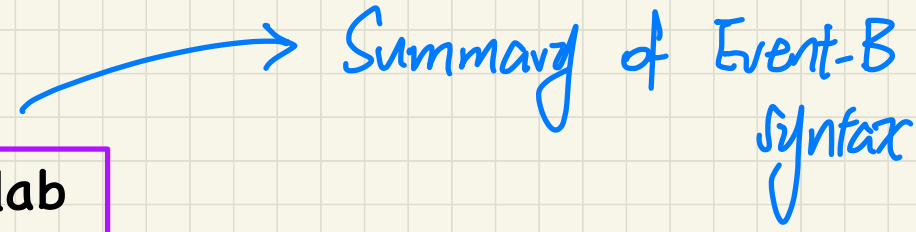
Lecture 2 - Sep. 10

Introduction

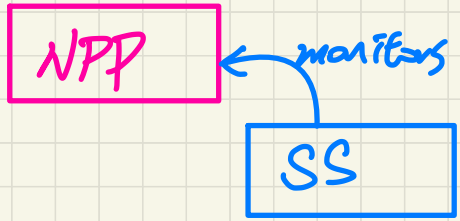
Safety-Critical vs. Missional-Critical
Professional Engineers: Code of Ethics
Safety Property/Invariant
Verification vs. Validation

Announcements/Reminders

- Priority: Lab1
- Wednesday's lab



Safety-Critical System



1. nuclear power plant
   + nuclear shutdown system

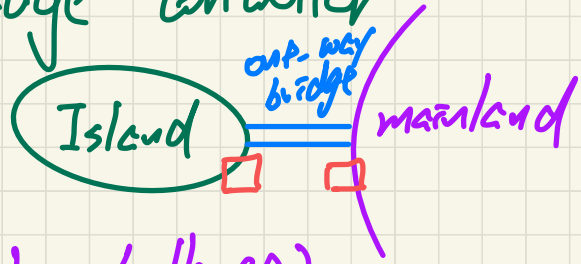
2. radiation

3. "glove"

4. pacemaker (pacemaker challenge)

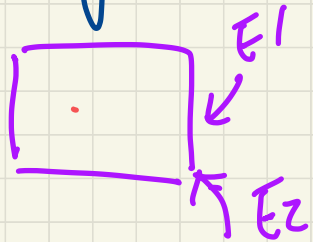
5. auto-pilot & auto-driving.

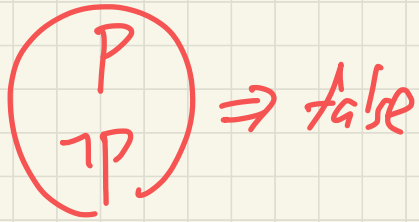
6. bridge controller



Acceptance Criteria

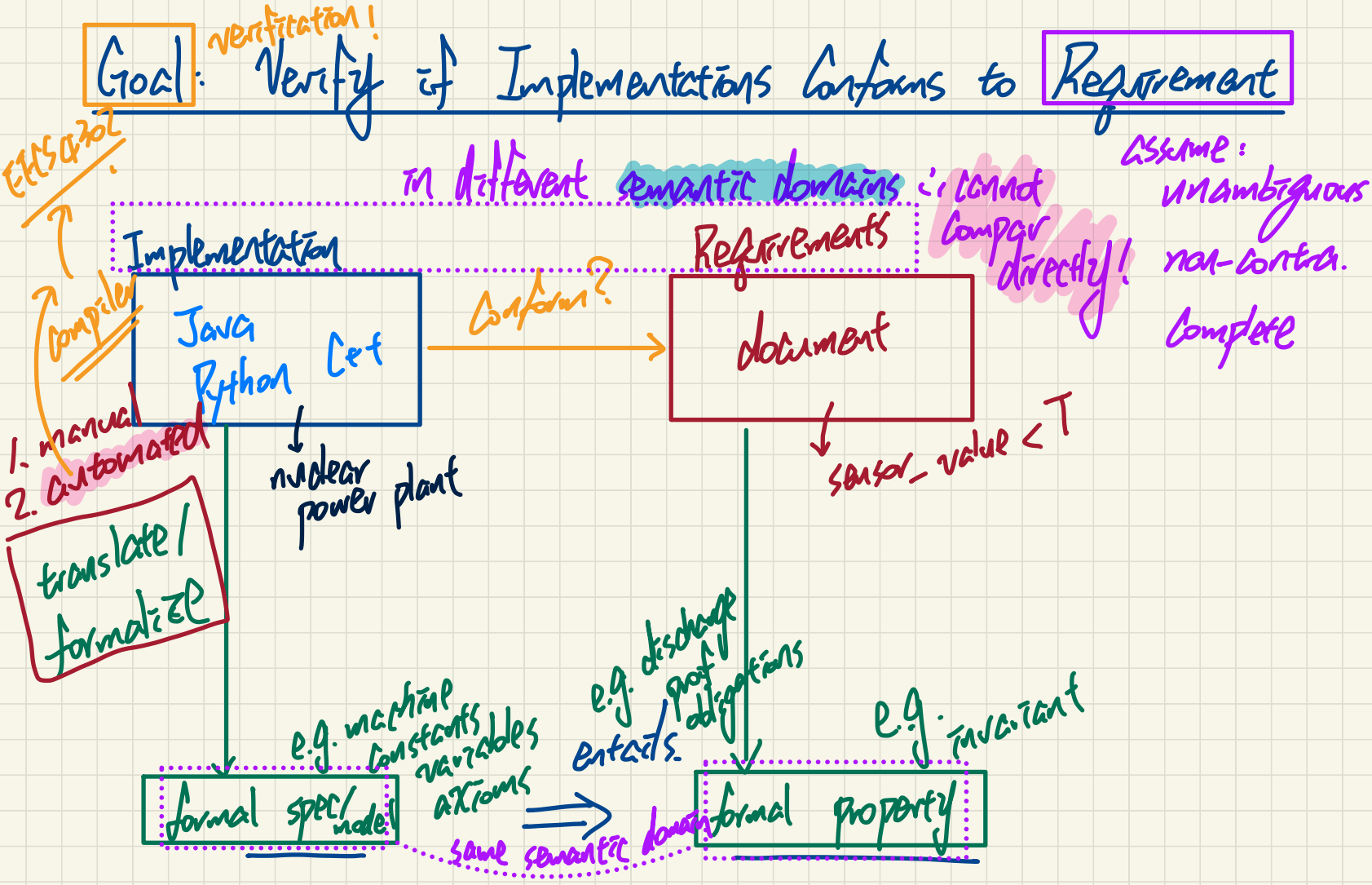
1. Req. precise
   - ↳ no ambiguities, no contradiction





   - ↳ complete
     - ↳ no missing scenarios

Labs


Goal: Verify if Implementation conforms to Requirement
verification!
Eclipse
compiler
in different semantic domains
cannot compare directly!
assume:
unambiguous
non-contra.
complete
Implementation
Java
Python
Cet
nuclear power plant
conform?
Requirements
document
sensor_value < T
1. manual
2. automated
translate / formalize
e.g. machine constants variables axioms
e.g. discharge proof obligations
entails
e.g. invariant
formal spec/model
formal property
same semantic domain

Mission-Critical vs. Safety-Critical

Safety critical

When defining safety critical it is beneficial to look at the definition of each word independently. Safety typically refers to being free from danger, injury, or loss. In the commercial and military industries this applies most directly to human life. Critical refers to a task that must be successfully completed to ensure that a larger, more complex operation succeeds. Failure to complete this task compromises the integrity of the entire operation. Therefore a safety-critical application for an RTOS implies that execution failure or faulty execution by the operating system could result in injury or loss of human life.

Safety-critical systems demand software that has been developed using a well-defined, mature software development process focused on producing quality software. For this very reason the DO-178B specification was created. DO-178B defines the guidelines for development of aviation software in the USA. Developed by the Radio Technical Commission for Aeronautics (RTCA), the DO-178B standard is a set of guidelines for the production of software for airborne systems. There are multiple criticality levels for this software (A, B, C, D, and E).

These levels correspond to the consequences of a software failure:

- Level A is catastrophic
- Level B is hazardous/severe
- Level C is major
- Level D is minor
- Level E is no effect

SCS

MCS

Safety-critical software is typically DO-178B level A or B. At these higher levels of software criticality the software objectives defined by DO-178B must be reviewed by an independent party and undergo more rigorous testing. Typical safety-critical applications include both military and commercial flight, and engine controls.

Mission critical

A mission refers to an operation or task that is assigned by a higher authority. Therefore a mission-critical application for an RTOS implies that a failure by the operating system will prevent a task or operation from being performed, possibly preventing successful completion of the operation as a whole.

Mission-critical systems must also be developed using well-defined, mature software development processes. Therefore they also are subjected to the rigors of DO-178B. However, unlike safety-critical applications, mission-critical software is typically DO-178B level C or D. Mission-critical systems only need to meet the lower criticality levels set forth by the DO-178B specification.

Generally mission-critical applications include navigation systems, avionics display systems, and mission command and control.

Source: http://pdf.cloud.opensystemsmedia.com/advancedtca-systems.com/SBS.Jan04.pdf

SCS vs. MCS

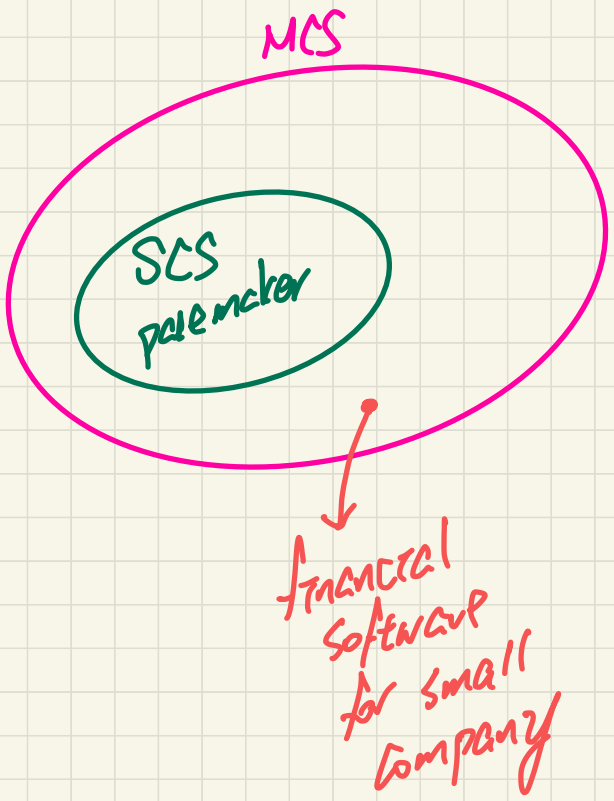
(1) ✗ SCS $\Leftrightarrow$ MCS

(2) ✓ SCS $\Rightarrow$ MCS

(3) ✗ MCS $\Rightarrow$ SCS

SCS $\subseteq$ MCS

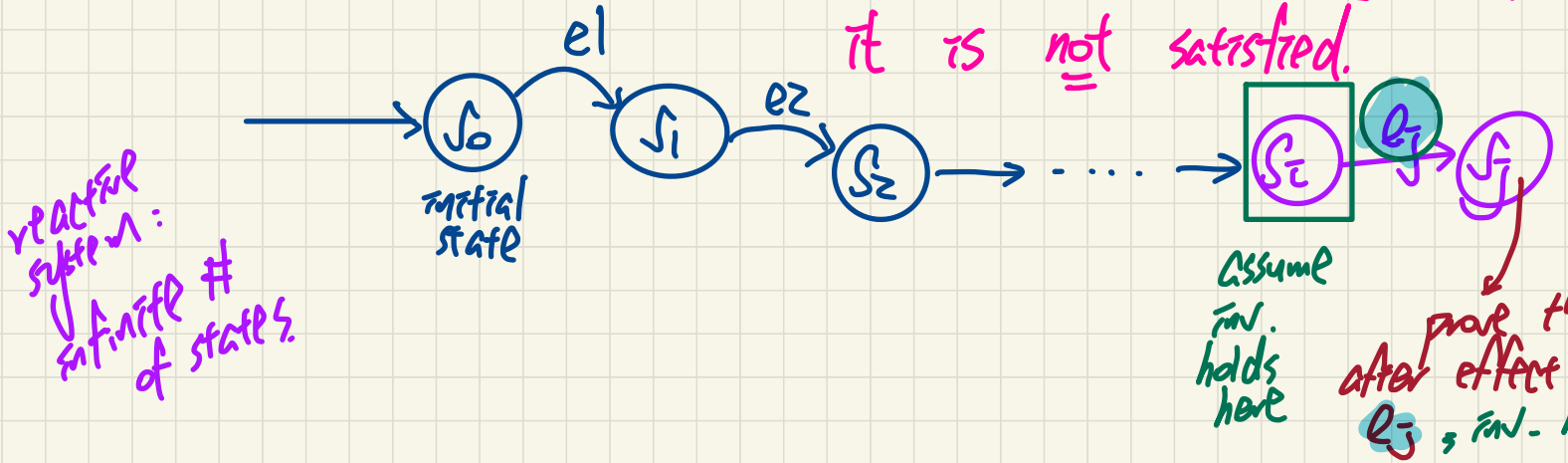
$\Rightarrow$



Safety Property / Invariant

↳ Every possible state of the system should satisfy it.

↳ If there's at least one state where the inv. does not hold, it is not satisfied.



assumption: req. already given

Verification: Are we building the product right?

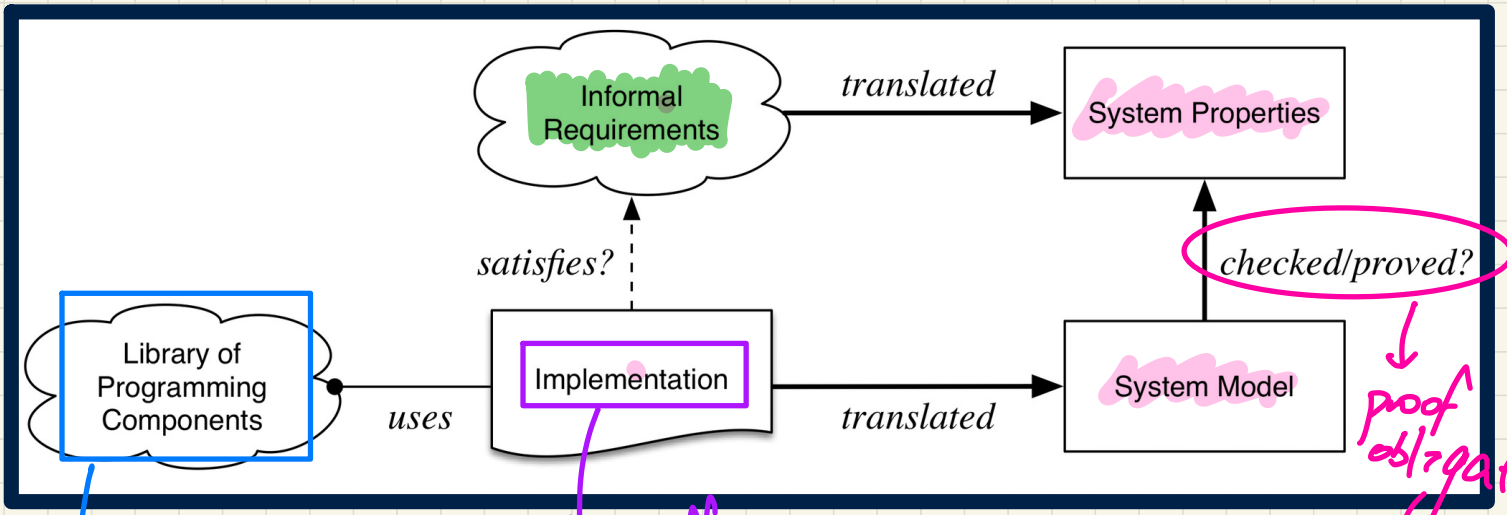
process of construction

4312.

Validation: Are we building the right product?

Are the req. given truly intended by customers?

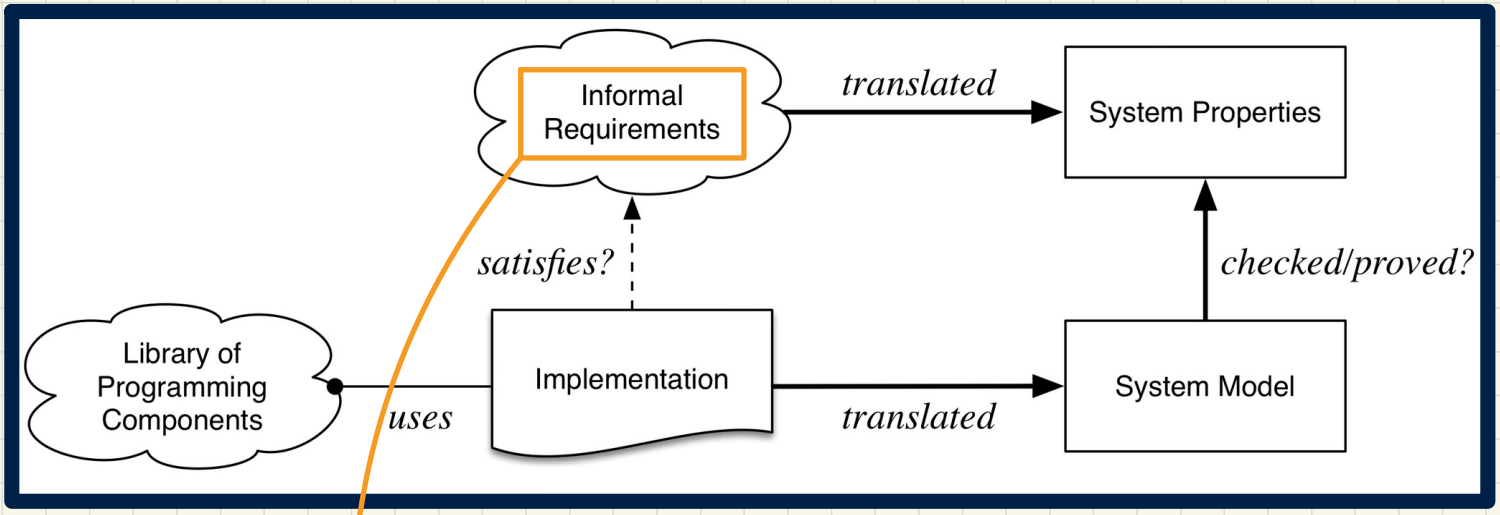
Building the product right?



Are the library classes reliable?
(additional verification needed e.g. testing)

1. In a diff. semantic domain
2. make use of library modules.

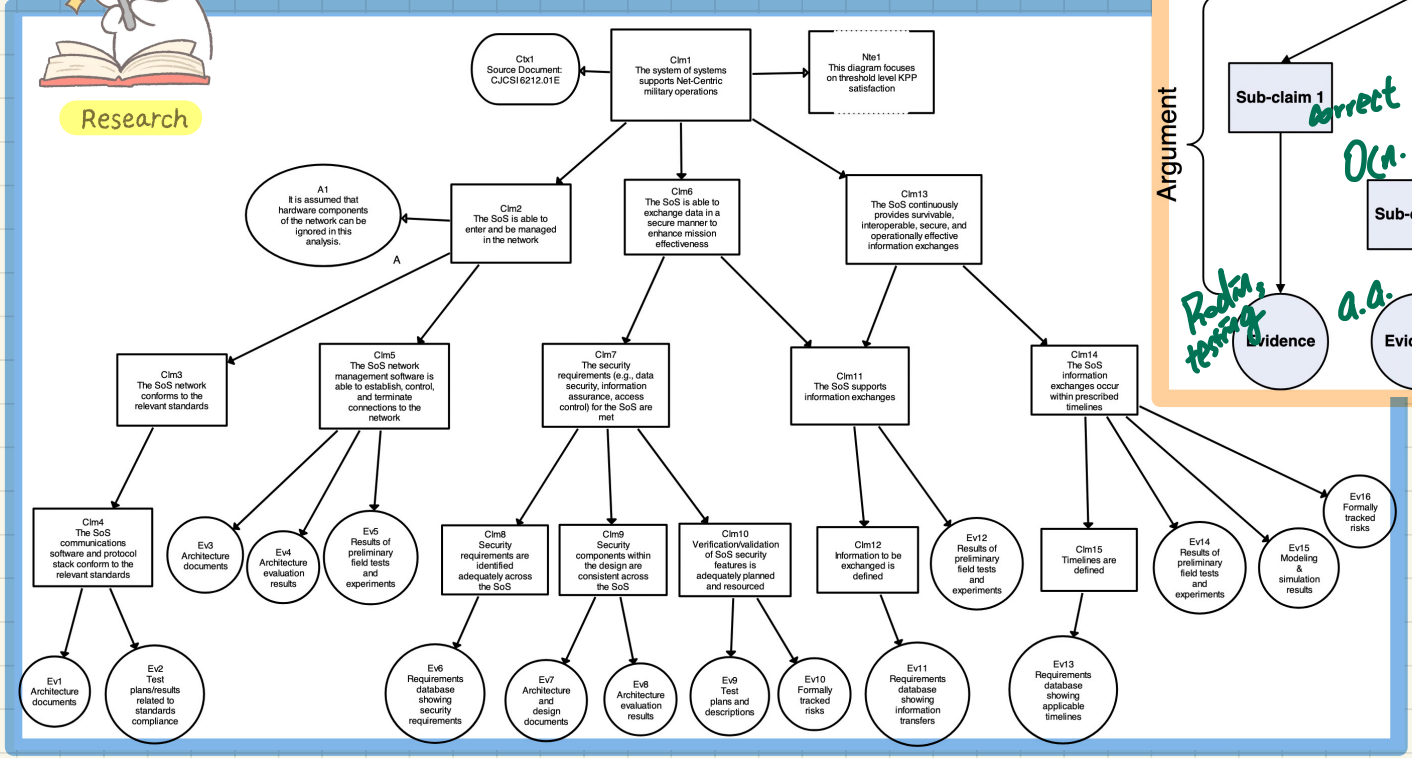
Building the right product?

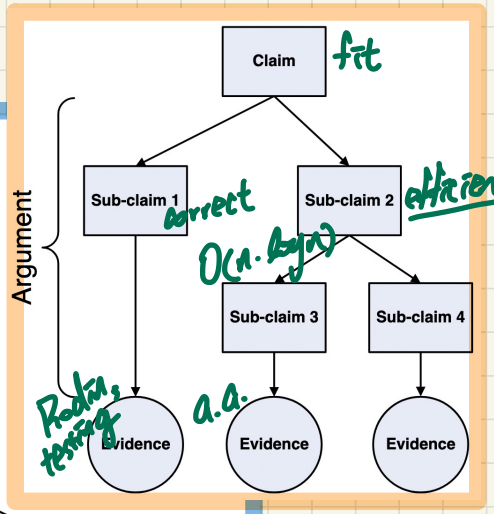


for validation, critical on what's given.

Certifying Systems: Assurance Cases

Research on "Assurance Cases" if interested!





Source: https://resources.sei.cmu.edu/asset_files/whitepaper/2009_019_001_29066.pdf