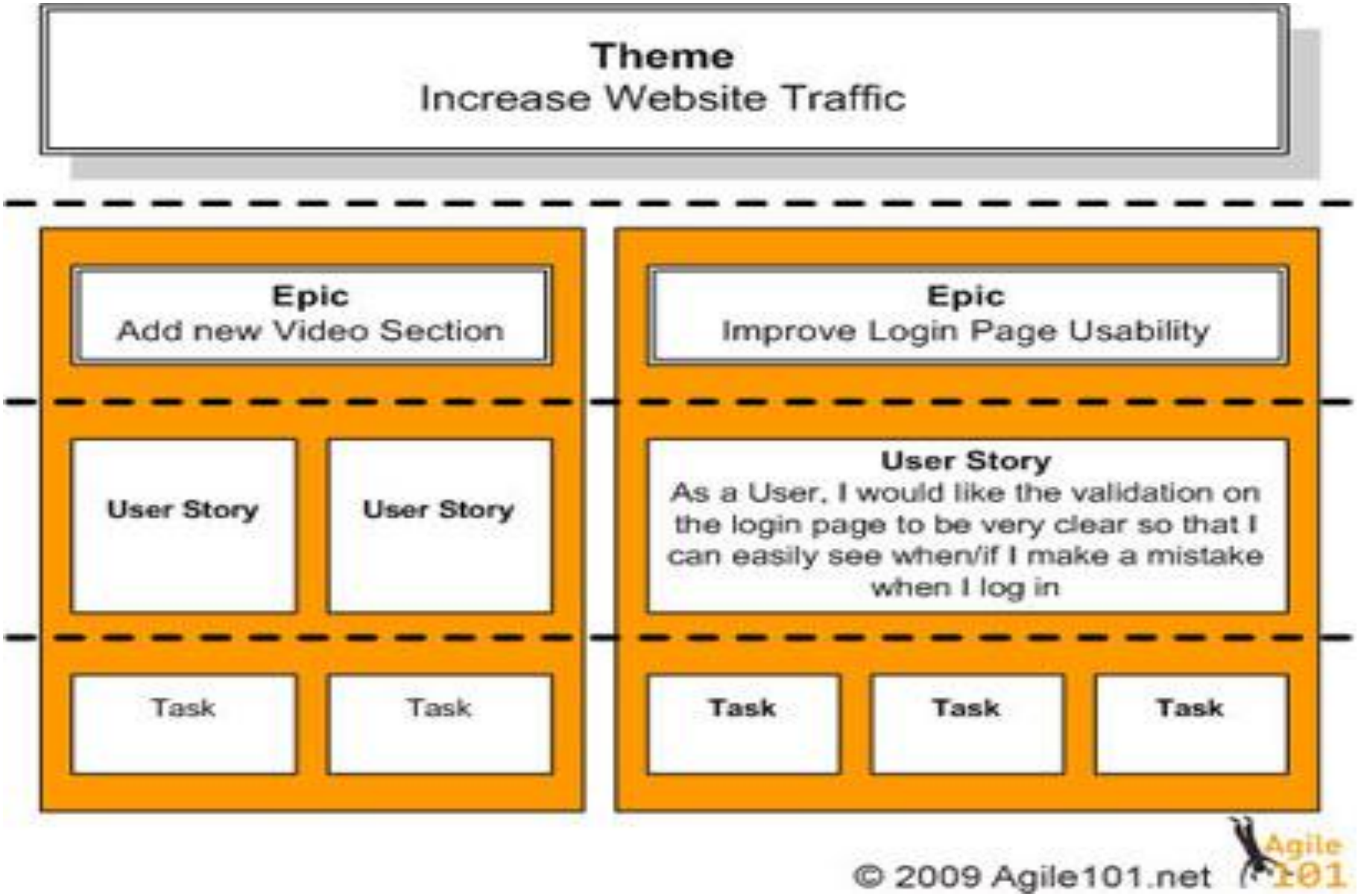
Early safety analysis – 2

We must be able to involve all types of stakeholders. Safety analysis is not a job only for the safety analysts.

The methods we use have to be easy to

- Learn – no extensive coursing needed
- Use – all categories of stakeholders must be able to contribute

Themes and epics



Preliminary Hazard Analysis – PHA

Epics and patterns

Hazard	Cause	Main effect	Preventive action

FMEA

Generic components

Unit description	**Failure description**		**Failure effect on the next level**	**Recommendation**
	Failure mode	Failure cause		

User stories

Function	**Function description**			
Functional failure mode	Effects	Cause	Detection Current method	Comments

Generic functional failure modes used as guide-words:
Over, Under, No, Intermittent, Unintended

Input Focused FMEA - Stories

Generic components

Story ID:		**List of component input sources:**		**Suggested barriers and new requirements**
Output failure mode	**Output failure mode description**	**Input deviation**	**Component failure**	
Omission				
Commission				
Wrong action				
Too late				

Generic failure modes

Be ware: Generic failure modes

- Is not a replacement for using your head
- Are most useful in the early stages where we still have a lot of choices when it come to
  - architecture
  - barrier solutions
- Could be used as guide words in the analysis

Generic failure modes – examples

Component type	Failure mode
Software systems - control system, e.g., a PLC	Omission – something is not done, no action
	Commission – something more is done
	Wrong action
	Delayed – right action but too late
Hardware component, e.g. a pump or a sensor	No action
	Wrong action
	Delayed action

We can use generic failure modes to

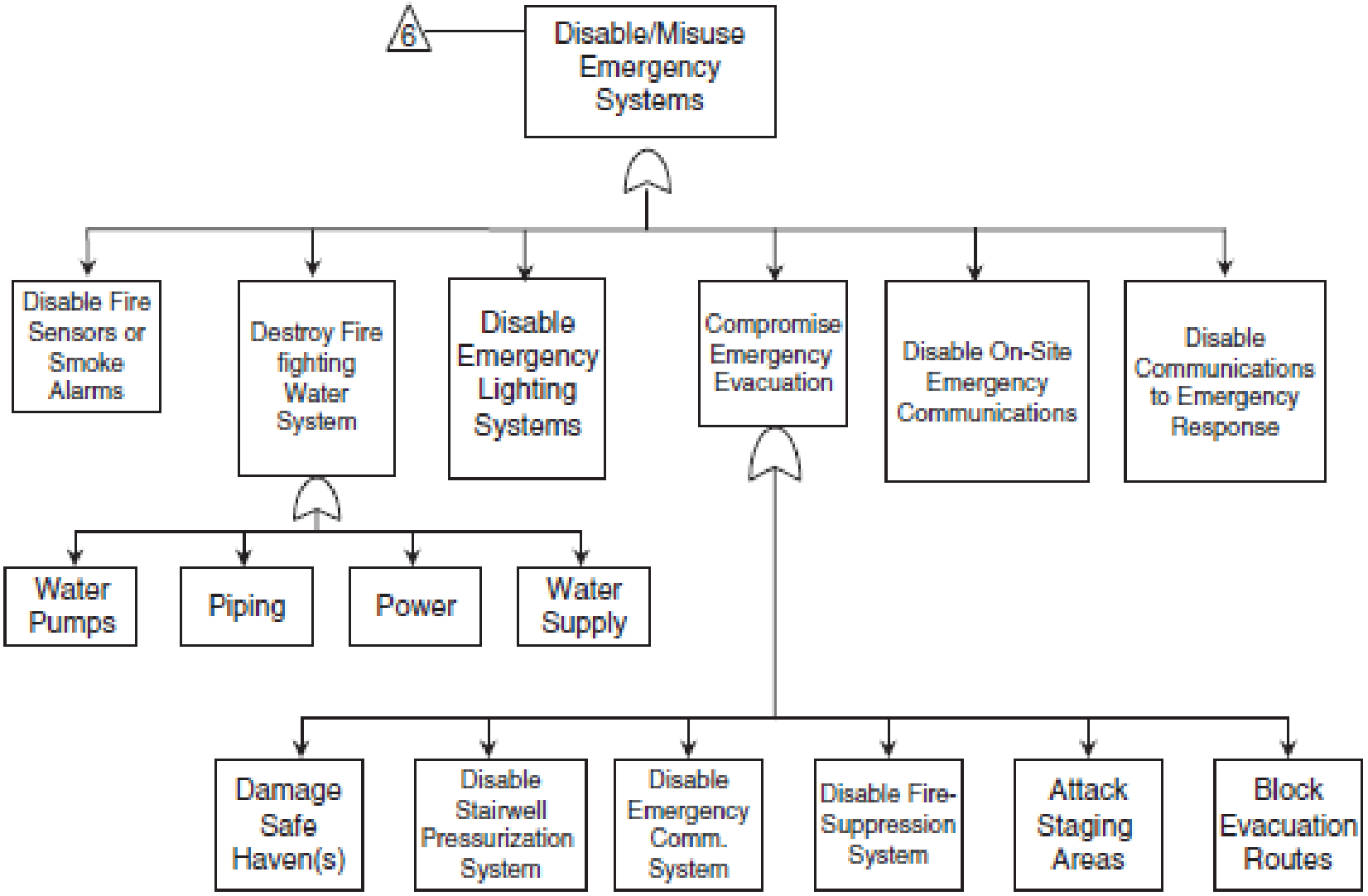
- simplify the FMEA process
- give the FMEA an easy start
- promote reuse of FMEAs wherever practical

Generic fault trees – 1

Generic fault trees give information needed to

- Get a broad overview on
  - the consequences of component failures
  - possible barriers
- Create checklists - what
  - have we included in our system
  - is left to be handled by others

Generic fault trees – 2

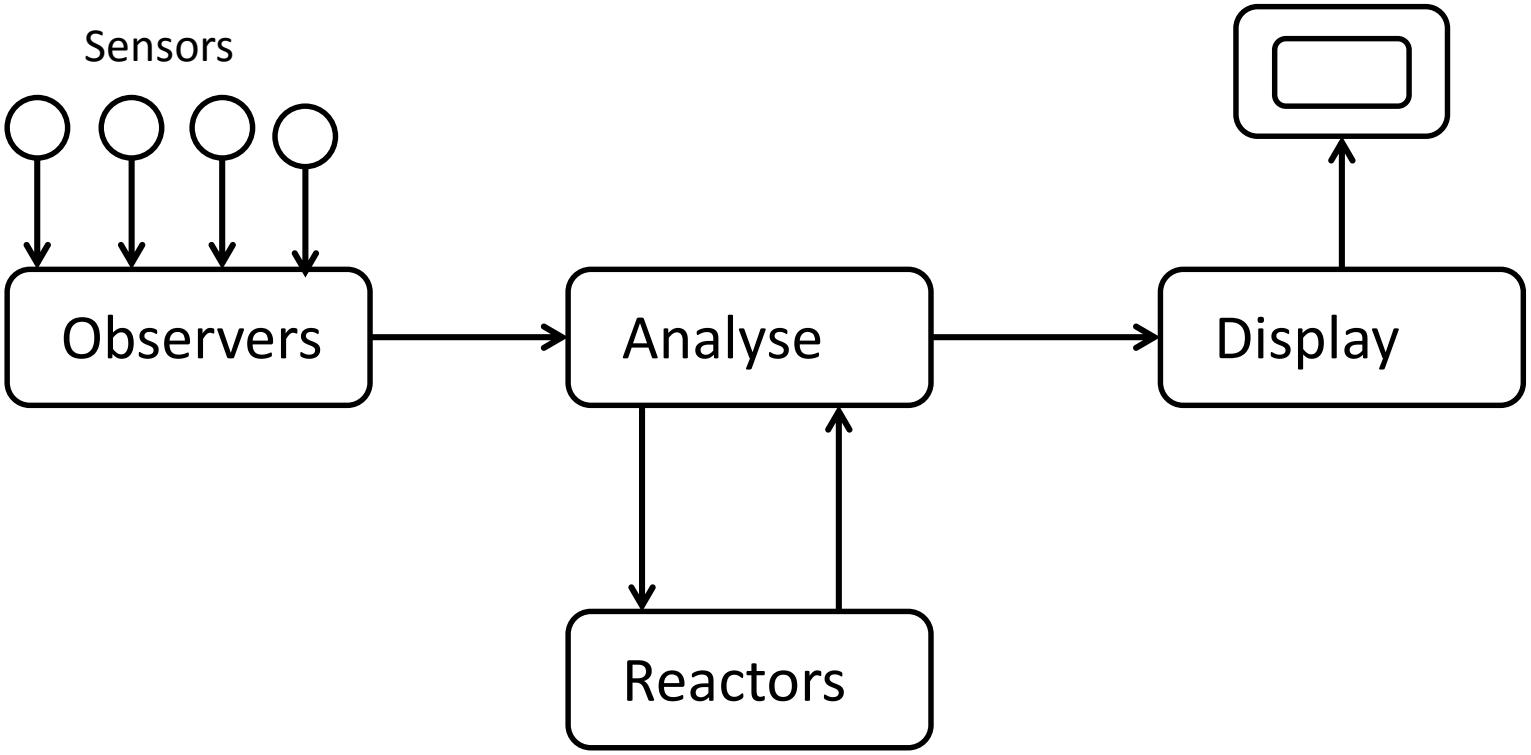


Architectural patters

There are several sources for real time software patterns described as e.g.

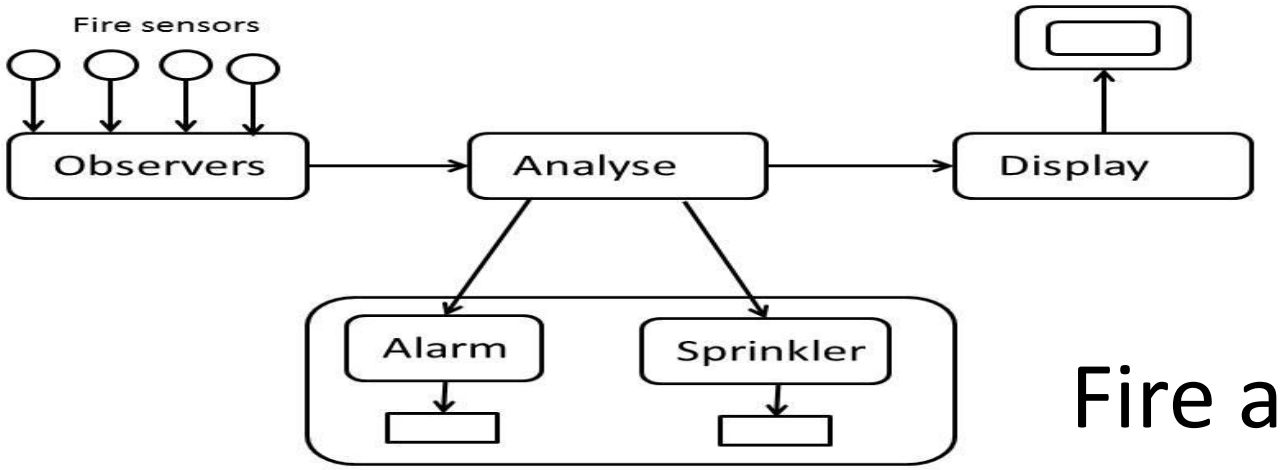
- Message sequence diagrams
- UML classes
- Architectural patterns. Example follows
- State diagrams

Observe-and-react pattern

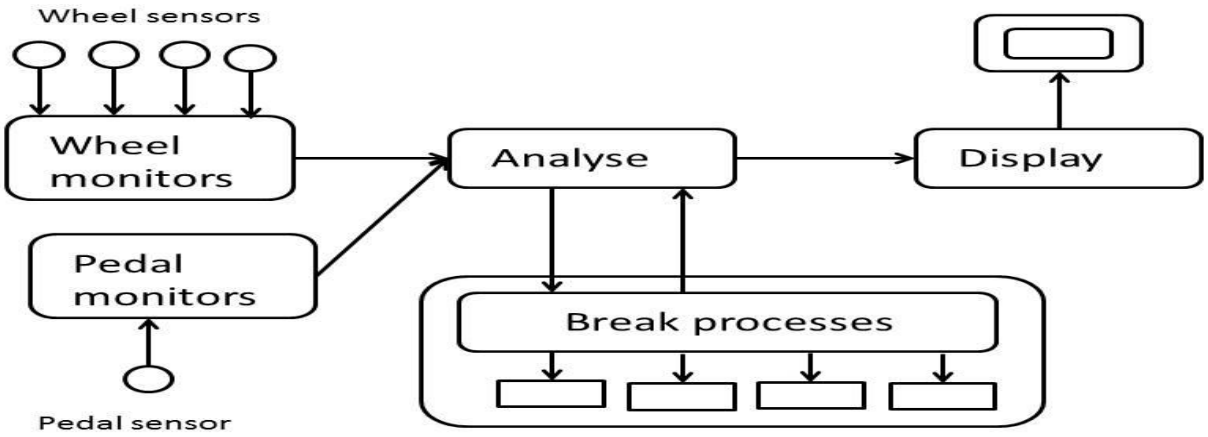


Possible weakness - environment not included =>
No feedback mechanism

Observe-and-react – examples (1)

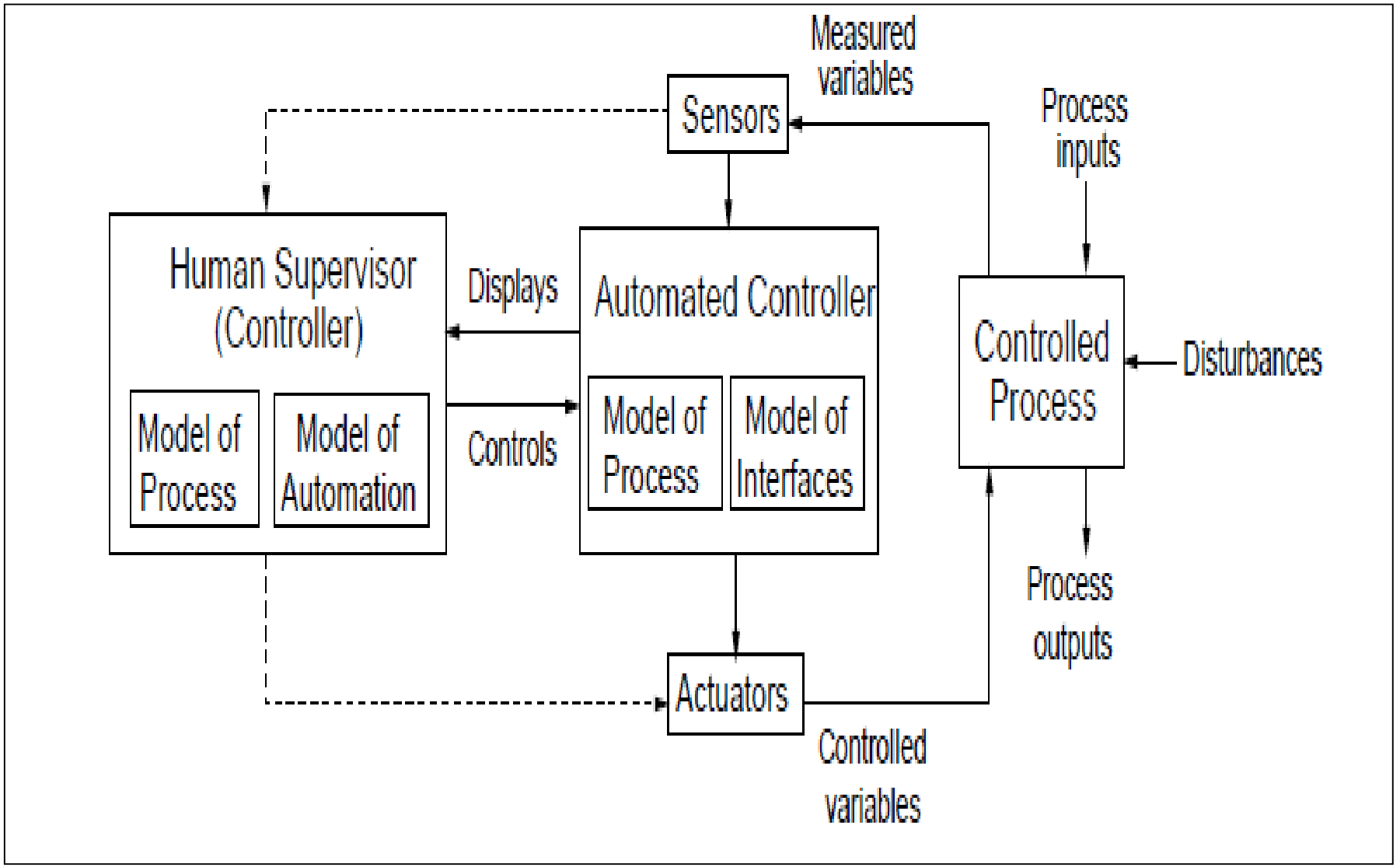


Fire alarm



ABS

Observe – React with Leveson's addition – 1

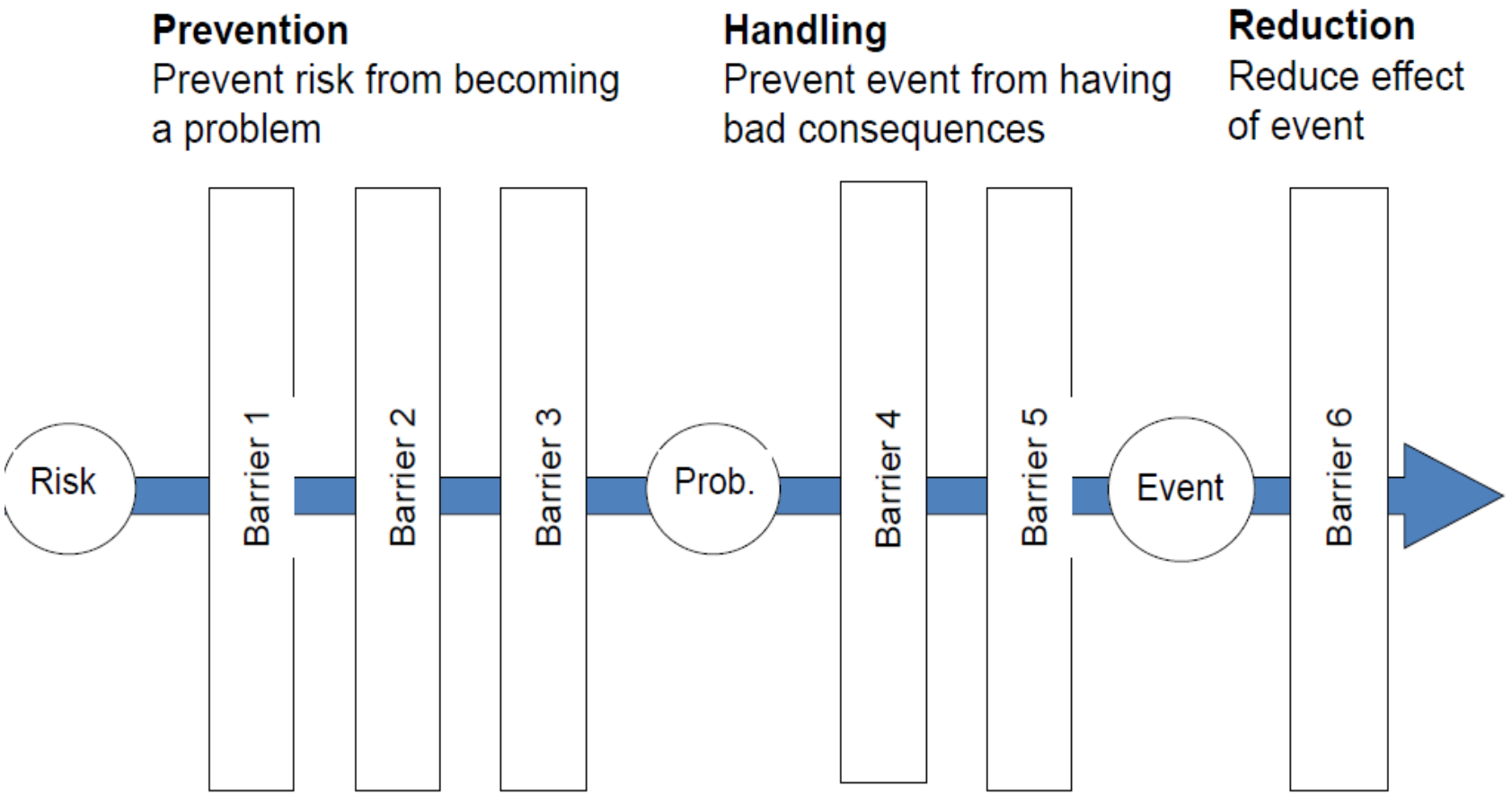


Observe – React with Leveson's addition – 2

Allows us to consider the effect of

- Process problems
  - Missing or wrong input
  - Output that can cause harm
  - Input that can create unforeseen – e.g. out of range – process disturbances
- Model problems – process, automation and interfaces
  - Inconsistent
  - Incomplete
  - incorrect

Barriers


Prevention
Prevent risk from becoming a problem
Handling
Prevent event from having bad consequences
Reduction
Reduce effect of event
Risk
Barrier 1
Barrier 2
Barrier 3
Prob.
Barrier 4
Barrier 5
Event
Barrier 6

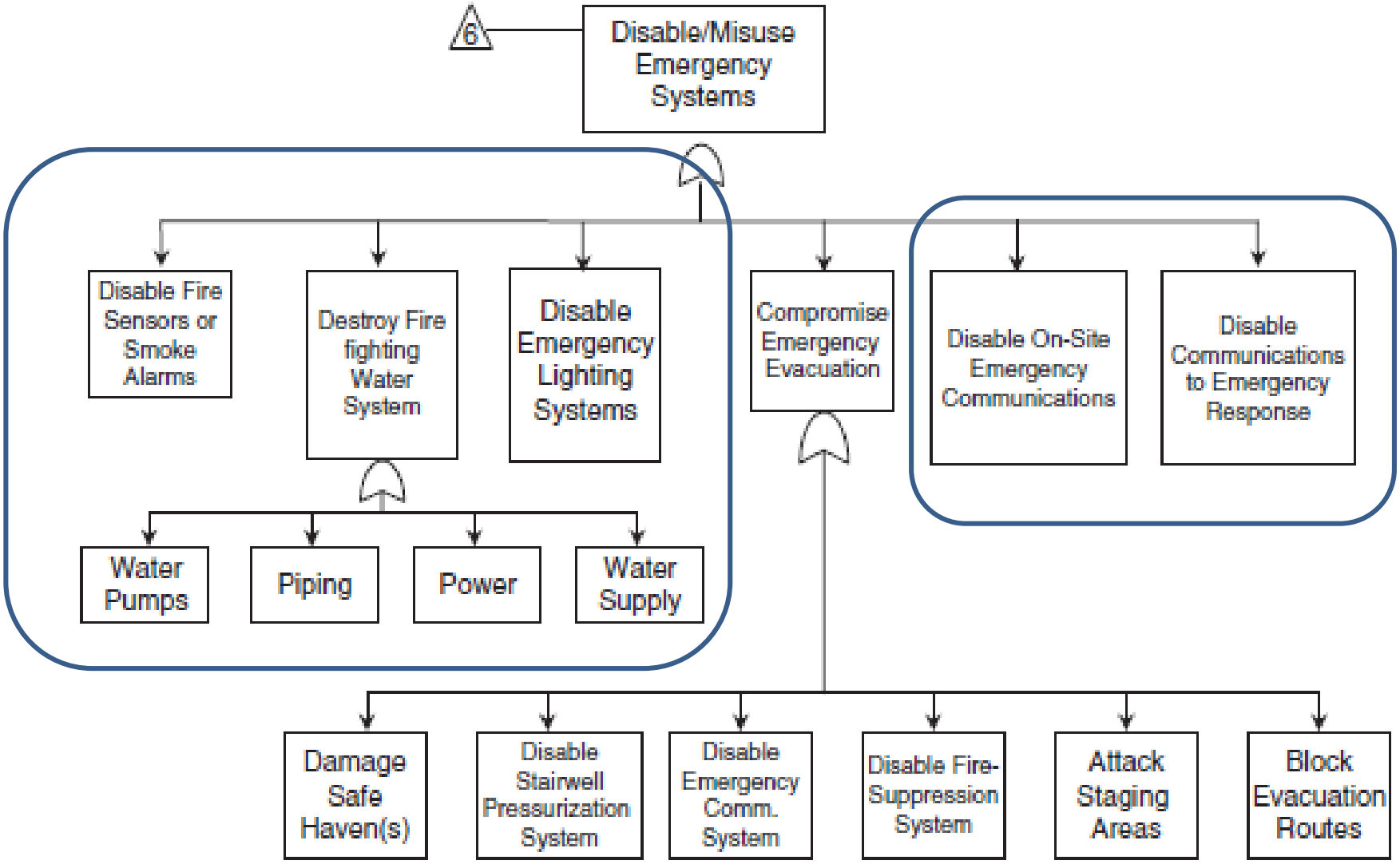
Example – theme and epics

Theme: a safer building

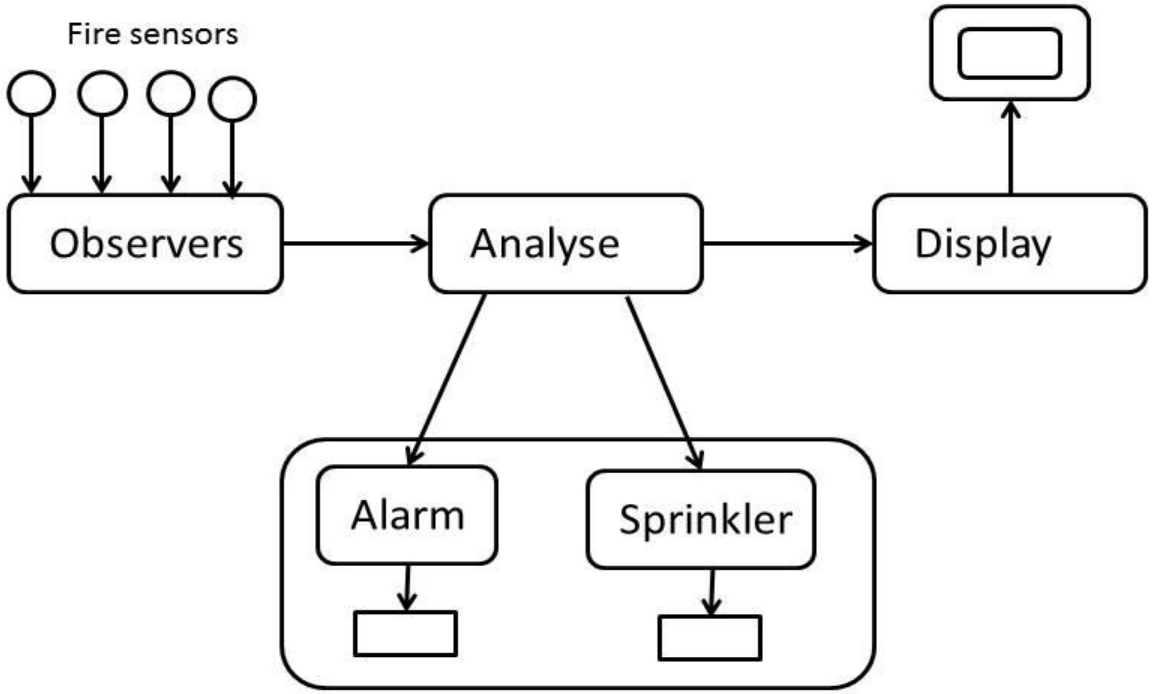
Epic ID:	**Fire alarm (1)**
As a	House owner
I want	To discover fire as quickly as possible
So that	I can evacuate people as early as possible

Epic ID:	**Fire alarm (2)**
As a	Fire brigade
I want	To discover the location of a fire as quickly as possible
So that	I can put out the fire as simple as possible

Generic fault tree for a building – fire fighting



Fire alarm pattern



Components:

- Fire sensors
- Alarm
- Alarm display
- Sprinkler
- Analyser - control unit

Example - PHA

Based on Epic 1 and Epic 2

Hazard	Cause	Main effect	Preventive action
No alarm in building	Alarm system failure Power failure	No or delayed evacuation	Periodic testing UPS
No alarm to fire brigade	Alarm system failure Power failure Transmission failure	No or delayed fire brigade	Periodic testing UPS Ping on transmission lines
False alarm	Alarm system failure	Unnecessary evacuation	-

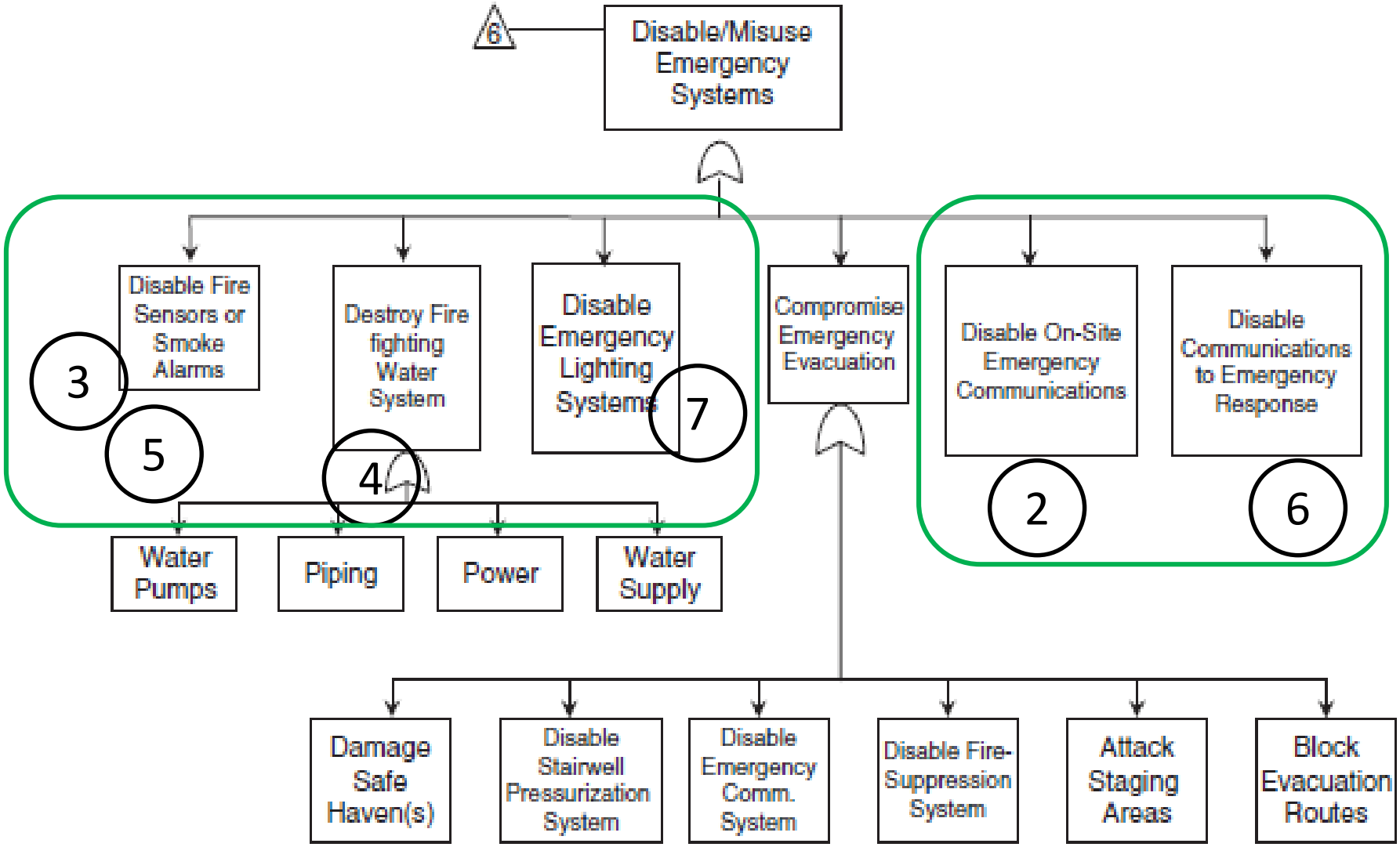
Choosing “The system”

There is a tight coupling between

- alarm system => discover and inform
- fire fighting system => put out or control
- the environment – the rest of the building.

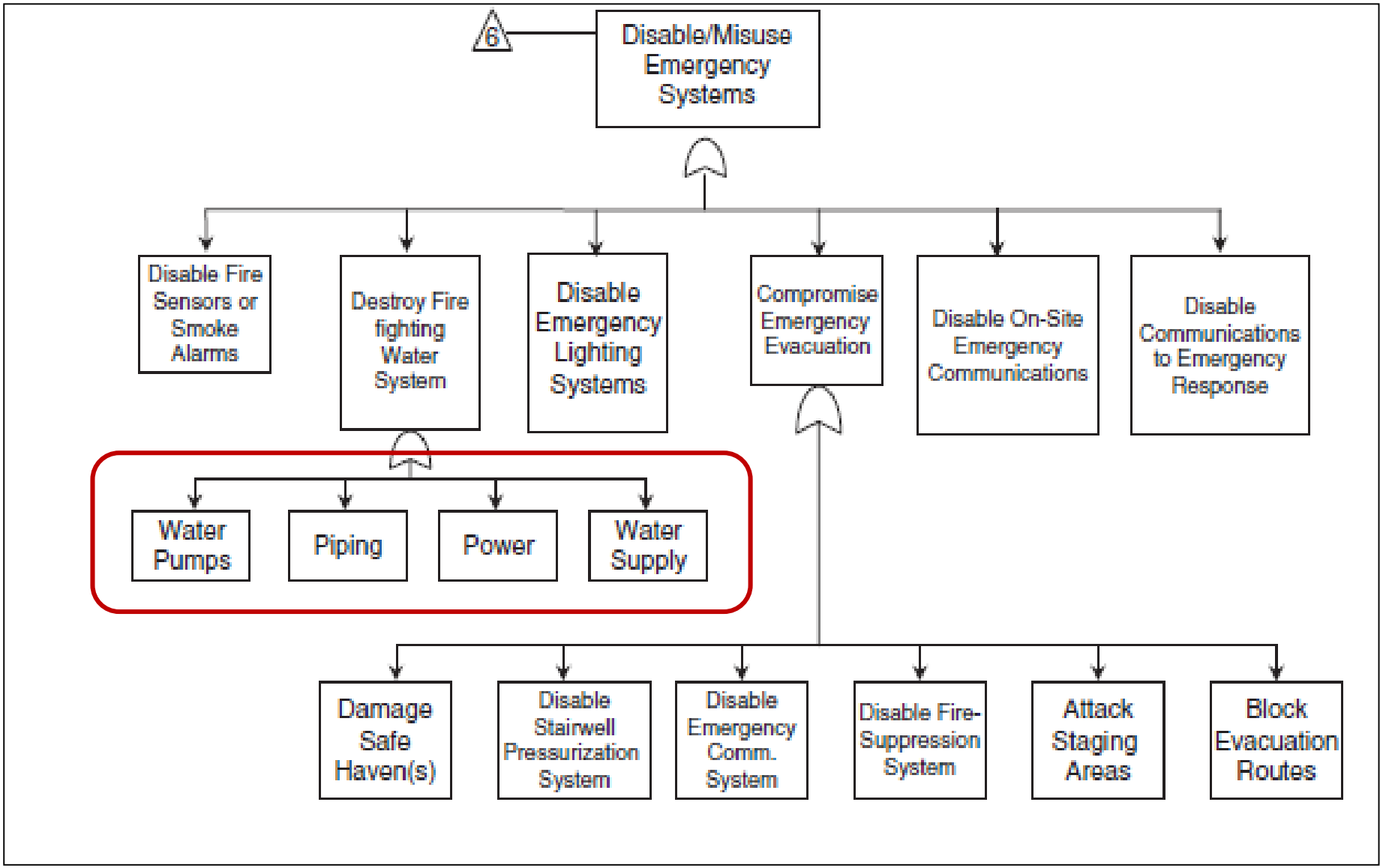
It is important to decide what is inside and what is outside the system

Our area of concern – inside

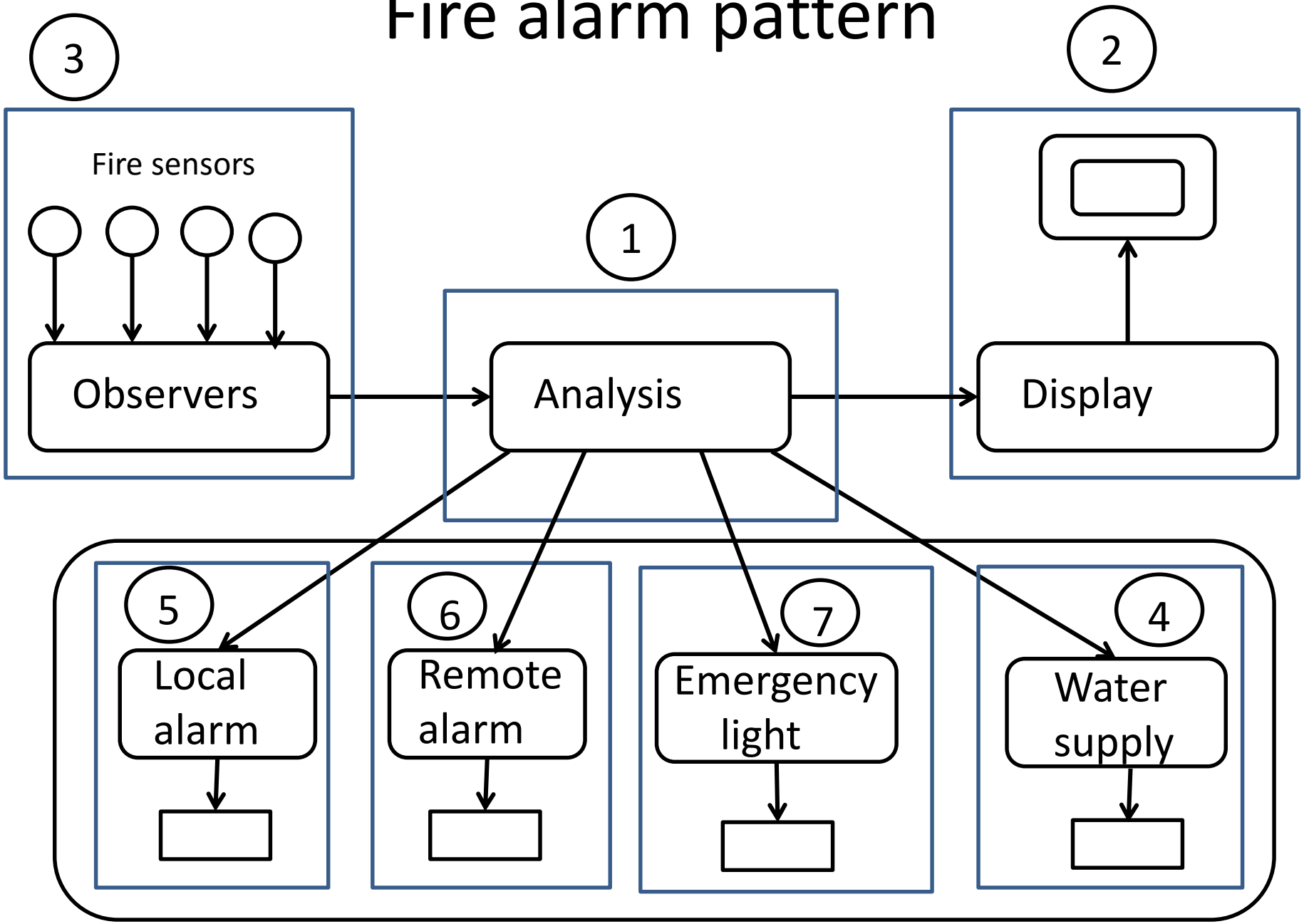


The environment – outside

Important to define what is inside and what is outside the system



Fire alarm pattern



From Epic to User stories

Epic ID:	**Fire alarm**
As a	House owner
I want	To discover fire as quickly as possible
So that	I can evacuate people as early as possible

Story ID:	**Fire display - 2**
As a	House owner
I want	To know where the fire is
So that	I can evacuate persons in the area

Story ID:	**Local alarm – 5**
As a	House owner
I want	To be made aware of the fire
So that	I can start necessary actions – e.g. call the fire brigade

User story IF-FMEA

Based on the observe – react pattern

Story ID: Local alarm – 5		**List of component input sources:** • **Analysis**		**Suggested barriers and new requirements**
Output failure mode	**Output failure mode description**	**Input deviation**	**Alarm component failure**	
Omission	No alarm	No alarm trigger	Alarm unit malfunction Lack of power	Equipment • Duplication • Periodic testing Pinging connection to analysis
Commission	Extra alarm	Extra alarm trigger	Alarm unit short-circuit	
Wrong action	No / false alarm	No / false alarm trigger	Alarm unit • malfunction • short-circuit	Periodic testing / maintenance
Delayed	Alarm delayed	Delayed trigger	-	-

Main conclusions

We can start safety analysis early in the development process if we

- Get the most important, high level requirements in place early
- Decide what is inside and what is outside our system
- Use
  - Generic failure modes and architectural patters
  - Domain specific fault trees and hazard lists