CASE WESTERN RESERVE
UNIVERSITY

Requirements Engineering

CSDS 393/493: Software Engineering

Spring 2025

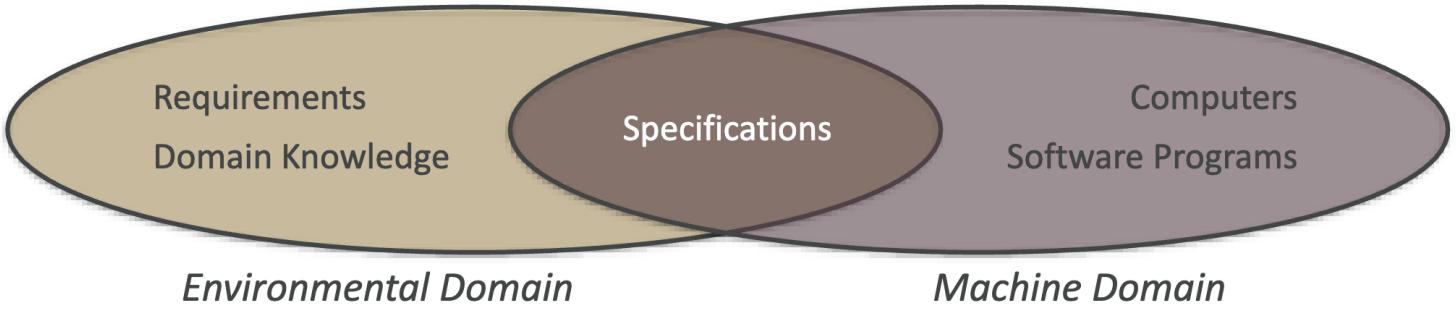
Agenda

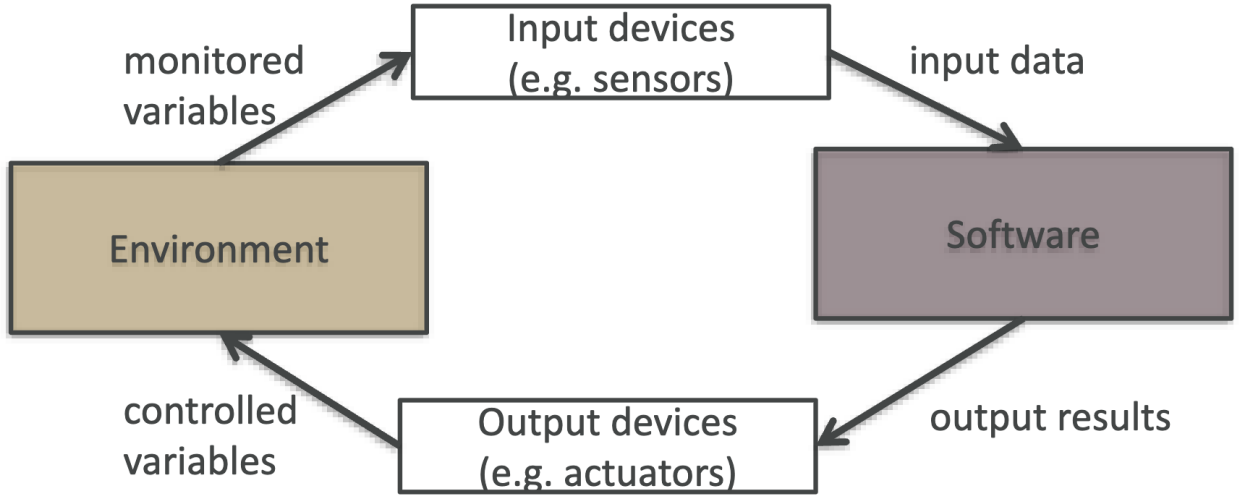
- Requirement engineering activities (cont.)
  - Case study
  - Prototypes, stories, etc.
- Requirement management
  - Validation, prioritization, etc.
  - Graphical models
  - Evolution
- Risks

Project Proposal and Weekly Meetings

- Proposal is due January 28
- By this week schedule weekly meeting with the supervisor TA
  - Discuss ideas
  - Responsibilities of team members
  - Confirm features
  - Demos

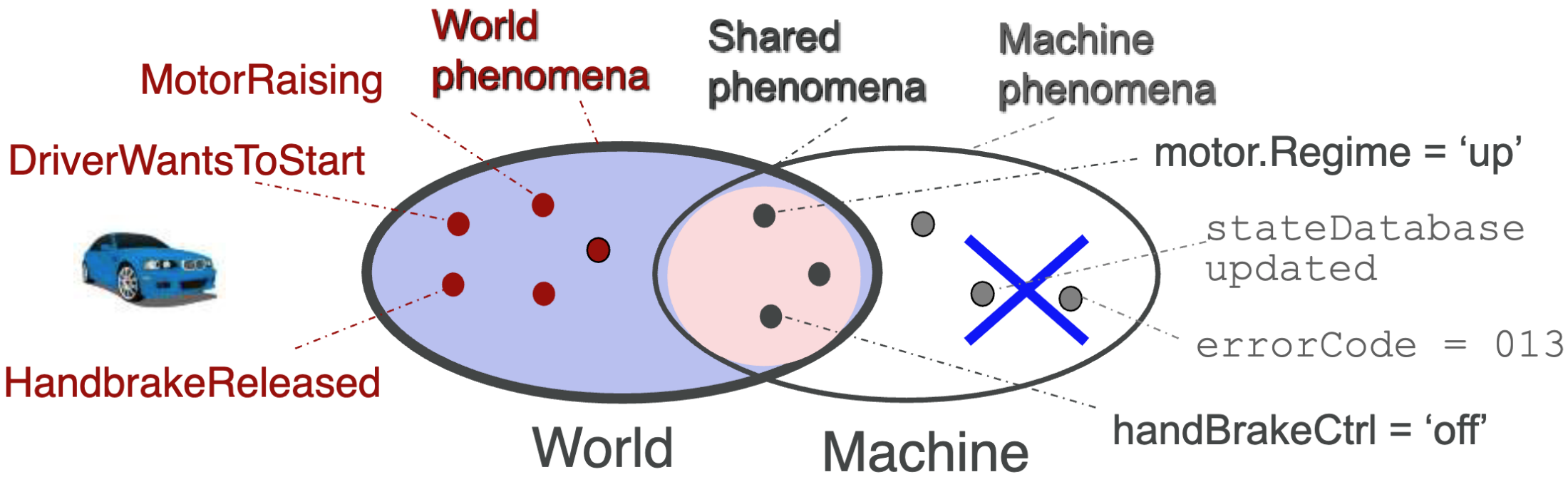
Environment and the Machine





Zave, Pamela, and Michael Jackson. "Four dark corners of requirements engineering." *ACM transactions on Software Engineering and Methodology (TOSEM)* 6.1 (1997): 1-30.

Environment and the Machine



Case Study: Web Video

Typical Steps

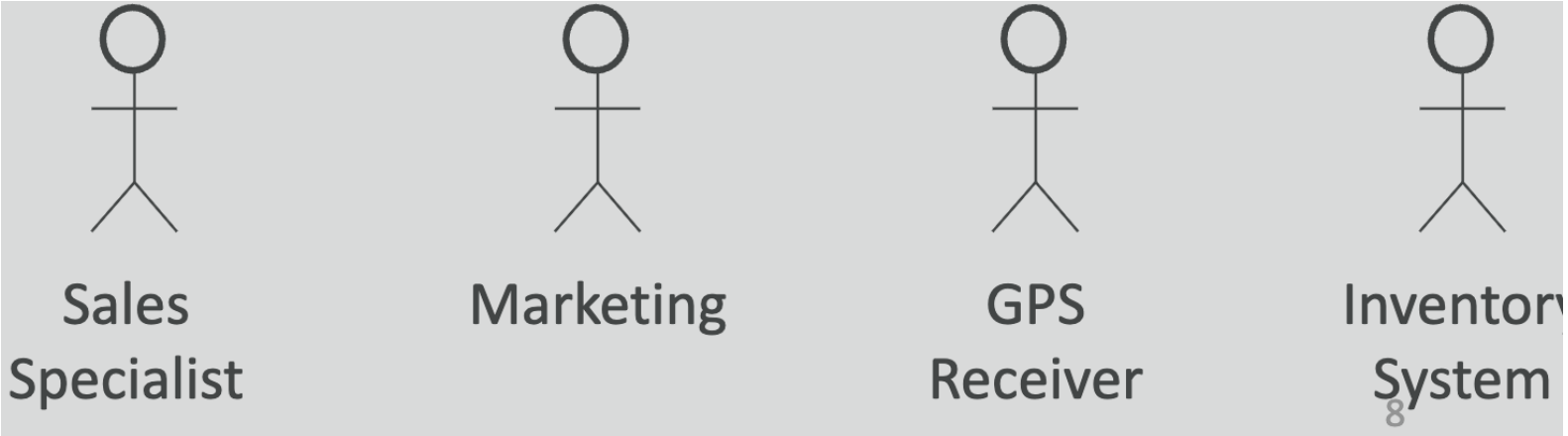
- Identify stakeholders
- Understand the domain
  - Analyze artifacts, interact with stakeholders
- Discover the real needs
  - Interview stakeholders
- Explore alternatives to address needs

Questions

- **Who** is the system for?
- Stakeholders:
  - End users
  - System administrators
  - Engineers maintaining the system
  - Business managers
  - ...who else?

Defining Actors/Agents

- An actor is an entity that interacts with the system for the purpose of completing an event [Jacobson, 1992].
  - Not as broad as stakeholders.
- Actors can be a user, an organization, a device, or an external system.



Personas

- Fictional character created to represent a user type
- Represent specific segments or skills
- Useful to think of diverse use cases
- Can be based on established cultural dimensions

https://gendermag.org/foundations.php

Abby Jones[1]

You can edit anything in blue print

- 28 years old
- Employed as an Accountant
- Lives in Cardiff, Wales

Abby has always liked music. When she is on her way to work in the morning, she listens to music that spans a wide variety of styles. But when she arrives at work, she turns it off, and begins her day by scanning all her emails first to get an overall picture before answering any of them. (This extra pass takes time but seems worth it.) Some nights she exercises or stretches, and sometimes she likes to play computer puzzle games like Sudoku

Background and skills

Abby works as an accountant. She is comfortable with the technologies she uses regularly, but she just moved to this employer 1 week ago, and their software systems are new to her.

Abby says she's a "numbers person", but she has never taken any computer programming or IT systems classes. She likes Math and knows how to think with numbers She writes and edits spreadsheet formulas in her work.

In her free time, she also enjoys working with numbers and logic. She especially likes working out puzzles and puzzle games, either on paper or on the computer

Motivations and Attitudes

- ***Motivations***: Abby uses technologies to accomplish her tasks. She learns new technologies if and when she needs to, but prefers to use methods she is already familiar and comfortable with, to keep her focus on the tasks she cares about.
- ***Computer Self-Efficacy***: Abby has low confidence about doing unfamiliar computing tasks. If problems arise with her technology, she often blames herself for these problems. This affects whether and how she will persevere with a task if technology problems have arisen.
- ***Attitude toward Risk:*** Abby's life is a little complicated and she rarely has spare time. So she is risk averse about using unfamiliar technologies that might need her to spend extra time on them, even if the new features might be relevant. She instead performs tasks using familiar features, because they're more predictable about what she will get from them and how much time they will take.

How Abby Works with Information and Learns:

- ***Information Processing Style:*** Abby tends towards a *comprehensive information processing style* when she needs to more information. So, instead of acting upon the first option that seems promising, she gathers information comprehensively to try to form a complete understanding of the problem before trying to solve it. Thus, her style is "burst-y"; first she reads a lot, then she acts on it in a batch of activity.
- ***Learning: by Process vs. by Tinkering***: When learning new technology, Abby leans toward process-oriented learning, e.g., tutorials, step-by-step processes, wizards, online how-to videos, etc. She doesn't particularly like learning by tinkering with software (i.e., just trying out new features or commands to see what they do), but when she does tinker, it has positive effects on her understanding of the software.

https://gendermag.org/foundations.php

Combining Techniques

- Many combined and more specific approaches
- For example Contextual Inquiry:
  - workplace observation +
  - open-ended interviews +
  - prototyping

Resolving Conflicts

- **Terminology clash**: same concept named differently
  - e.g., library management: “borrower” vs. “patron”
- **Designation clash**: same name for different concepts
  - e.g. “user” for “library user” vs. “library software user”
- **Structure clash**: same concept structured differently
  - e.g. “latest return date” as time point (e.g. Fri 5pm) vs. time interval (e.g. Friday)

Types of Inconsistency

- **Strong conflict**: statements not satisfiable together
  - "participant constraints may not be disclosed to anyone else" vs.
  - "the meeting initiator should know participant constraints"

- **Weak conflict** (divergence): statements not satisfiable together under some boundary condition
  - "patrons shall return borrowed copies within X weeks" vs
  - "patrons shall keep borrowed copies as long as needed" contradict only if "needed>x weeks"

Handling Inconsistencies

- Terminology, designation, structure: Build glossary, domain model

- Weak, strong conflicts: Negotiation required
  - Cause: different objectives of stakeholders => resolve outside of requirements
  - Cause: quality tradeoffs => explore preferences

* Various specific processes, heuristics, and techniques exist for identifying and resolving conflicts.

Implementation Bias

- Requirements say what the system will do (and not how it will do it).
  - Why not "how"?

Avoiding implementation bias

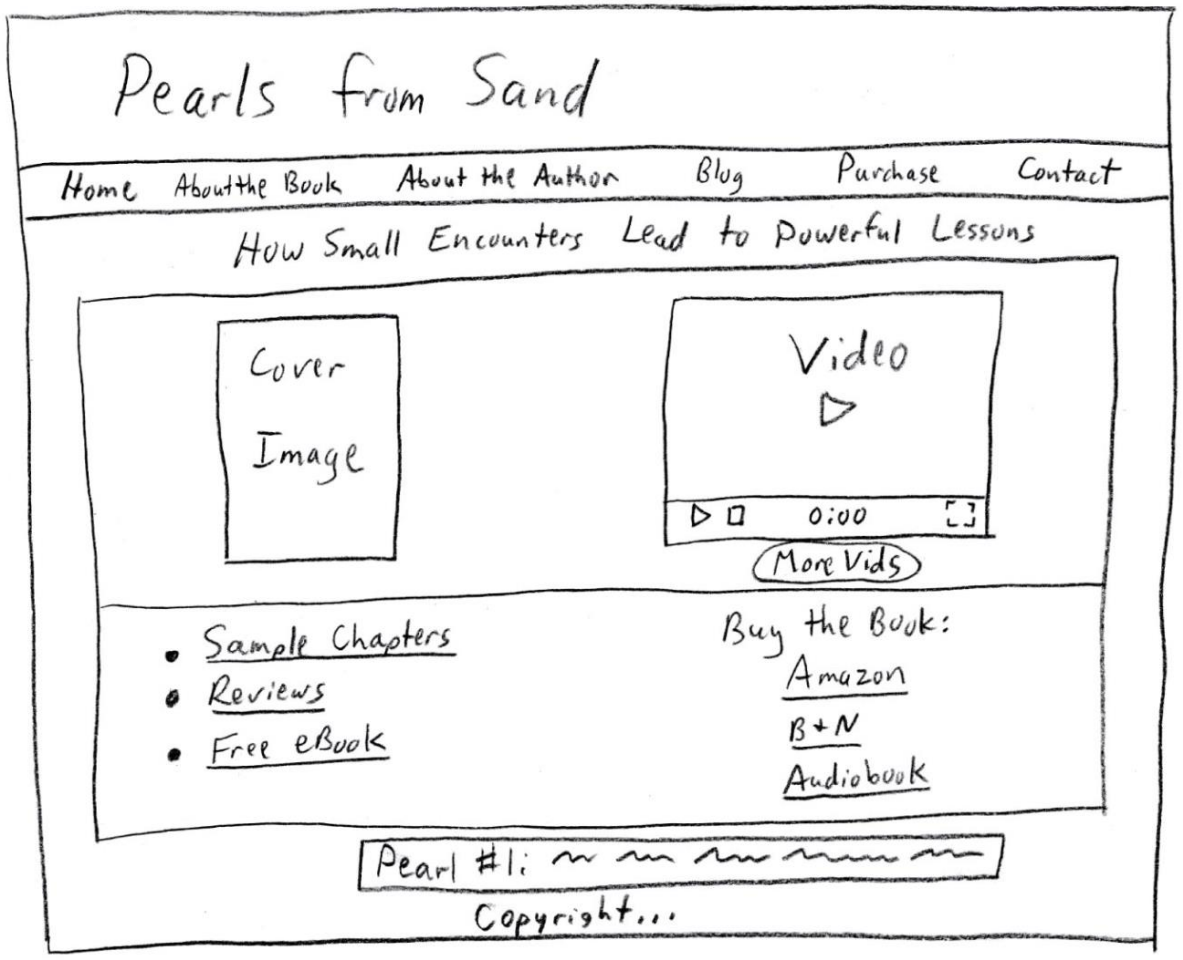
- Requirements describe what is observable at the environment-machine interface.
- Indicative mood describes the environment (as-is)
- Optative mood to describe the environment with the machine (to-be).

PROTOTYPES, MOCKUPS, STORIES

Mockups, Prototypes, Stories

- Humans: better at recognizing whether a solution is correct than solving the problem from a blank page.
- Mock-ups/prototypes help explore uncertainty in the requirements.
  - Validate that we have the right requirements.
  - Elicit requirements at the "borders" of the system.
  - Assert feasibility of solution space.
  - Get feedback on a candidate solution.
- "**I'll know it when I see it**"

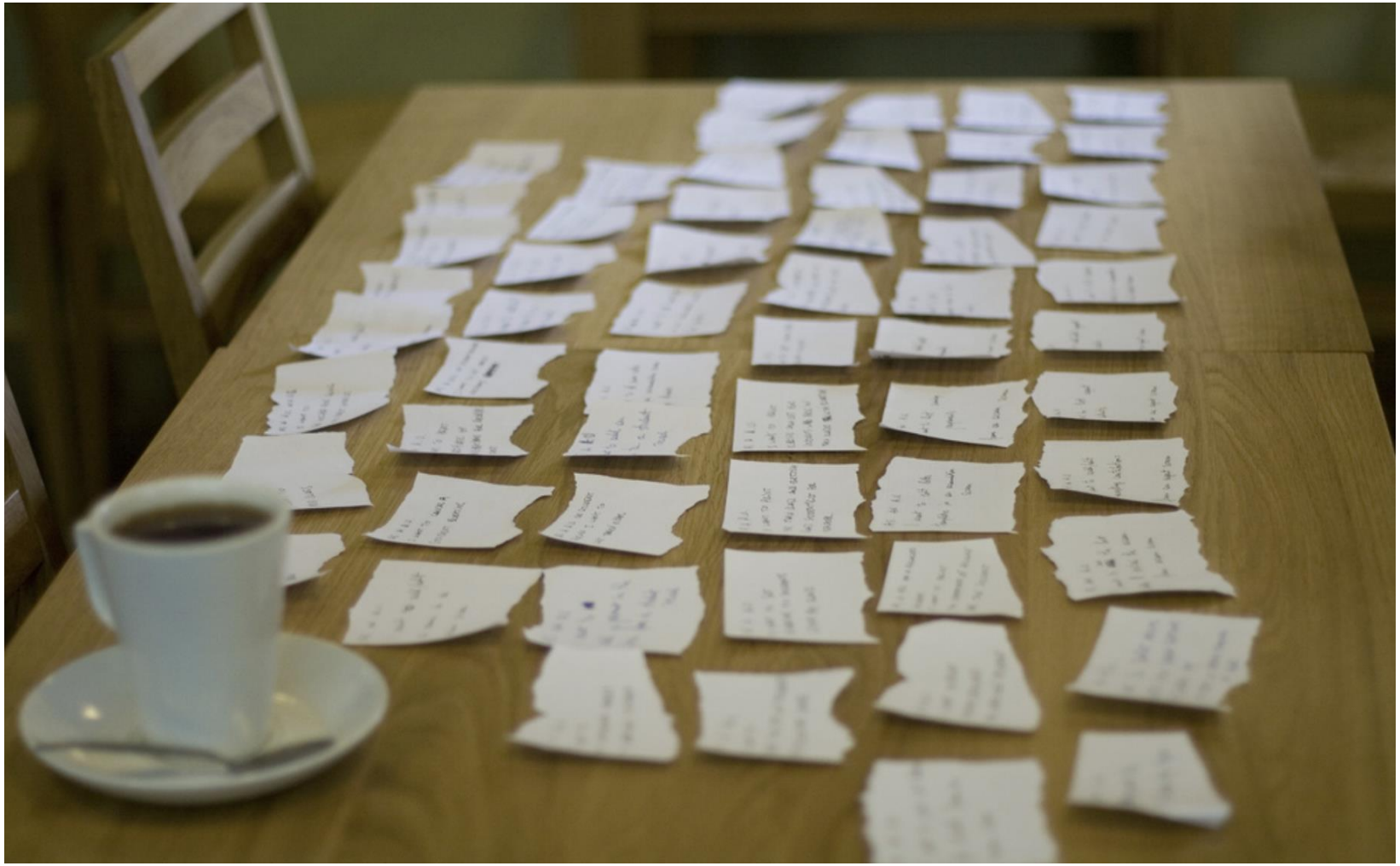
UI "Sketch" Suitable for Inclusion in SRS



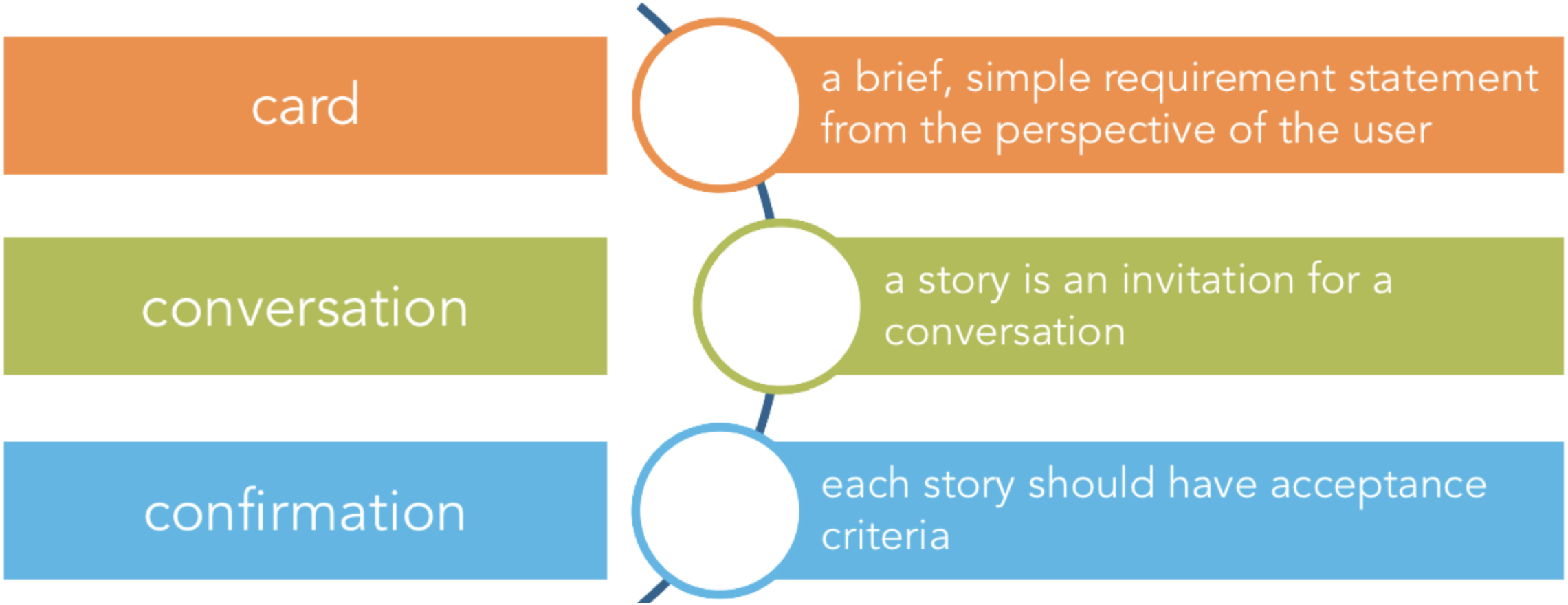
Story

- **Who** the players are
- **What** happens to them
- **How** it happens through specific episode
- **Why** this happens
- **What if** such and such an event occurs
- **What could go wrong** as a consequence

User Stories

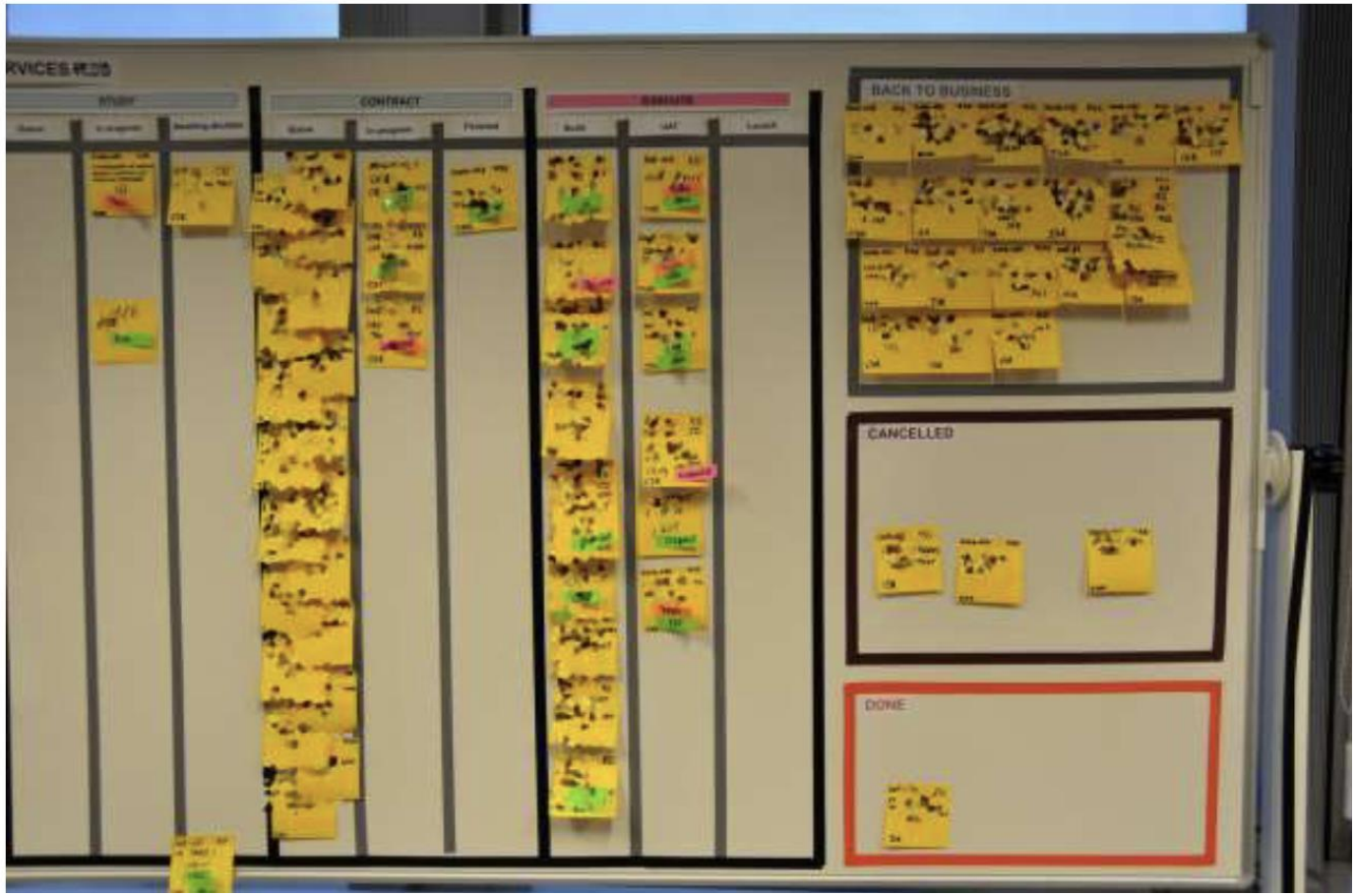


User Stories



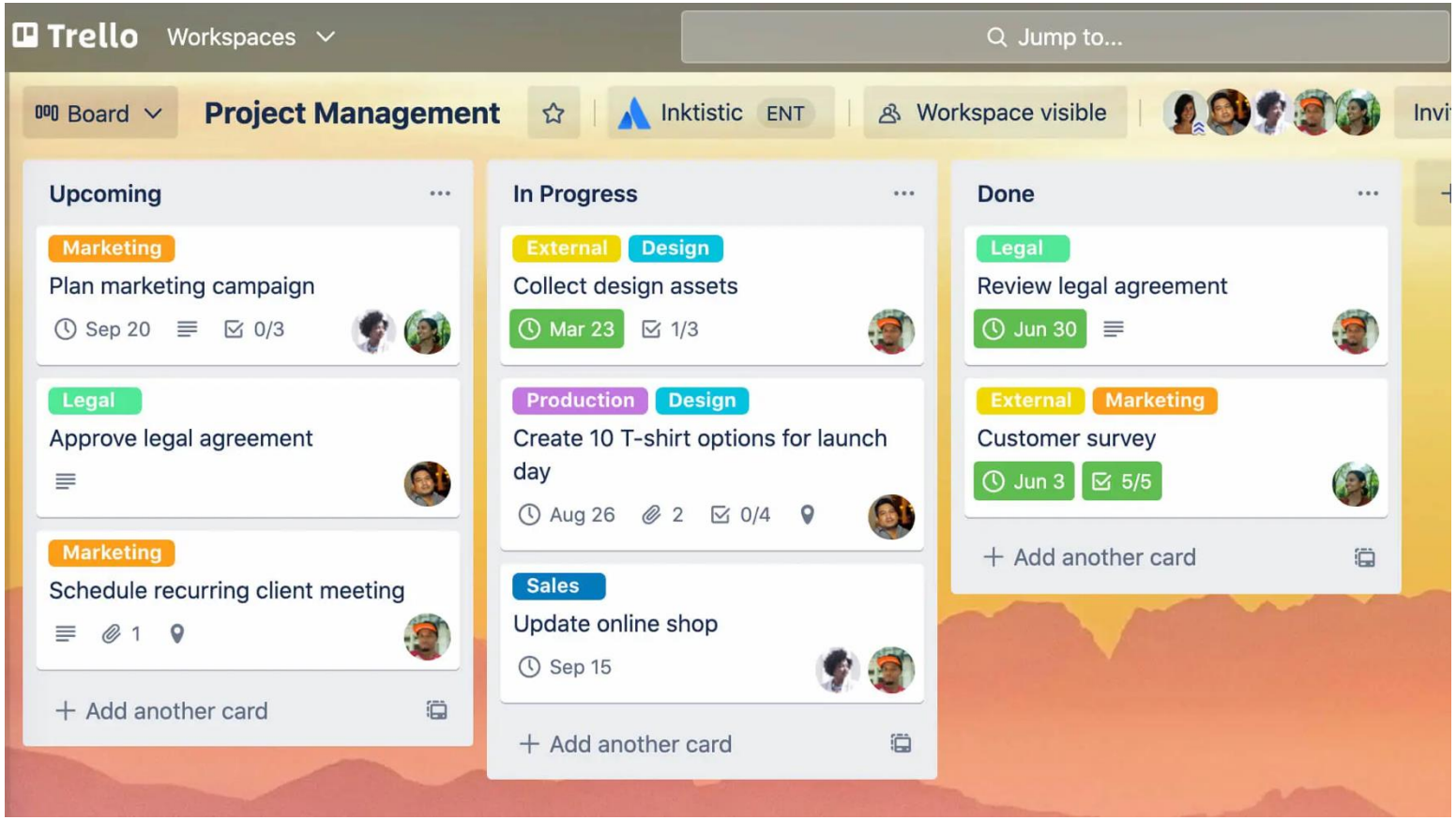
Use of User Stories

Keep a board of user stories, group them into “epics”



Trello Boards

We'll use this for creating and management of stories/tasks in CSDS 393/493



https://trello.com/

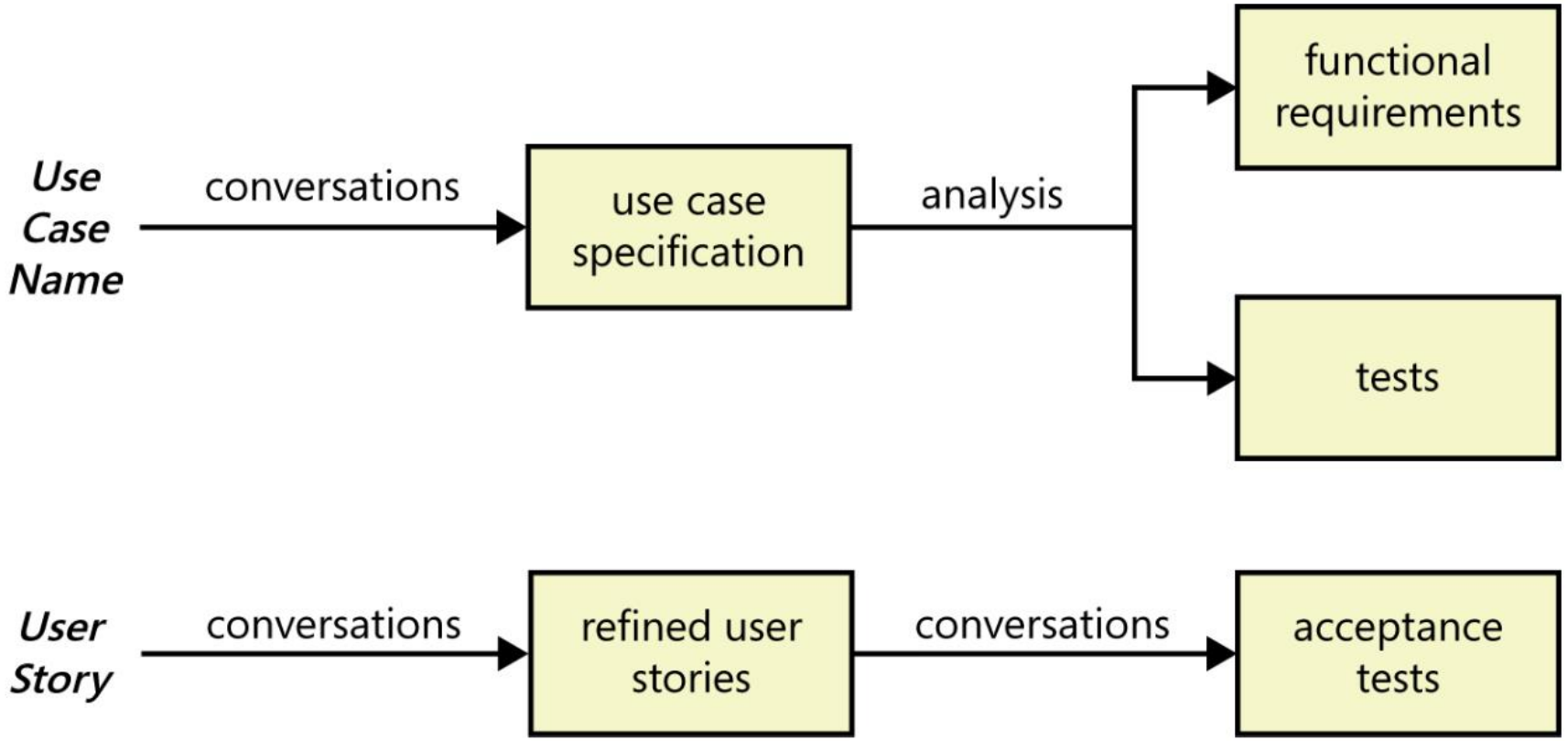
Scenarios

- Storyboards illustrate scenarios: a typical sequence of interaction among system components that meets an implicit objective.
  - Storyboards explicitly cover **at least who, what, and how**.
- Different types:
  - Positive vs negative (should and should not happen)
  - Normal vs abnormal
- Pluses: Concrete, support narrative description
- Minuses: inherently partial

Use Cases

- We talk about many types, at different granularities:
  - Full use case model (whole-system, higher-level)
  - "Agile" use case: small, concrete pieces of system functionality to be implemented (sometimes conflated with "user stories")
- Used at multiple stages:
  - Requirements elicitation (illustrated, validate, requirements; highlight conflicts, prioritize requirements, etc.)
  - Requirements documentation
  - Concrete design: UML diagrams

Use Cases vs. User Stories

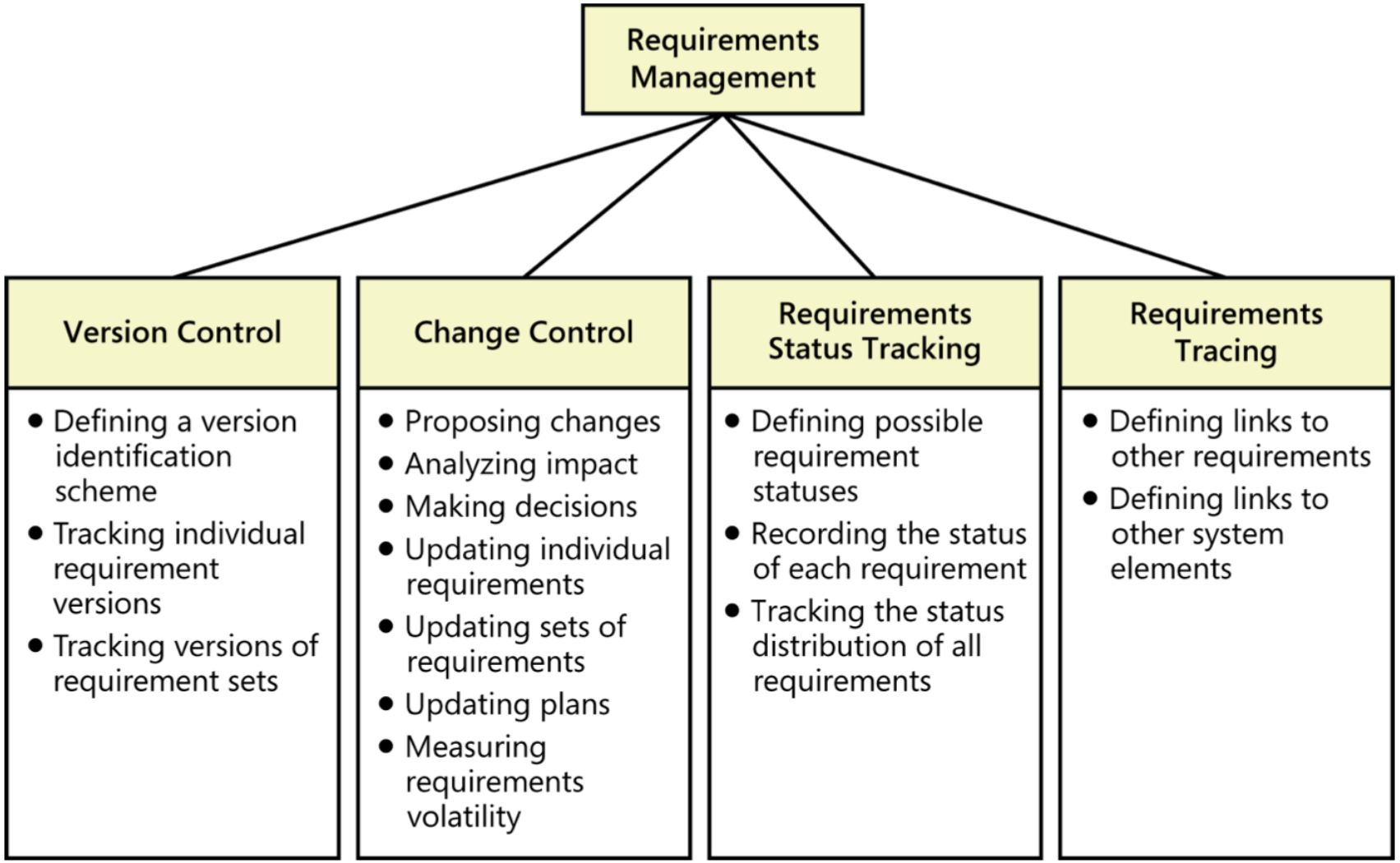


Requirements Management

Requirements Management

- Includes activities that maintain integrity, accuracy, and currency of requirements throughout project
- SRS should be updated whenever requirements change
- Facilitated by traceability between SRS, design, code

Requirements Management Process

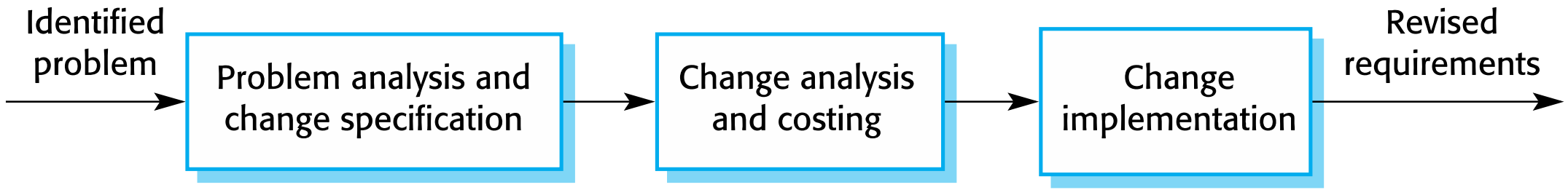


Requirements Management Activities

- Define a change-control process
- Establish a change control “board”
  - Reviews and approves/rejects formal change requests
- Perform change impact analysis
- Establish a baseline and control the versions
- Maintain a history of changes
- Track the status of each requirement
- Measure volatility
- Create a traceability matrix

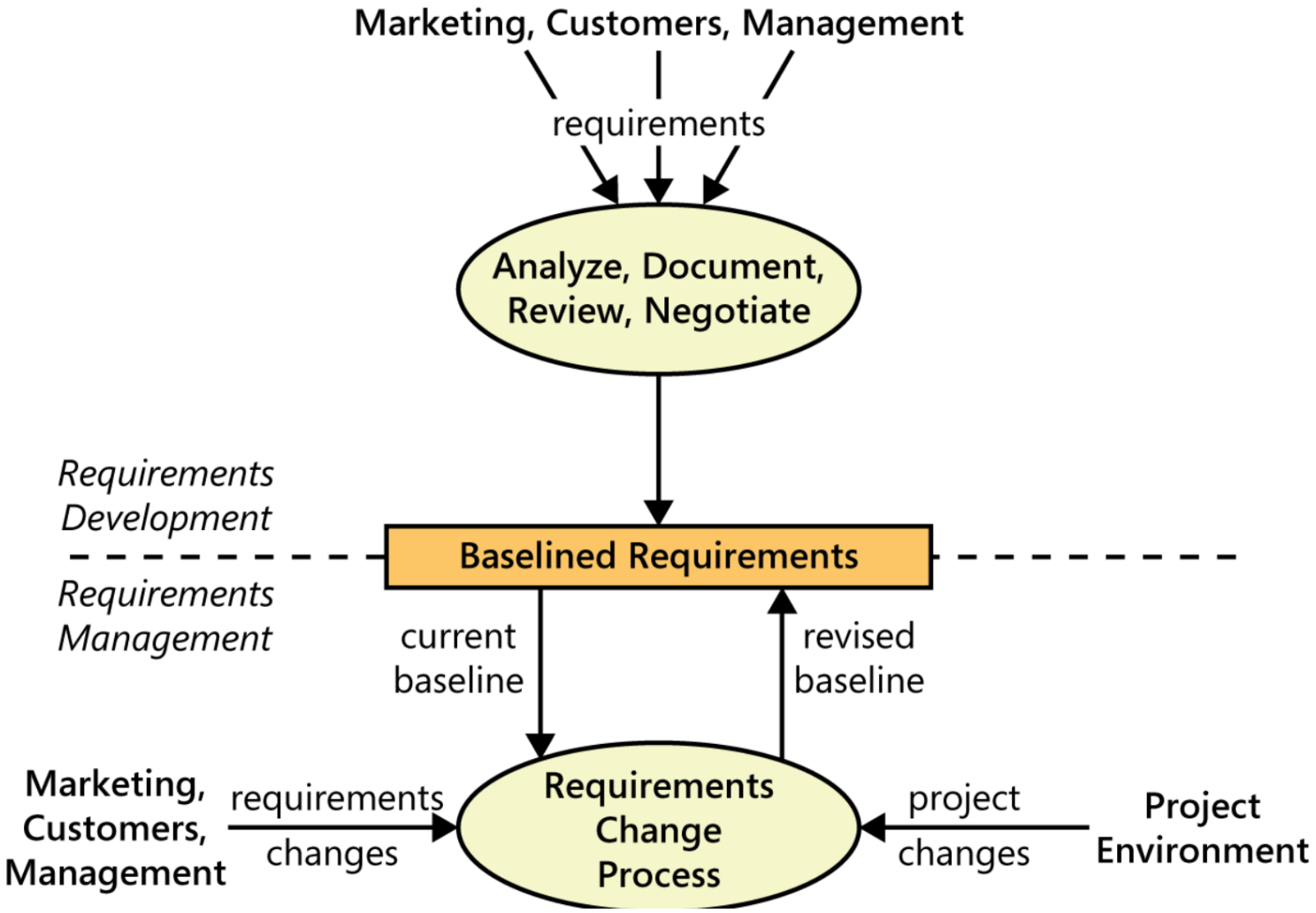
Requirements Evolution

New requirements emerge as a system is being developed and after it has gone into use.



Requirements change management

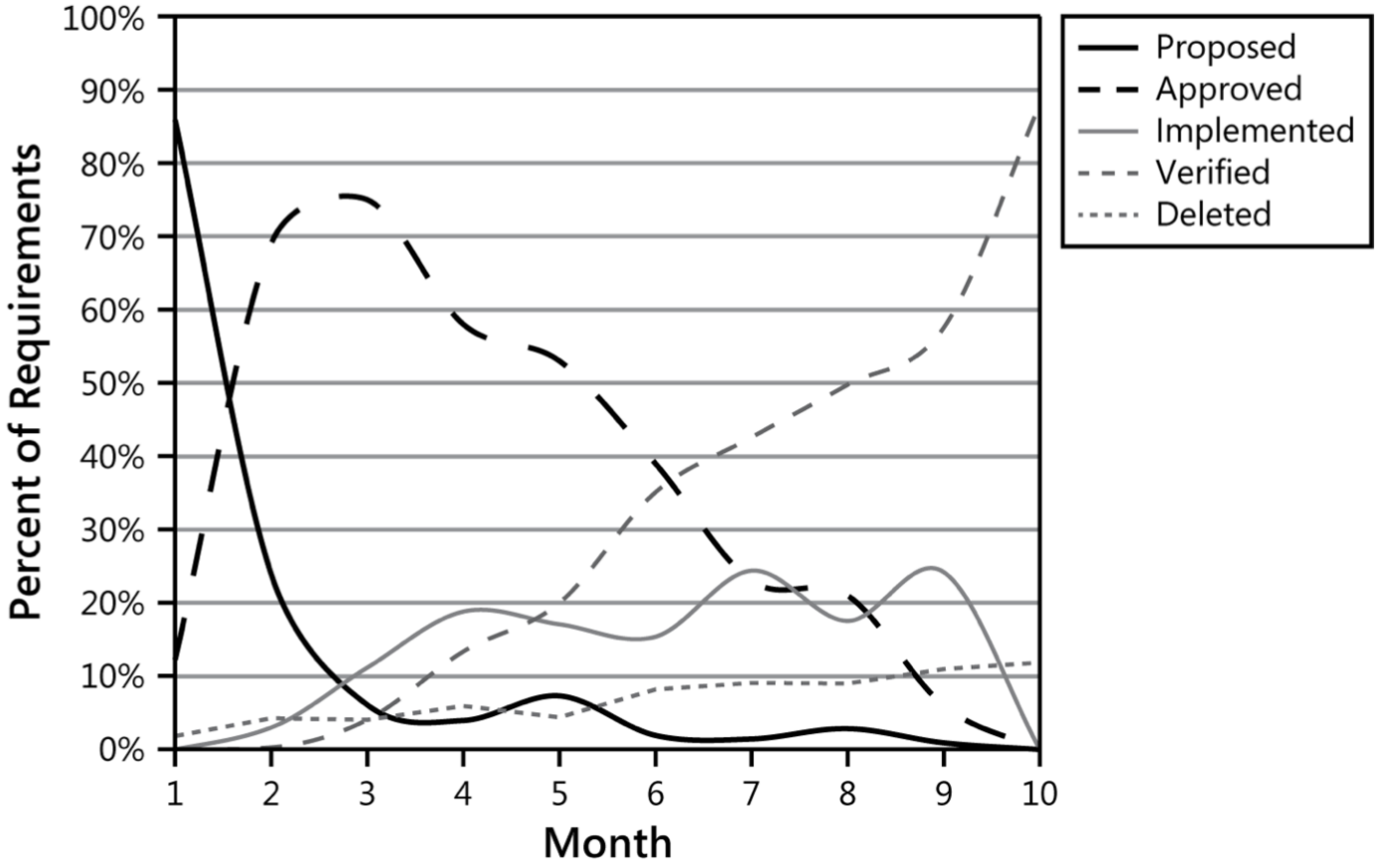
Requirements Development and Management



Requirements Status Tracking

Status	Definition
Proposed	The requirement has been requested by an authorized source.
In Progress	A business analyst is actively working on crafting the requirement.
Drafted	The initial version of the requirement has been written.
Approved	The requirement has been analyzed, its impact on the project has been estimated, and it has been allocated to the baseline for a specific release. The key stakeholders have agreed to incorporate the requirement, and the software development group has committed to implement it.
Implemented	The code that implements the requirement has been designed, written, and unit tested. The requirement has been traced to the pertinent design and code elements. The software that implemented the requirement is now ready for testing, review, or other verification.
Verified	The requirement has satisfied its acceptance criteria, meaning that the correct functioning of the implemented requirement has been confirmed. The requirement has been traced to pertinent tests. It is now considered complete.
Deferred	An approved requirement is now planned for implementation in a later release.
Deleted	An approved requirement has been removed from the baseline. Include an explanation of why and by whom the decision was made to delete it.
Rejected	The requirement was proposed but was never approved and is not planned for implementation in any upcoming release. Include an explanation of why and by whom the decision was made to reject it.

Requirements Status Tracking



Traceability

- Traceability between requirements and corresponding
  - design elements,
  - code elements, and
  - test cases assists developers in understanding their relationships.
- It also facilitates debugging.

Example Traceability Matrix

Functional Requirement	Implemented in	Tested in
CAT.FR1	../src/cat.py:23	test_plan.sgm:429
CAT.FR2	../src/cat.py:33	test_plan.sgm:438
CAT.FR3	../src/cat.py:53	test_plan.sgm:448
CGI.CHANGE-HTML-TEMPLATES.FR1	../src/html_substitutor.py:20	test_plan.sgm:753
CGI.CHANGE-HTML-TEMPLATES.FR2	../src/html_substitutor.py:45	test_plan.sgm:753
CGI.CHANGE-HTML-TEMPLATES.FR3	../src/html_substitutor.py:48	test_plan.sgm:754
CGI.CHANGE-HTML-TEMPLATES.FR4	../src/html_substitutor.py:54	test_plan.sgm:754
CGI.EDIT.FR1	../src/cgi_switchboard.py:208	test_plan.sgm:998
CGI.EDIT.FR2	../src/cgi_edit_form.py:27	test_plan.sgm:1012
CGI.EDIT.FR3	../src/cgi_edit_form.py:30 ../src/cgi_edit_form.py:34	test_plan.sgm:1005

Note: this example does not exhibit backward traceability to evidence for requirement

https://yaktrack.sourceforge.net/yaktrack_docs/a2332.html

Example Traceability Matrix

Requirements Traceability Matrix

Project Name	<Project Name here>	Created On	3-Oct-11	Reviewed On	4-Oct-11
Release No	<Project Release>	Created By	<Creator's Name>	Reviewed By	<Reviewer's Name>
Version	<Doc version>				

ID	Requirement ID	Requirement Description	Status	Design Document	Code Module	TestCase ID	Test Case Name	User Manual	Tested On/ Verification
001	UC 1.0	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-001	CM-001	TC-001	ProjName_UCID_TestCase Name	Section 4.5	Pending
002	UC 1.1	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-002	CM-002	TC-002 TC-003	N.A	Section 4.6	Verified
003	UC 1.2	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-003	CM-003	TC-004 TC-006 TC-007 TC-008		Section 5.7	Verified
004	UC 1.3	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-004	CM-004			Section 6.8	In-progress
005	UC 1.4	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-005	CM-005			Section 7.9	Not Verified
006	UC 1.5	Testing Requirement Description here. It should not be more than 2-3 lines	TBD	DM-006	CM-006			Section 4.10	Verified
007	UC 1.6	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.11	Not Verified
008	UC 1.7	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.12	Pending
009	UC 1.8	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.13	Not Verified
010	UC 1.9	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.14	Pending

https://staragile.com/blog/requirement-traceability-matrix

Requirements Prioritization

- Nice to have
- Cost, time, and other limits
- Dependencies among requirements
- Strategies to base on value contribution

Requirements Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Checking
  - Validity
  - Consistency
  - Completeness
  - Realism
  - Verifiability

Requirements Validation Techniques

- Requirements reviews
- Prototyping
- Test-case generation

Graphical Models for Requirements

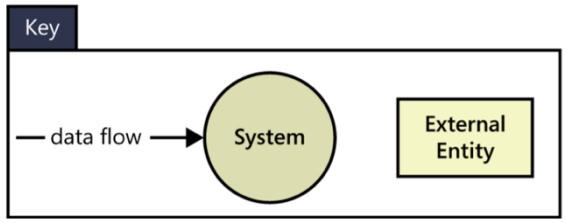
- Used in requirements analysis & specification to augment text, e.g.,
  - Data flow diagrams
  - Process flow diagrams
  - Object diagrams
  - State transition diagrams
  - Decision tables and trees
  - Feature trees
  - Use-case diagrams
  - Activity diagrams
  - Entity-relationship diagrams

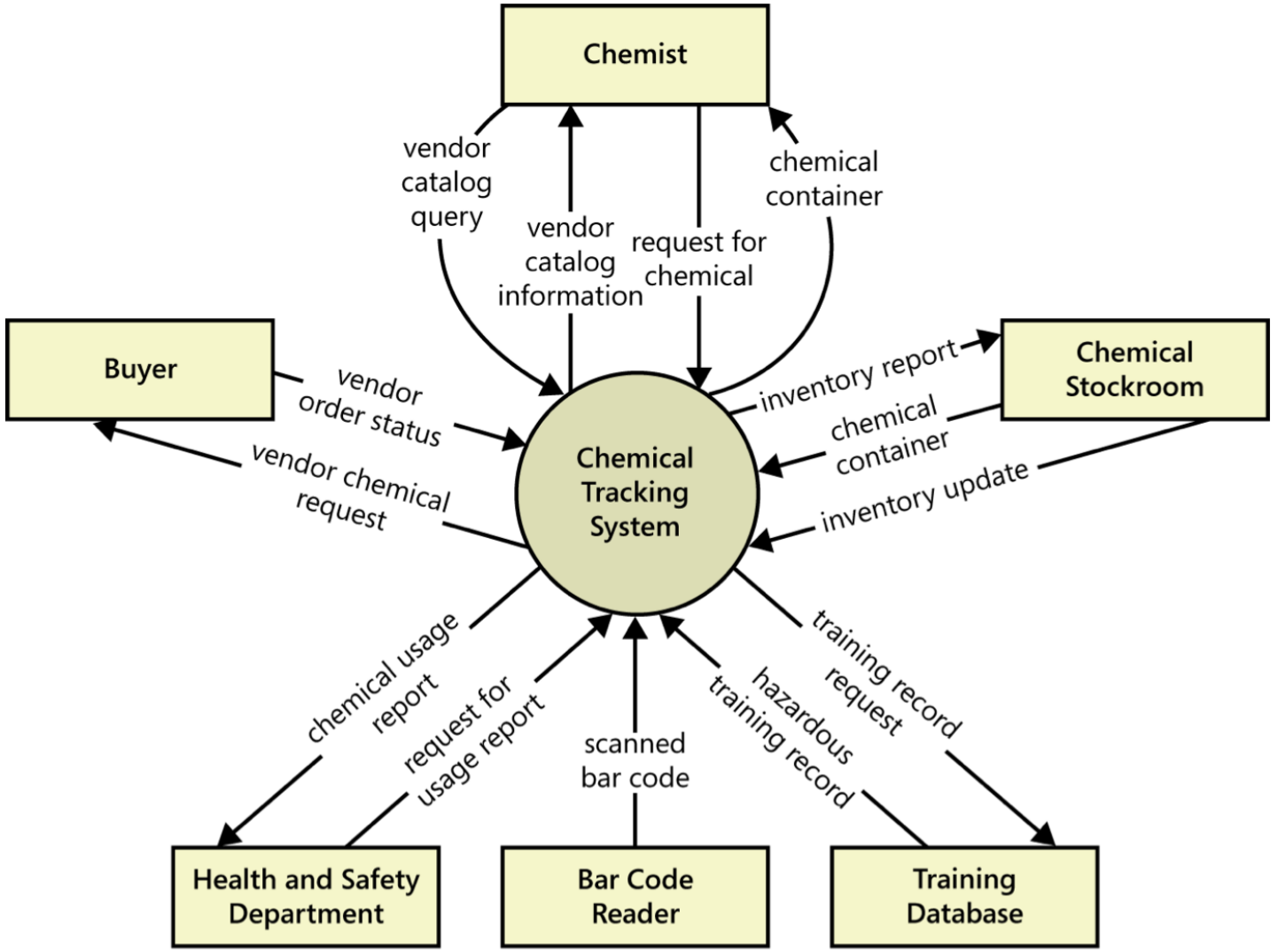
Graphical Models

- Uses of graphical models:
  - Describing existing or new business, process, system, or interface

- Should not be used in SRS to describe detailed design

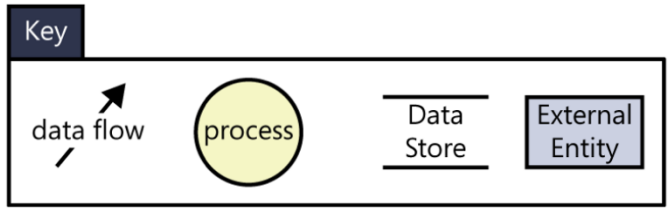
Example: Context Diagram

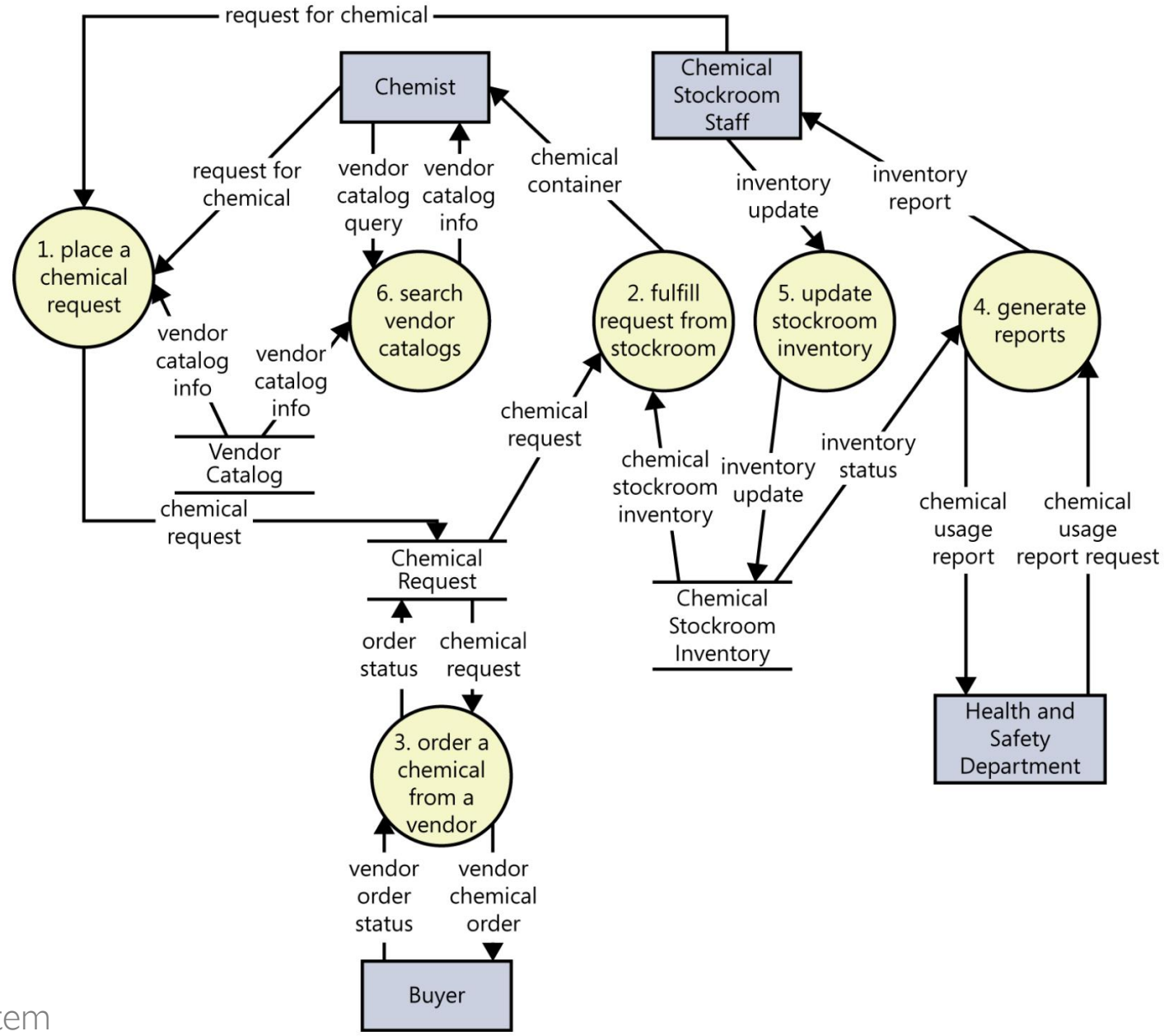




Context diagram for the Chemical Tracking System

Example: Data Flow Diagram

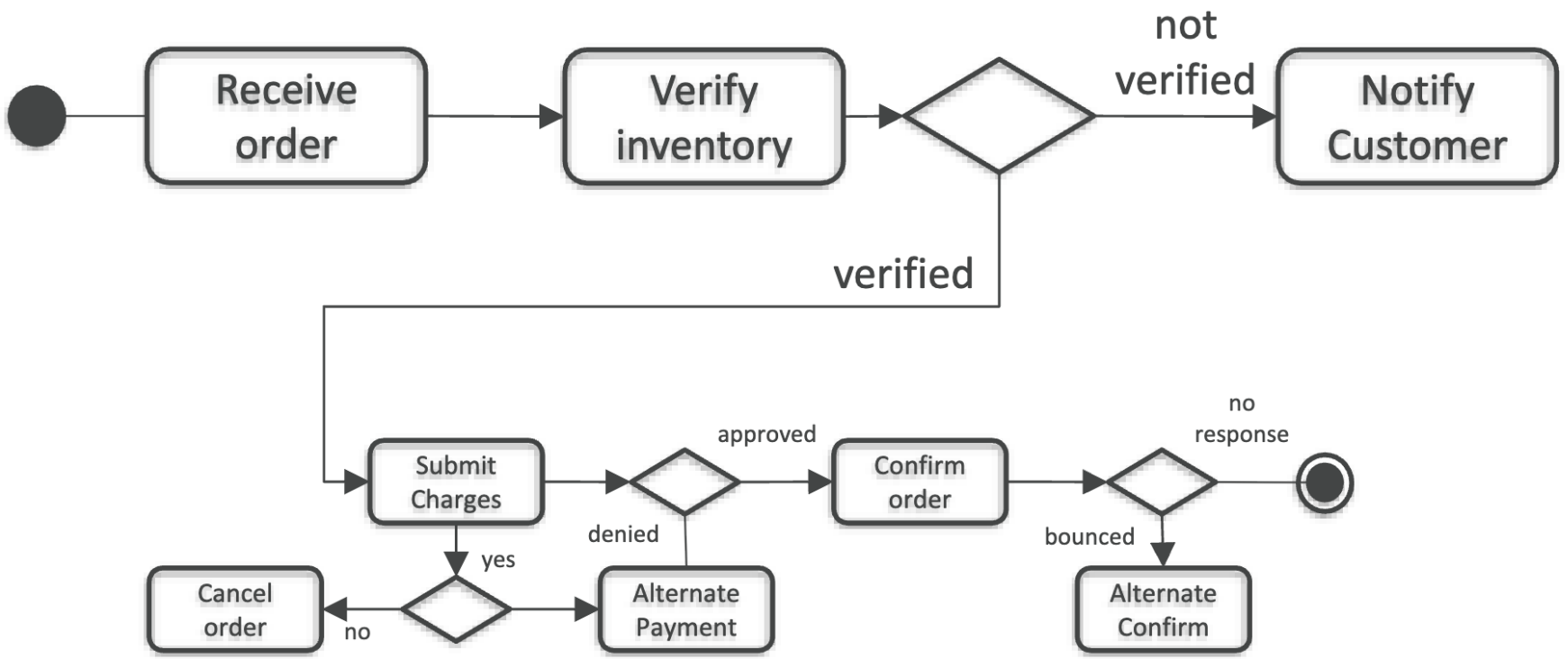




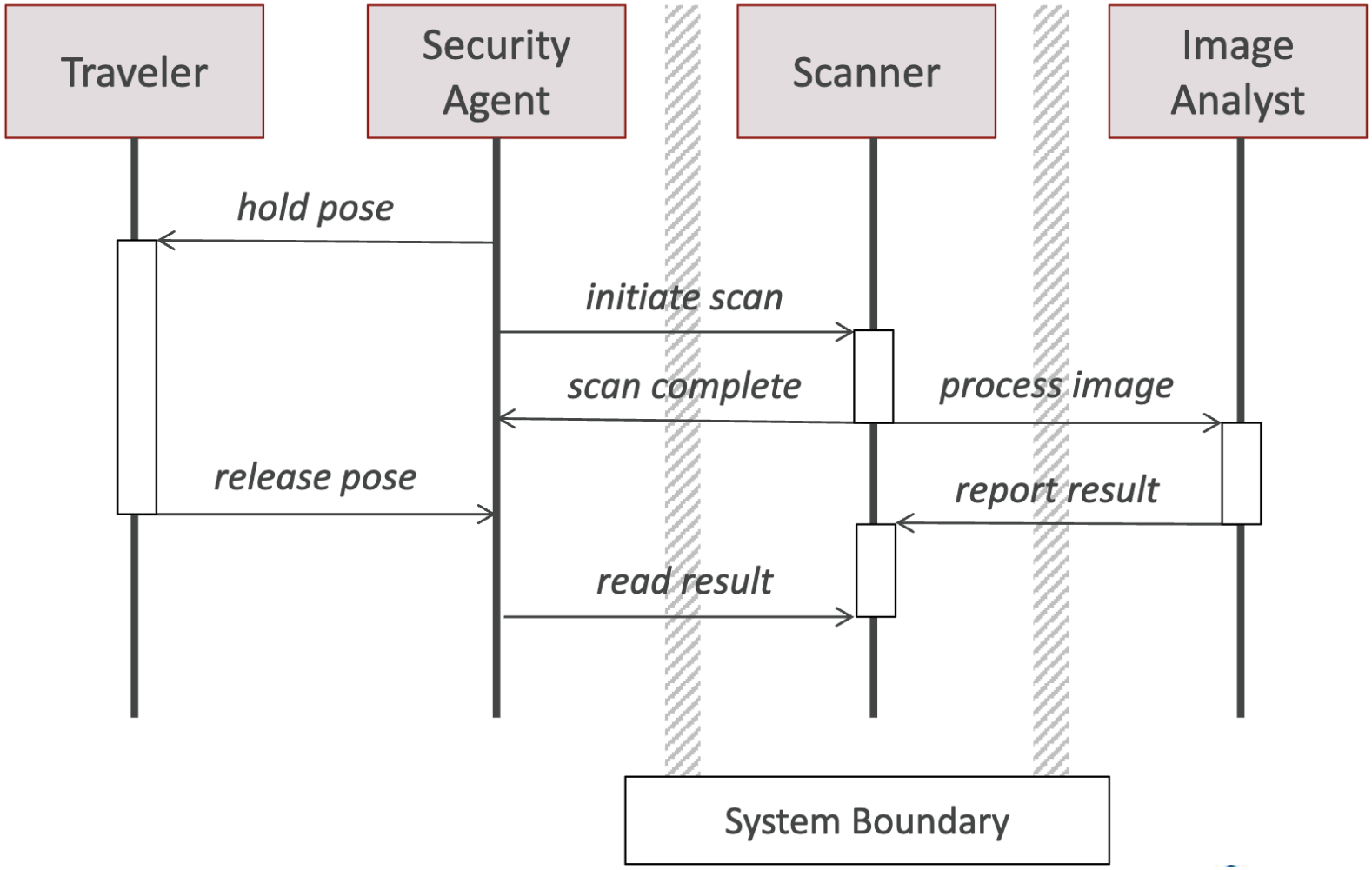
Data flow diagram for the Chemical Tracking System

Activity Diagrams

Activity diagrams (or flow charts) represent the logic in a graph notation



Sequence Diagramming



Formal Specifications

- Logical expressions of shared actions at the interface of the machine
- Includes
  - linking domain properties
  - agent actions as **pre- and post-conditions**

$$\forall\, s\, \forall\, c(\text{enrolled}(s, c) \Rightarrow \text{student}(s) \wedge \text{course}(c))$$

Requirements "Bill of Rights" for Customers

1. Expect analysts to speak your language.
2. Expect analysts to learn about your business and your objectives.
3. Expect analysts to structure the information you provide during requirements elicitation into a written SRS.
4. Have analysts explain all work products of the requirements process.
5. Expect analysts and developers to treat you with respect and to maintain a collaborative and professional attitude throughout your interaction.

Customers Rights (2)

6. Have analysts and developers provide ideas and alternatives for your requirements
7. Describe characteristics of the product that will make it easy and enjoyable to use.
8. Be given opportunities to adjust your requirements to permit reuse of existing software components.
9. Receive good faith estimates of the costs, impacts, and trade-offs when you request a change in the requirements.
10. Receive a system that meets your functional and quality needs.

Customer "Bill of Responsibilities"

1. Educate analysts and developers about your business and define business jargon.
2. Spend the time needed to provide requirements, clarify them, and iteratively flesh them out.
3. Be specific and precise when providing input about the system's requirements.
4. Make timely decisions about requirements when requested to do so.
5. Respect a developer's assessment of the cost and feasibility of requirements.

Customer Responsibilities (2)

6. In collaboration with the developers, set priorities for functional requirements.
7. Carefully review requirements documents and evaluate prototypes.
8. Communicate changes to the requirements as soon as you know about them.
9. Follow the development organization's process for requesting requirements changes.
10. Respect the process the analysts use for requirements engineering.

Risks

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”

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

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, ...)

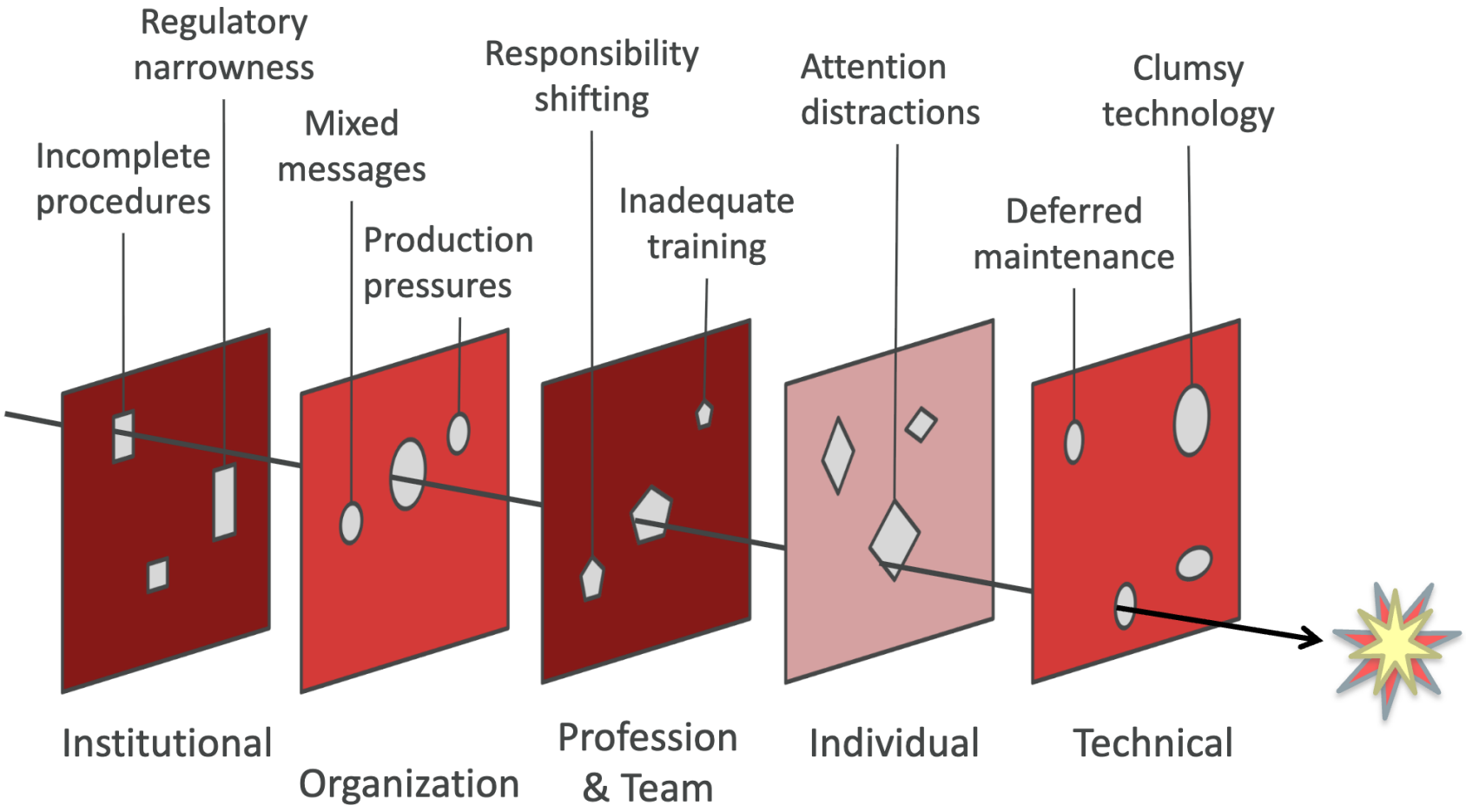
BaseScore = *round_to_1_decimal(((0.6*Impact)+(0.4*Exploitability) 1.5)*f(Impact))*

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

Exploitability = *20* AccessVector*AccessComplexity*Authentication*

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

The Swiss Cheese Model



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

Risk Assessment Matrix

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

DEPARTMENT OF DEFENSE STANDARD PRACTICE