Foundations of Software Engineering

Lecture 7: Requirements Validation and Risk

Claire Le Goues

institute for SOFTWARE RESEARCH

Examples adapted arbitrarily from prior years without identifying information!

REFLECTIONS ON REFLECTIONS

Reflection documents

Shallow

- Recite facts about what happened without adding anything.
- Recite statements from class without connecting to experience.
- State lessons learned without any reason why.

Good

- Extrapolate from the facts to add insight.
- Meaningfully connect prior experience or class material to assignment experience.
- Support lessons learned with evidence.

Shallow reflection examples

[PROCESS]

At our first meeting, we developed an initial outline of our approach. This was followed by preparing a list of tasks which were required for implementing the X system. Next, we divided the tasks among ourselves and came up with a rough timeline of the process to be followed."

[SCHEDULE]

"Although we managed to meet all the milestones and implement all the desired features, the exact dates for the same could not be followed towards the end."

[PLANNING]

"Learning how to use API X took a little longer than expected, which caused a setback of a day; but overall we managed to complete the entire project before the deadline and adhered to the timeline."

[TEAM WORK / COMMUNICATION]

"We all agreed to use tool Y to keep in touch. We used it to announce when we started or completed individual tasks, current milestone statuses.. We also used Y to schedule a group meeting for the integration portion of our coding assignment"

Good reflection examples

[PLANNING / PROCESS]

"Since I was interested in the planning, we decided as a team I would be in charge of documenting our progress.. It worked really well to have one person managing what needed to get done or who needed to do it, and ensuring a shared single vision and set of goals as a group. However, there exist negatives approaching things this way…I found that my teammates sometimes would rely on me too heavily."

[TEAM WORK / COMMUNICATION]

"An example of something that [would] work well is...issue tracking – something I asked them to do since first meeting. It's easy to forget this information over time... If we had simply reminded ourselves on a regular basis, we would have had fewer problems forgettng these things."

[PLANNING]

"People seemed to be annoyed because X "was not doing any work". I believe X did the least amount of work, but we also assigned X the least amount of work. I wonder if this can all be traced back to the fact that X could not attend our first group meeting"

More good examples

[TEAMWORK]

"It helps to say 'thank you' before complaining about a teammate's work. Only take conflict-inducing action if you think it is extremely important and are willing to follow up. Otherwise, you are wasting everyone's time. Would we have treated each other differently if we had known we would be partnered up on more than just this assignment for the class?"

[TEAMWORK]

"two takeaways I had from this project are :

– It is best to present yourself as someone who is willing to help out, and do more than what was originally asked of you. This way, if people decline your offer to help out, they will be okay with the fact that you may not be working as hard as them at that point in time.

– Respect other people's time and work, and take that into consideration when you decide to criticize their work or bring up issues. "

Also

- The homework document includes bulleted lists and prose outlining what a "good solution" looks like.
- Consider checking your submission against it, at the very least before submitting, if not sooner.

Learning goals

- Differentiate between verification and validation.
- Explain the purpose of requirements decomposition, allocation, and flowdown.
- Identify strategies for dealing with conflicts.
- Understand risk and its role in requirements, specifically how it can be modeled, analyzed, and then mitigated/handled in system design.

Industrial Requirements Tools

'Stakeholder Requirements' current 2.2 (Review Phase 3) in /New Family Car Project/Requirements (Formal module) - DO...

File Edit View Insert Link Analysis Table Tools Discussions User RQM Help

View 01 - Collect reqts | All levels

- Stakeholder Requirements
  - 1 Introduction
  - 2 User types
    - 2.1 Nationalities
      - The car will be usec
    - 2.2 User sizes
      - People come in all s
  - 3 Requirements
    - 3.1 Capability Requirem
      - 3.1.1 Carrying Capa
      - 3.1.2 Cost Points
      - 3.1.3 Movement
      - 3.1.4 Fuel economy
      - 3.1.5 Safety
      - 3.1.6 Noise levels
      - 3.1.7 Ease of Acces
      - 3.1.8 Visibility
      - 3.1.9 Equipment ma
      - 3.1.10 Entertainmer
      - 3.1.11 Maintenance
      - 3.1.12 Servicing
      - 3.1.13 Indication rec
      - 3.1.14 Terrain
      - 3.1.15 Refueling
    - 3.2 Constraint Requirem
      - 3.2.1 Availability
      - 3.2.2 Lifetime
      - 3.2.3 Security
      - 3.2.4 Accessories

Reqt	Car user requirements	Priority	Acceptability	Queries
TRN-CSR-83	Users shall be able to receive a warning when a service is due.	Mandatory	Acceptable	(JC) What color indicators are we using for the warning system? Are we going to put requirements in place to accomidate this system?
TRN-CSR-84	**3.1.13 Indication requirements**	N/A		
TRN-CSR-85	The user shall be able to see at all times an indication of speed to within + or - 1%.	1	Acceptable	
TRN-CSR-86	The user shall be able to see at all times an indication of engine revolutions to within + or - 1%.	2	Acceptable	
TRN-CSR-92	The user shall be able to obtain direction to go information. 	2	Acceptable	

Username: lbus | Exclusive edit mode

Requirements should be

1. Correct
2. Consistent
3. Unambiguous
4. Complete
5. Feasible
6. Relevant
7. Testable
8. Traceable

According to both the engineer and the customer

In that there are no conflicting requirements. Quality requirements

Ambiguous: multiple readers can walk away with different but valid

Covers all required behavior and output for all inputs under all

Can it be do n, quality/ non-functi articularly

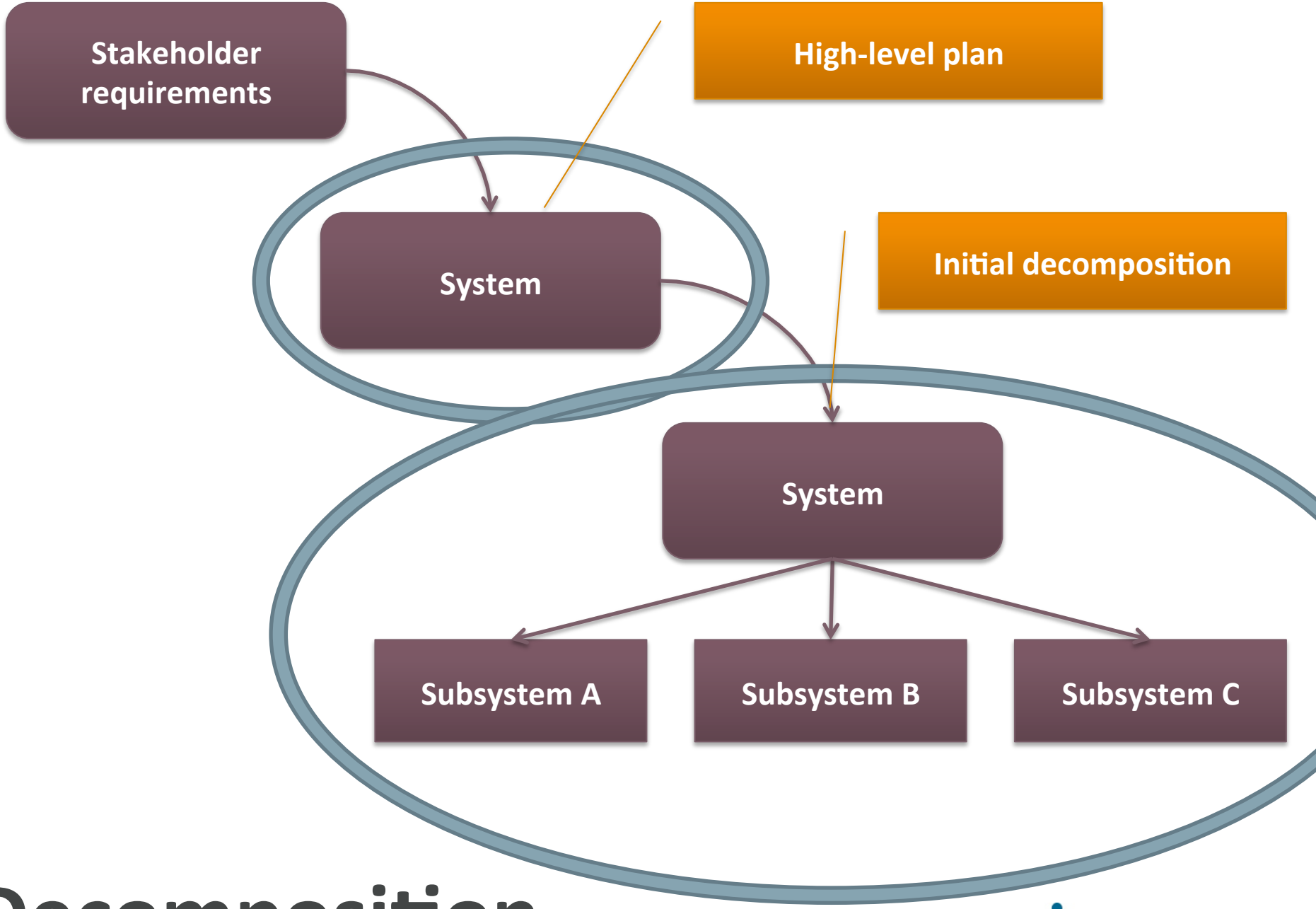
Acceptance tests and metrics are

Organized, uniquely labeled.

Verification vs. Validation

- Verification – is the software correct?
  - Does the software satisfy the specification?
  - Is the specification correct with respect to the requirements, assuming the domain properties hold?
- **Validation – are the requirements correct?**
  - Are the requirements complete? Do they accurately reflect the client's problem?
  - Are the requirements consistent?

Validation	walkthroughs
	Reading
	Interviews
	Reviews
	Checklists
	Models to check functions and relationships
	Scenarios
	Prototypes
	Simulation
	Formal inspection
Verification	Cross-referencing
	Simulation
	Consistency checks
	Completeness checks
	Checks for unreachable states or transitions (cf. Model checking)
	Mathematical proofs



Decomposition

Why?

- Decomposition into a hierarchy helps establish, **traceability**, which identifies relationships between requirements.
- **Traceability** is important for when requirements change.
- Decomposition also helps both *validate* and *verify* the requirements.

Allocation and flowdown

- Grouping (e.g., by business function) enables analysis
- Allocation is the assignment of requirements to subordinate systems
- Flowdown is the discovery of additional requirements from the allocated requirements

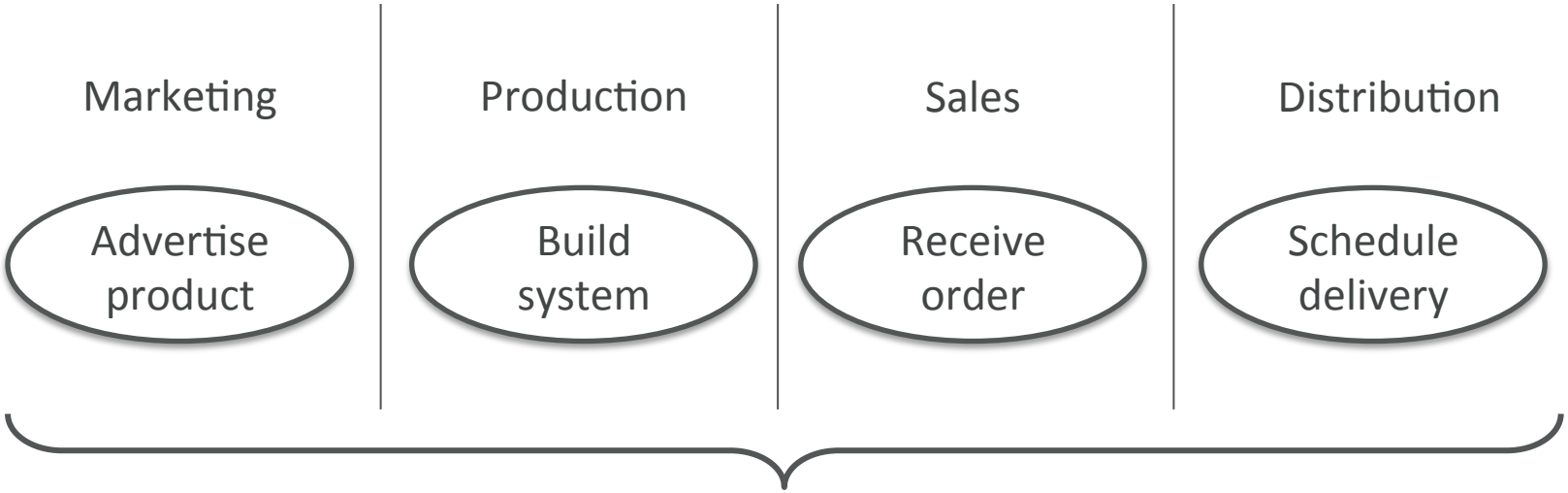
Group by business function

Business functions provide logical groups for use cases

- Marketing
- Human Resources
  - Hiring and Recruitment
  - Benefits
  - Performance Reviews
  - Promotion
  - Retirement
- Production
- Sales
  - Shipping
  - Receiving
  - Inventory
- Distribution

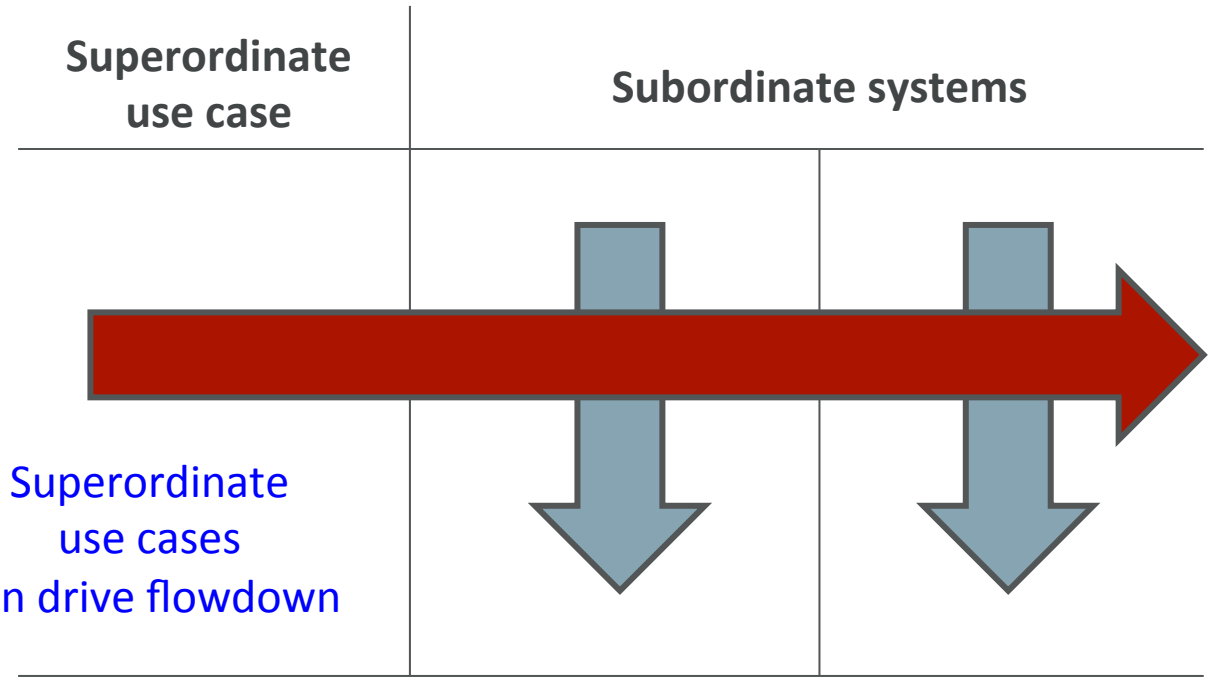
Group by superordinate system

- Superordinate systems cut across subordinate systems, such as business functions



A superordinate system that traces a product from marketing to distribution.
The analyst can now focus on developing the interfaces
between these use cases

Allocation and flowdown

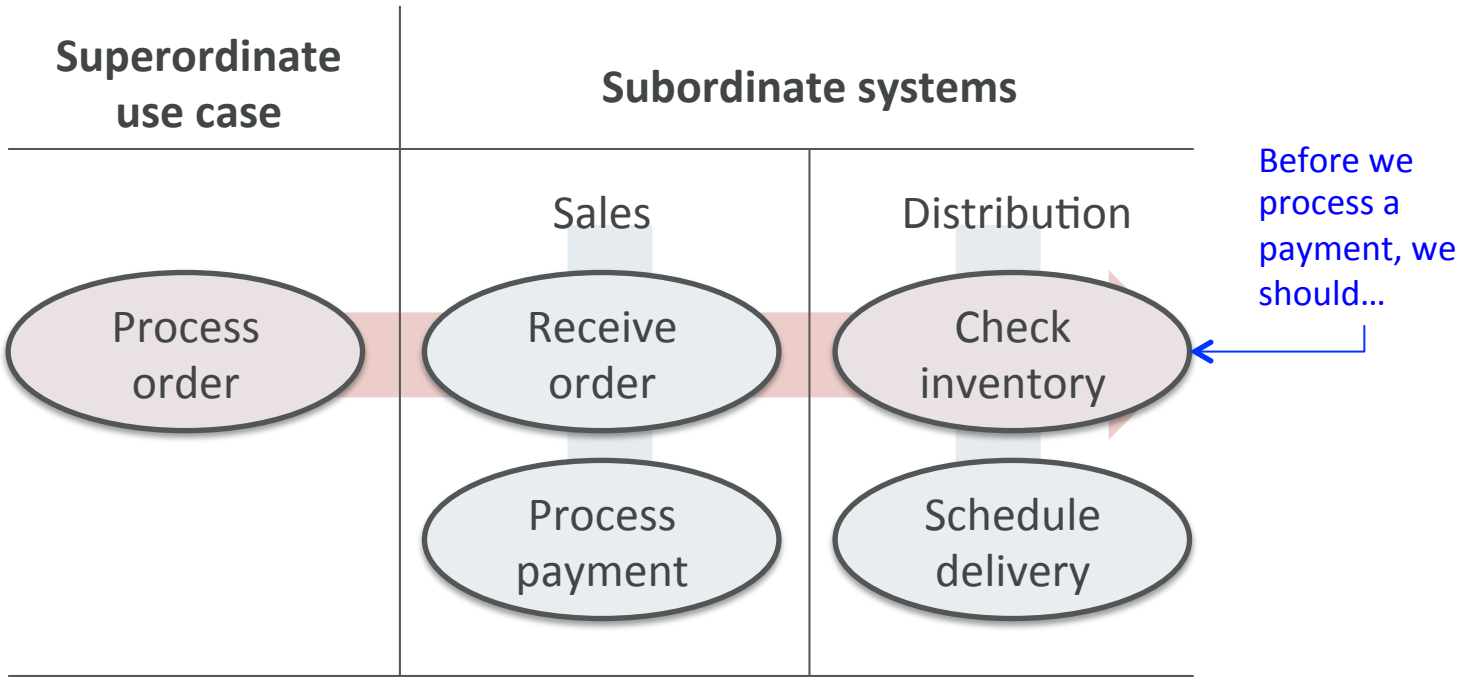


Superordinate use cases can drive flowdown

Allocate use cases to subordinate systems

Flowdown discovers “derived requirements” to fulfill the allocated requirements

Allocation and flowdown



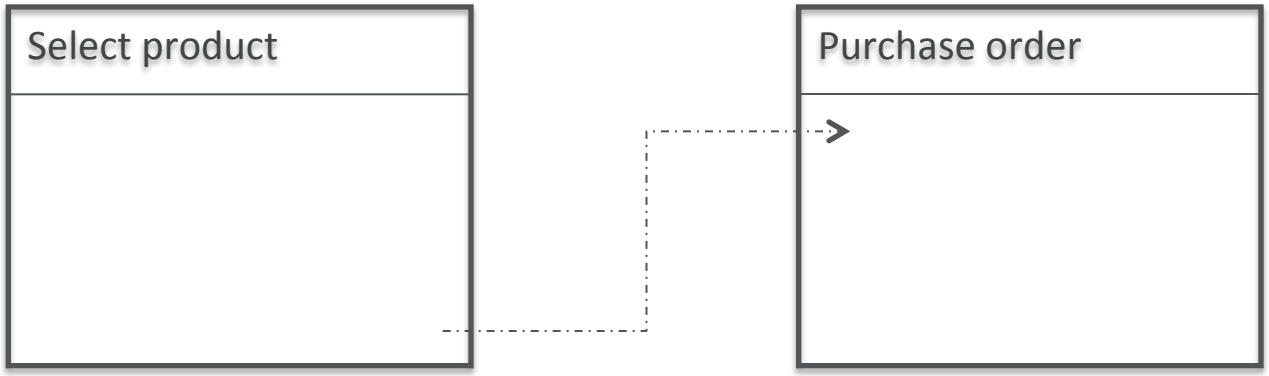
Allocate use cases to subordinate systems

Flowdown discovers “derived requirements” to fulfill the allocated requirements

Dependency Streams

- Dependency streams describe use cases that depend on each other
  - Matching pre- and post-conditions
  - Performing actions in parallel
  - Highlight interfaces between business functions

Matching pre- and post-conditions

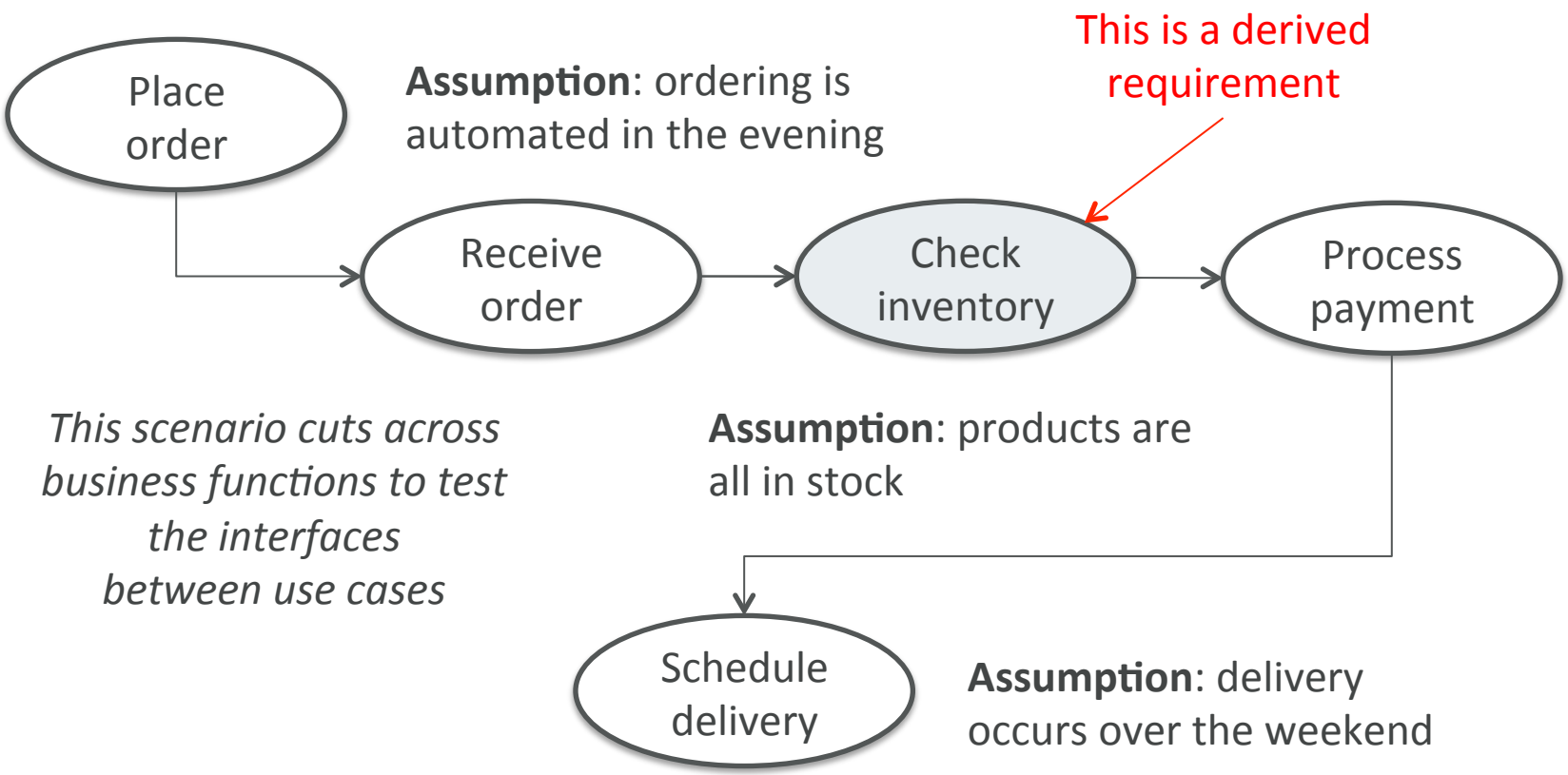


Post-condition: Customer has selected products

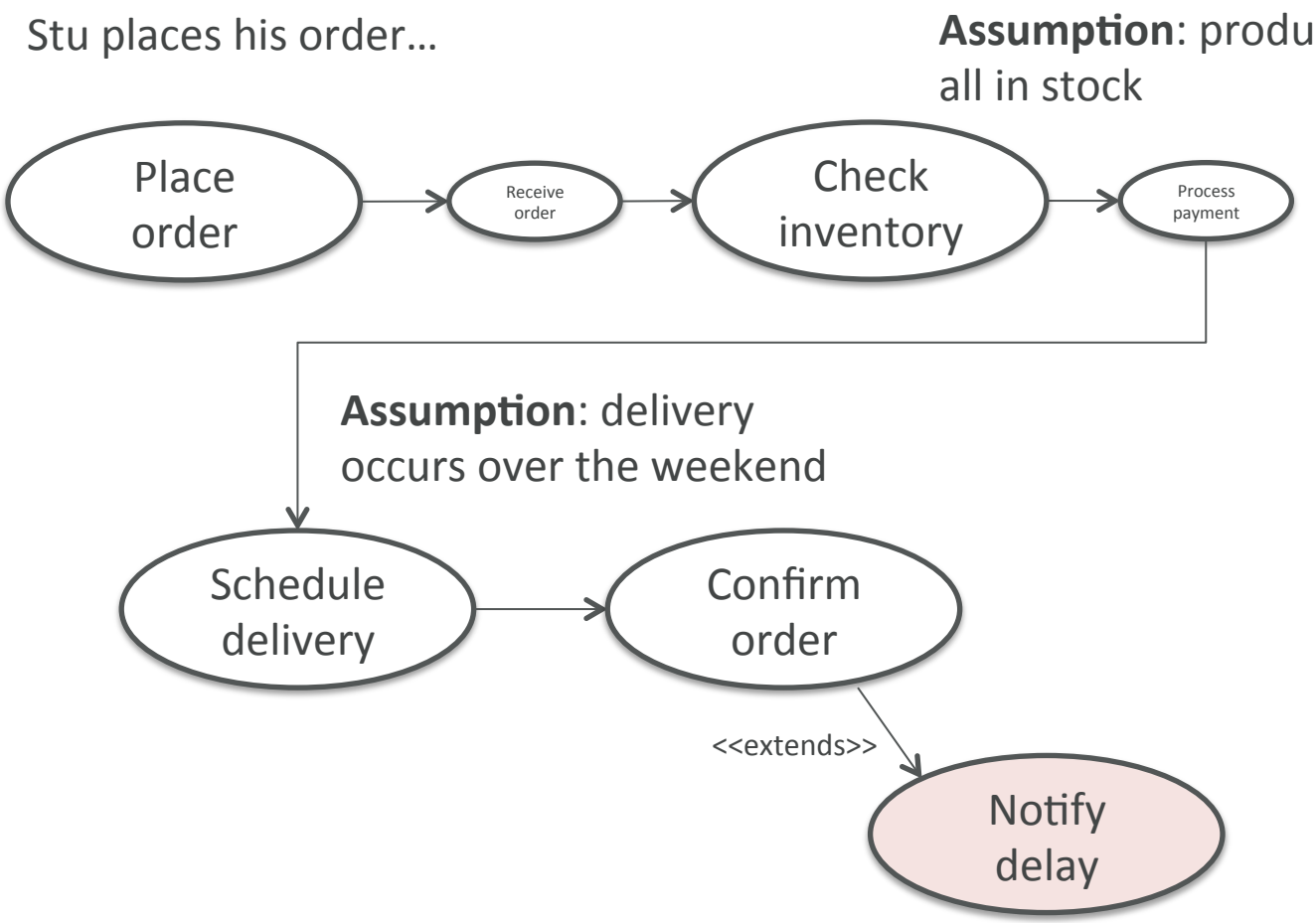
Pre-condition: Customer is ready to check out

Tracing scenarios through cases

Stu places his order…



Modifying use cases for scenarios



Prioritizing Use Cases

- Assigning a priority to a use case to increase or decrease the activity's importance.
- What are some options?
  - Customer priority
  - Risk
  - Complexity
  - Dependencies
  - Core functionality
  - User-facing activities
  - Uncertainty (same as risk?)

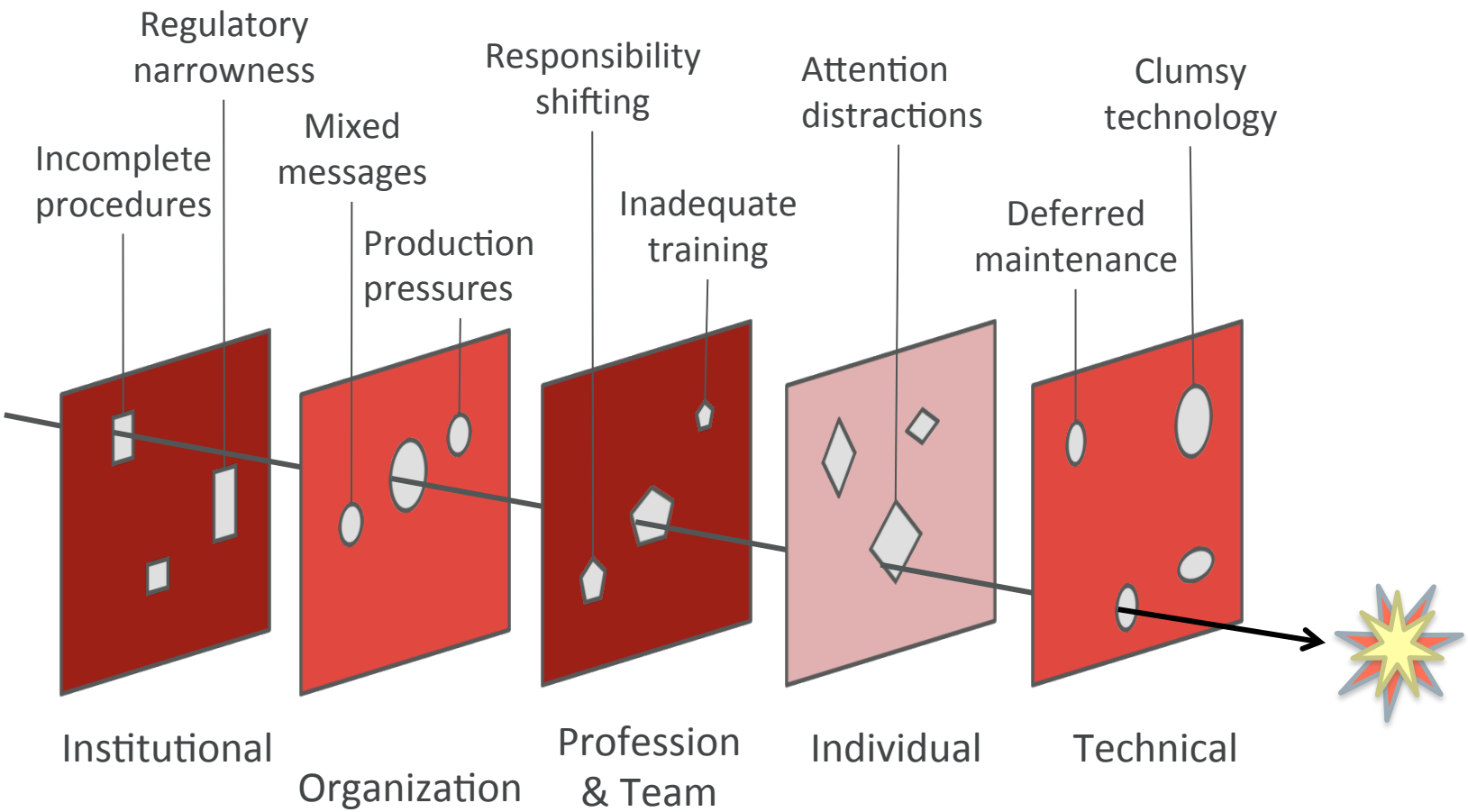
What are risks?

- A **risk** is an uncertain factor that may result in a loss of satisfaction of a corresponding objective

For example…

- System delivers a radiation overdose to patients (Therac-25, Theratron-780)
- Medication administration record (MAR) knockout
- Premier Election Solutions vote-dropping "glitch"

The Swiss cheese model

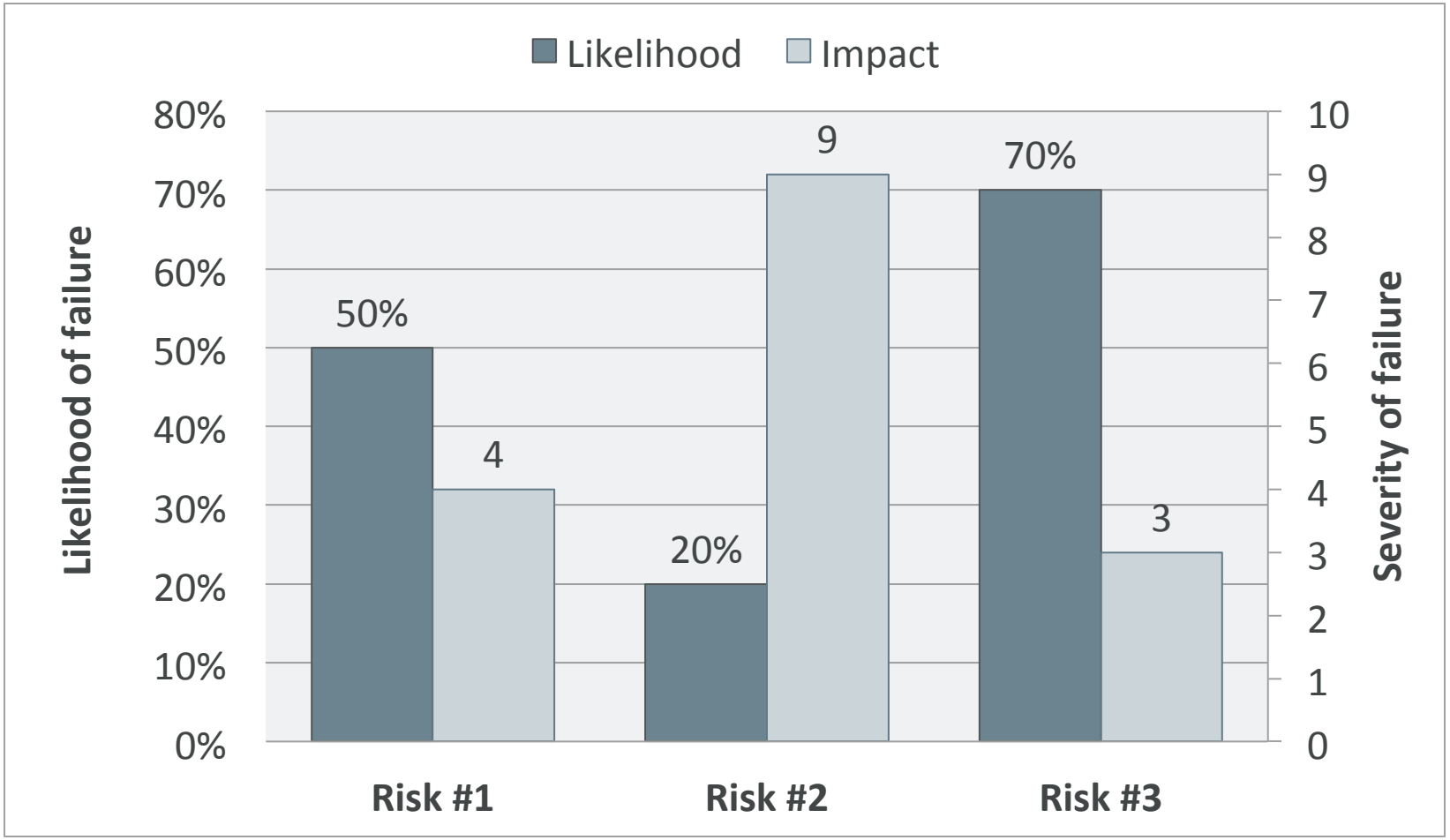


Modified from Reason, 1999, by R.I. Crook

How to assess the level of risk?

- Risks consist of multiple parts:
  - Likelihood of failure
  - Negative consequences or impact of failure
  - Causal agent and weakness (in advanced models)
- Risk = Likelihood x Impact

Likelihood vs. Impact Severity



institute for SOFTWARE RESEARCH

CVSS V2.10 Scoring

The Common Vulnerability Scoring System consists of:

- 6 base metrics (access vector, complexity, confidentiality impact, …)
- 3 temporal metrics (exploitability, remediation, …)
- 5 environmental metrics; all qualitative ratings (collateral damage, …)

BaseScore =

round_to_1_decimal(((0.6*Impact)+(0.4*Exploitability)–1.5)*f(Impact))

Impact =

10.41*(1-(1-ConfImpact)*(1-IntegImpact)*(1-AvailImpact))

Exploitability =

20* AccessVector*AccessComplexity*Authentication

f(impact) = 0 if Impact=0, 1.176 otherwise

Aviation failure impact categories

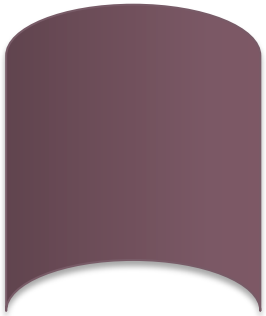
- No effect – failure has no impact on safety, aircraft operation, or crew workload
- Minor – failure is noticeable, causing passenger inconvenience or flight plan change
- Major – failure is significant, causing passenger discomfort and slight workload increase
- Hazardous – high workload, serious or fatal injuries
- Catastrophic – loss of critical function to safely fly and land

DO-178b, Software Considerations in Airborne Systems and Equipment Certification, RTCA, 1992

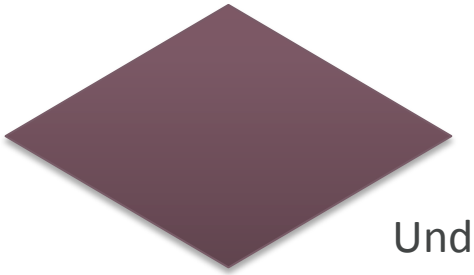
Fault tree analysis

- Top-down analysis technique to model, reason about, and analyze risk.
- Decompose a particular type of failure into constituent potential causes and probabilities.
- Define scope of system responsibilities, identify unacceptable risk conditions that should be mitigated.

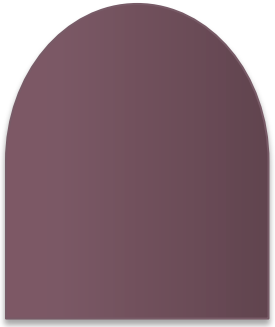
Top-level or intermediate event



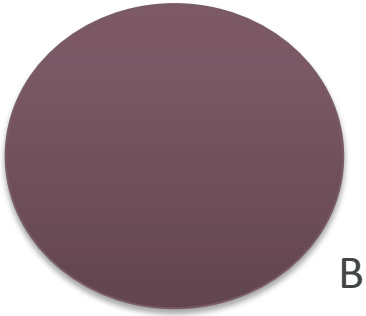
Or gate



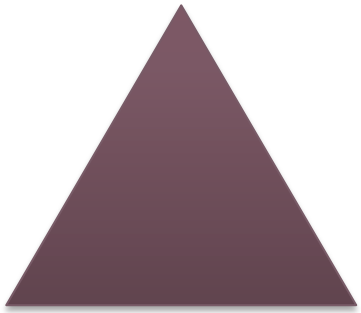
Undeveloped event



And gate

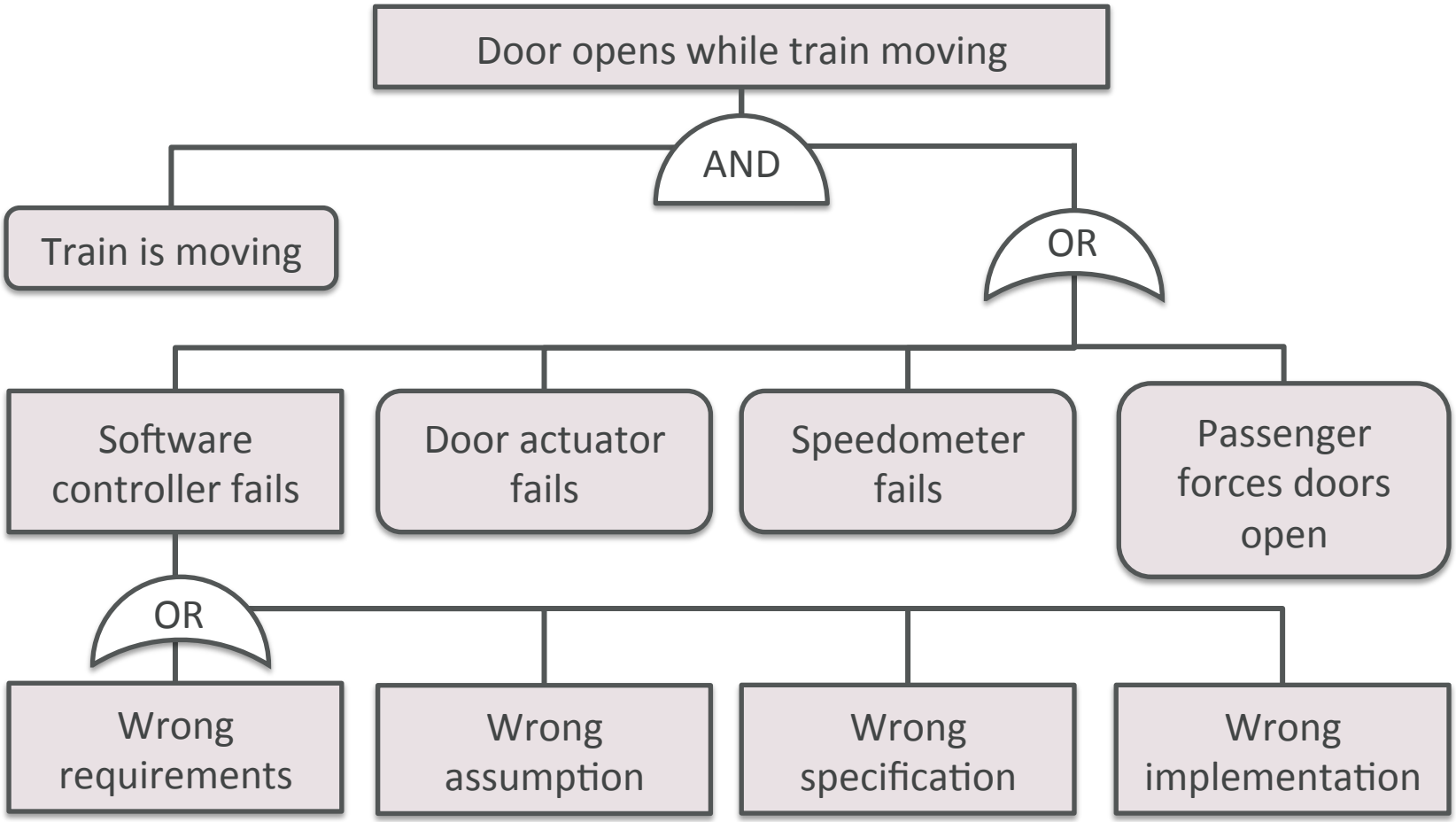


Basic event



Transfer gate

Fault trees to quantify risk



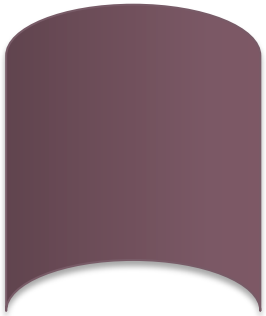
Risk mitigation/response strategies

- Accept the risk – for low likelihood or low impact risks, or where cost of mitigation precludes system
- Transfer the risk – push the risk outside the system boundary
- Mitigate the risk – introduce active countermeasures
  - Reduce likelihood of failure
  - Reduce severity of impact
  - Change ors to ands!
- Avoid the risk – redesign so that risk cannot occur

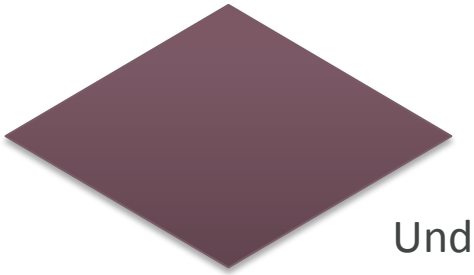
Exercise!

- Unacceptable system failure: a given CMU student fails a midterm.

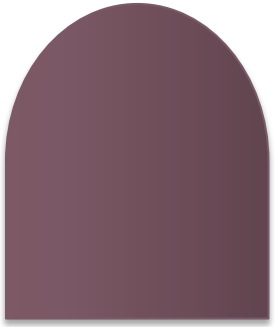
Top-level or intermediate event



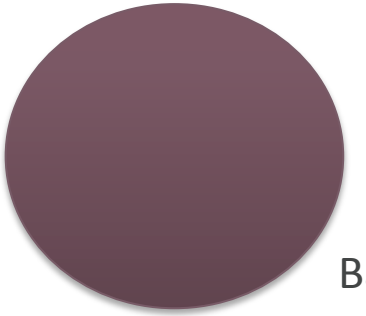
Or gate



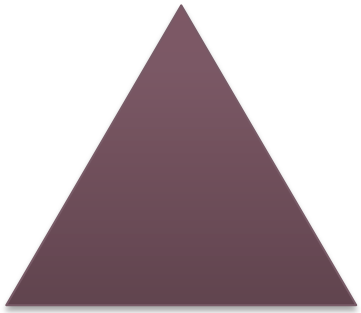
Undeveloped event



And gate



Basic event



Transfer gate