17-423/723: Designing Large-scale Software Systems

Arguing for & Reviewing Designs

Feb 26, 2024

Logistics

- M2 deadline extended to Friday
- Midterm on Wednesday
  - Covers up to the lectures last Wednesday (Design with reuse)
  - In class, open-book (but no ChatGPT or LLMs!)
  - Bonus points for submitting hand-written study notes with the exam

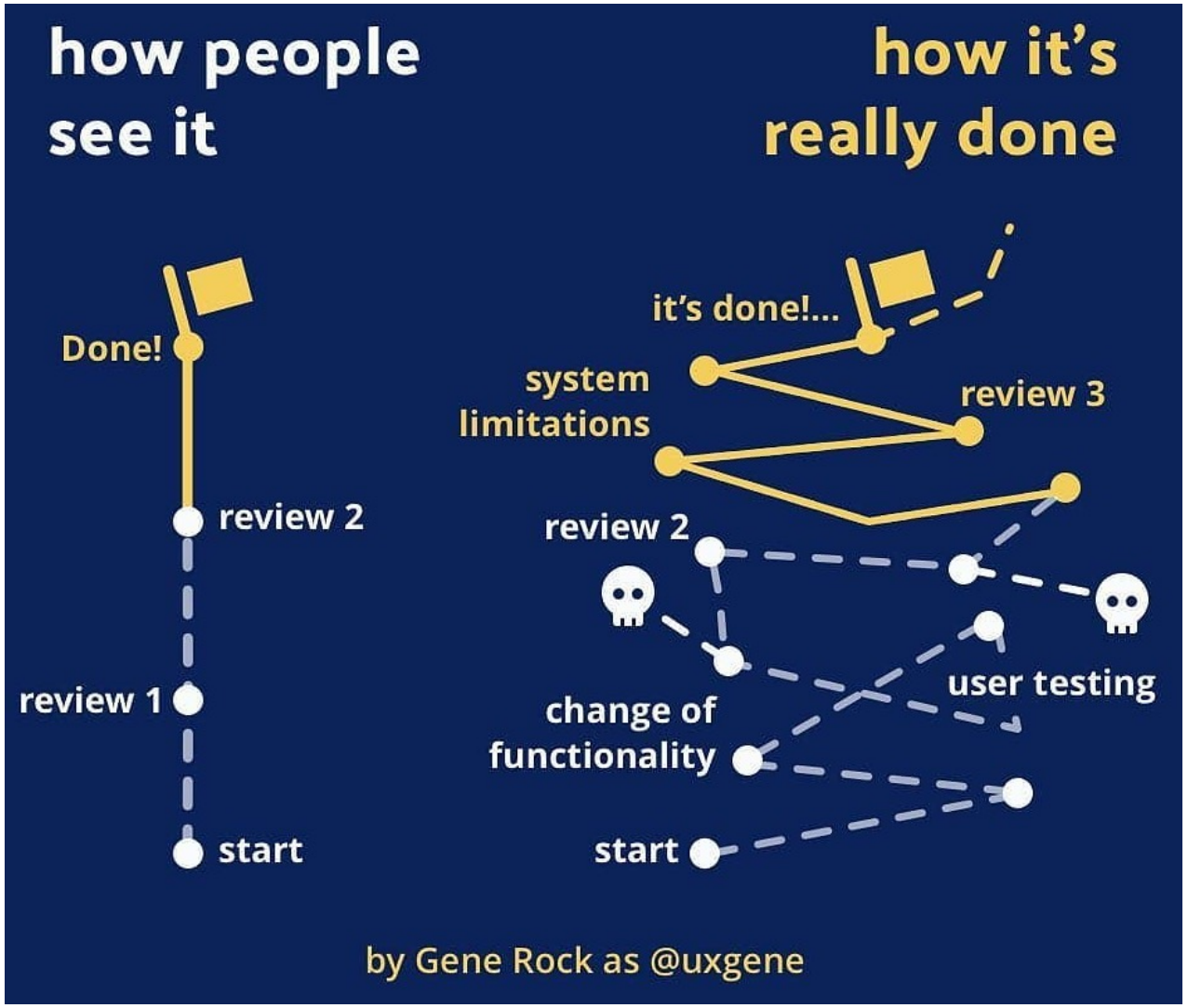
Leaning Goals

- Devise and document an argument for why the design achieves a desired function or quality attribute
- Review and identify weaknesses in an existing design argument

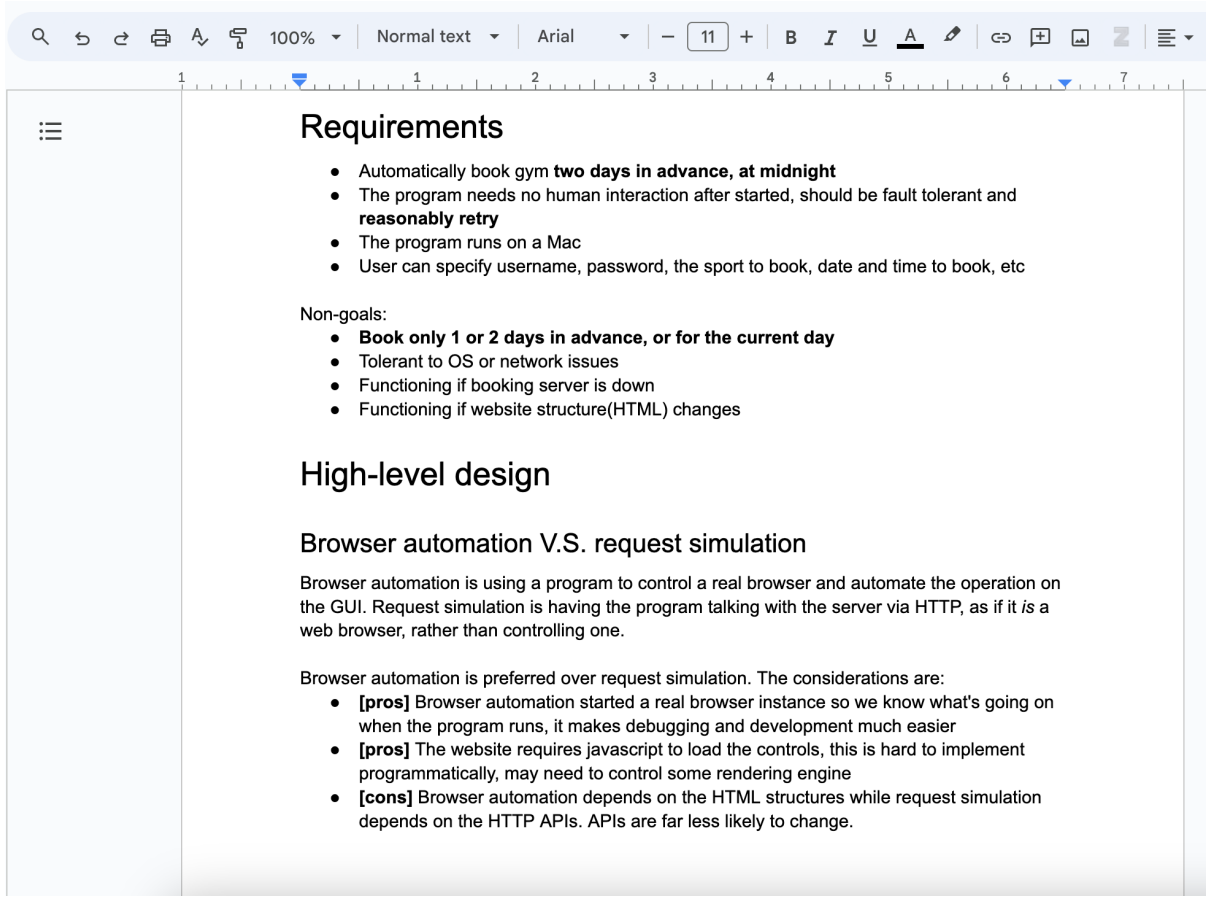
Design Review

- An activity for evaluating a design against its requirements
  - Check whether a product (designed or implemented) achieves its expected functionality and quality attributes
  - Identify potential issues to be addressed
- An important part of a software development process in practice
- Not the same as code review!
  - Design review: Focus is on higher-level design decisions
  - Code review: Focus is on the quality of the source code (e.g., correctness, readability, etc.,)

Design Review in Practice



Design Review at Google



Requirements

- Automatically book gym **two days in advance, at midnight**
- The program needs no human interaction after started, should be fault tolerant and **reasonably retry**
- The program runs on a Mac
- User can specify username, password, the sport to book, date and time to book, etc

Non-goals:

- **Book only 1 or 2 days in advance, or for the current day**
- Tolerant to OS or network issues
- Functioning if booking server is down
- Functioning if website structure(HTML) changes

High-level design

Browser automation V.S. request simulation

Browser automation is using a program to control a real browser and automate the operation on the GUI. Request simulation is having the program talking with the server via HTTP, as if it *is* a web browser, rather than controlling one.

Browser automation is preferred over request simulation. The considerations are:

- **[pros]** Browser automation started a real browser instance so we know what's going on when the program runs, it makes debugging and development much easier
- **[pros]** The website requires javascript to load the controls, this is hard to implement programmatically, may need to control some rendering engine
- **[cons]** Browser automation depends on the HTML structures while request simulation depends on the HTTP APIs. APIs are far less likely to change.

- Widely performed at Google
- Design docs are written using Google Docs
- Stakeholders leave comments directly on the docs

Improving Design Reviews at Google.
Ziftci & Greenberg. IEEE/ACM ASE (2023).

Challenges with Design Reviews

Documenting for Design Reviews

- Code is a poor abstraction for understanding why/how design works
- To facilitate a design review, design decisions must be documented
- We have already discussed different notations for documenting a design:
  - Context (domain) models
  - Component diagrams
  - Data models
  - Sequence diagrams
- But these notations don't explicitly say **why** the design decisions were made, and **how** they support the system in achieving desired functionality

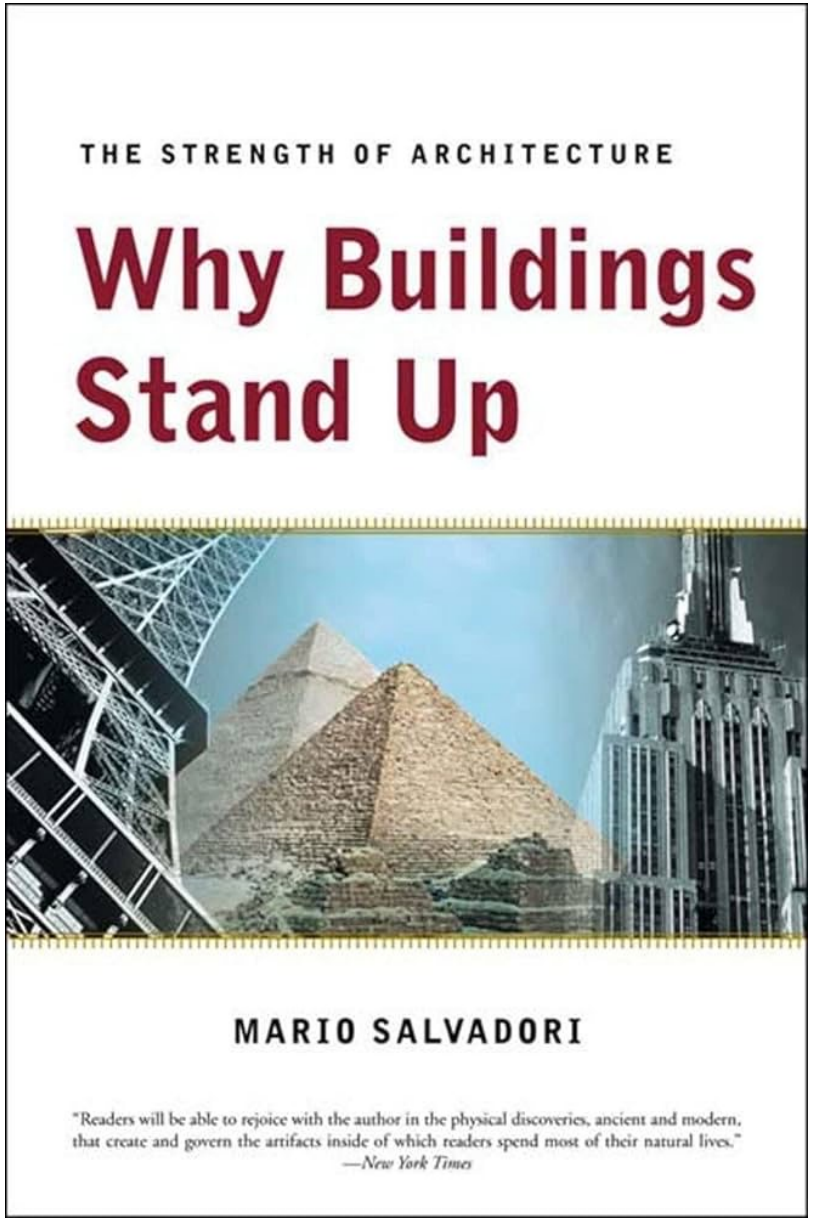
Today's Class

- **Design argumentation**: Devising and documenting an explicit **argument** for why the system design achieves its expected functionality
- **Design review**: Identifying weaknesses in the argument & suggesting ways to improve the design

Design Arguments

Design Argumentation

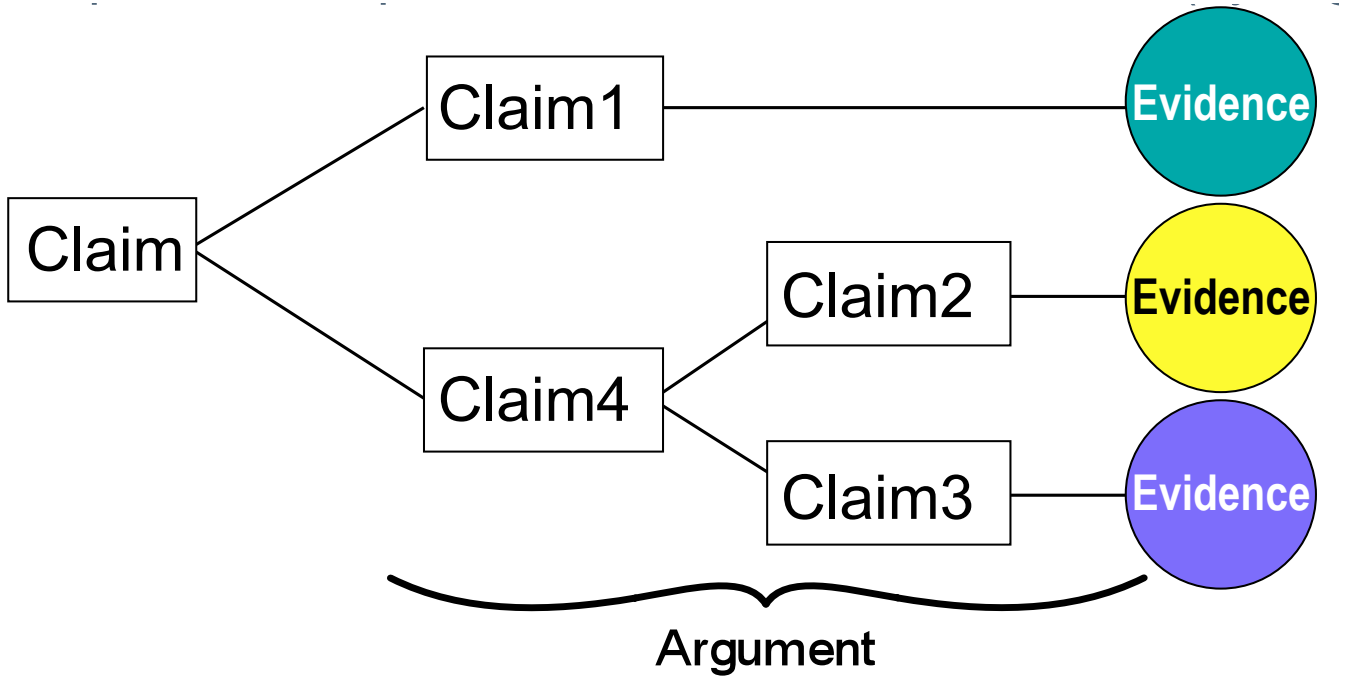
- **Goal**: Argue “why my design works”
- An argument is often implicit and incomplete in the designer’s mind
- If you can’t produce a strong argument, how do you know that your system works?
- Allow another person to review & identify weaknesses in the argument
- **One approach**: Assurance case
  - Assurance: The process of demonstrating that the system will function and satisfy its quality attributes as intended



Assurance Case

- An explicit argument that a system achieves a desired requirement, along with supporting evidence
- **Claim:** A statement about a piece of functionality or quality attribute of the system
- **Argument**: A top-level decomposed into multiple, hierarchical sub-claims
- **Evidence**: A documented piece of evidence that supports a leaf sub-claim
  - Results of testing, software analysis, formal verification, inspection, expert opinions, architecture design
  - Must be auditable & verifiable independently by a third party

Assurance Case: Structure



IF ● **THEN** Claim1; **IF** ● **THEN** Claim2; **IF** ● **THEN** Claim3;
IF Claim2 **and** Claim3 **THEN** Claim4; **IF** Claim1 **and** Claim4 **THEN** Claim

Example: Sidewalk Delivery Robot

Building an Assurance Case

1. Identify a **top-level claim** to demonstrate: A statement about a piece of desired functionality or a quality attribute
   - The intrusion detection system notifies the homeowner in time when a stranger appears around the house (**functionality**)
   - The movie streaming app delivers its content at 1080p resolution with less than 1 second buffering event (**performance**)
   - The stock tracker app can be extended with new types of output format without impacting the rest of functionality (**changeability**)
   - The sidewalk robot avoids collision with pedestrians (**safety**)

Assurance Case: Delivery Robot

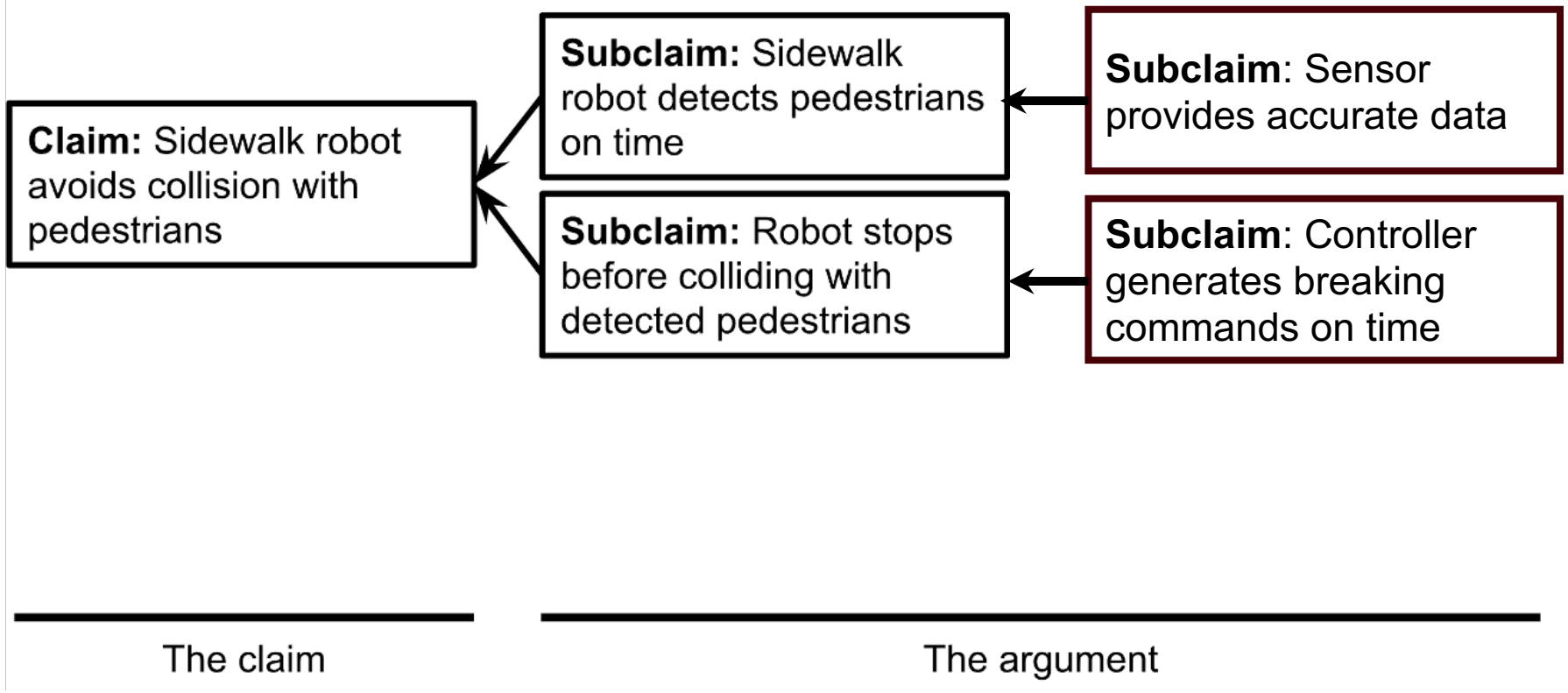
Claim: Sidewalk robot avoids collision with pedestrians

The claim

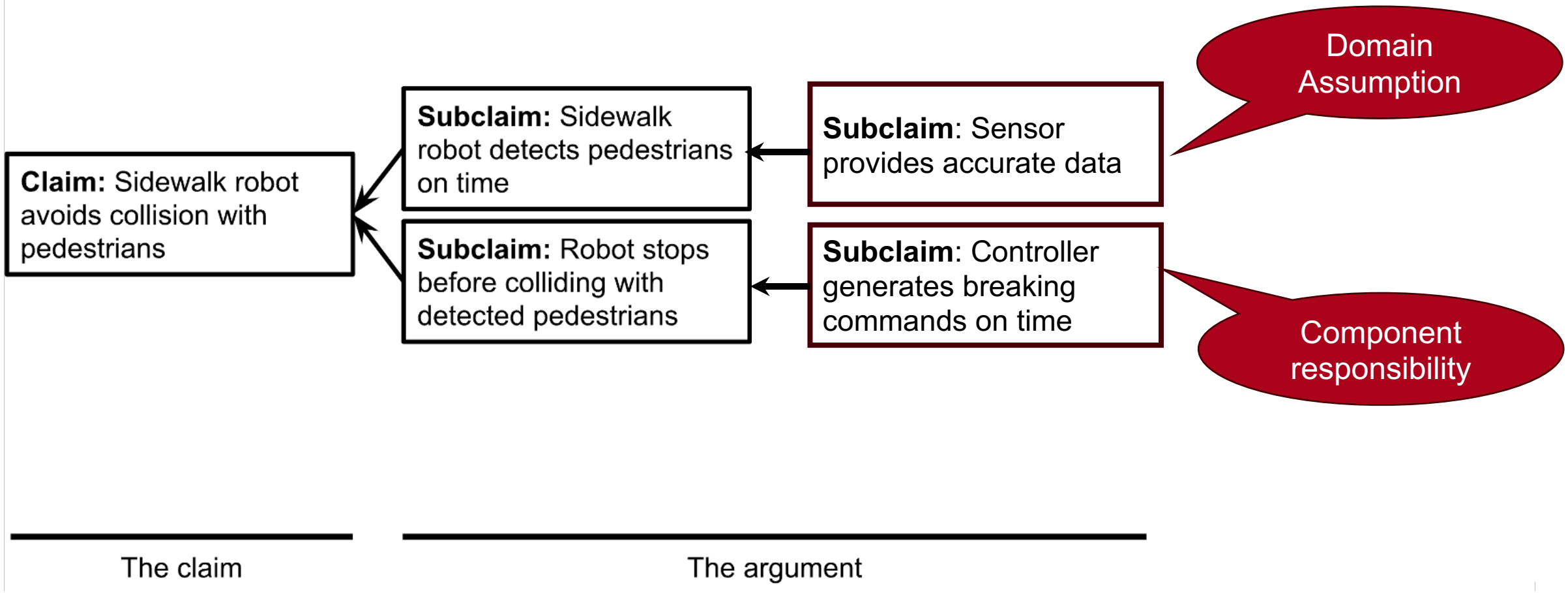
Building an Assurance Case

1. Identify a **top-level claim** to demonstrate: A statement about a piece of desired functionality or a quality attribute
2. Identify one or more **sub-claims** to support a higher-level claim.
   - Logically, "If all the sub-claims hold, then their parent claim also holds”
   - Each sub-claim can, in turn, be decomposed into further sub-claims
   - Each leaf-level sub-claim describes (1) the responsibility of a software component or (2) an assumption about a domain entity

Assurance Case: Delivery Robot



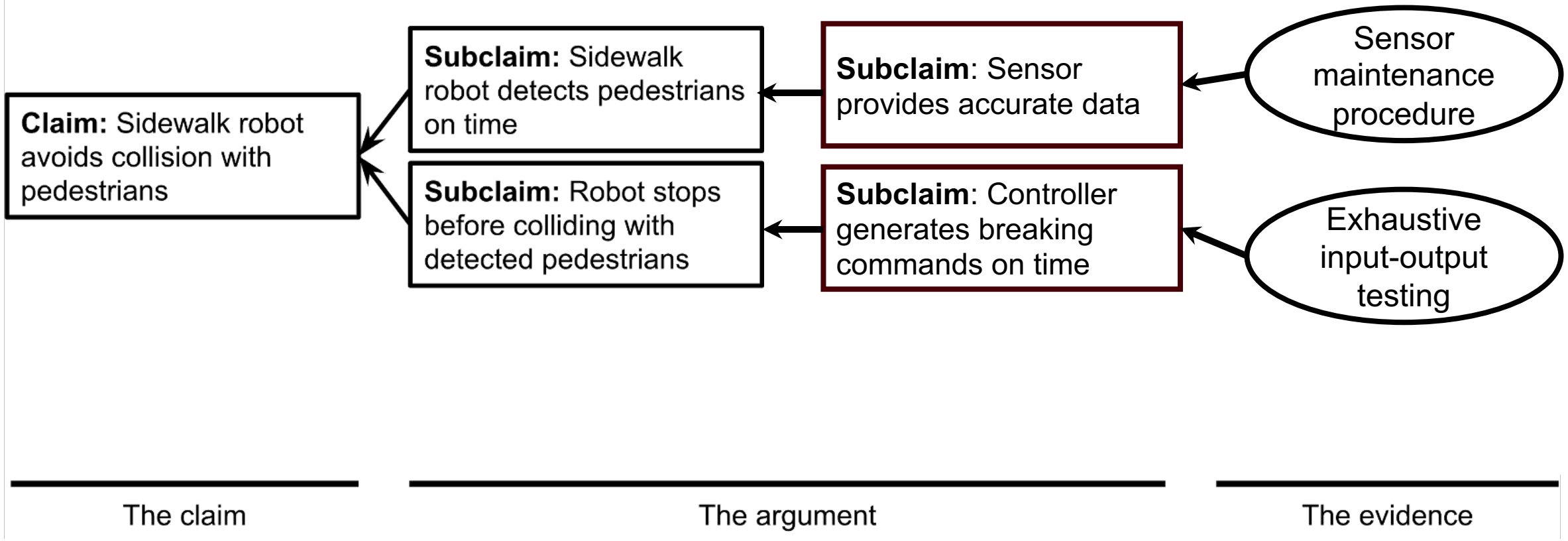
Assurance Case: Delivery Robot



Building an Assurance Case

1. Identify a **top-level claim** to demonstrate: A statement about a piece of desired functionality or a quality attribute
2. Identify one or more **sub-claims** to support a higher-level claim.
3. For each leaf-level sub-claim, provide a piece of evidence to support the claim
   - **Results of testing or program analysis** (e.g., “The app successfully handled stress testing with 1,000 user requests per second”)
   - **Design decisions** (e.g., ”Backup servers are deployed in case the primary one fails” or “An interface is used to hide details about the format of a stock quote from its clients”)
   - **Empirical data** (e.g., “The battery is expected to last 3 months before failing”)
   - **Procedures** (e.g., “The battery is replaced regularly by the user”)

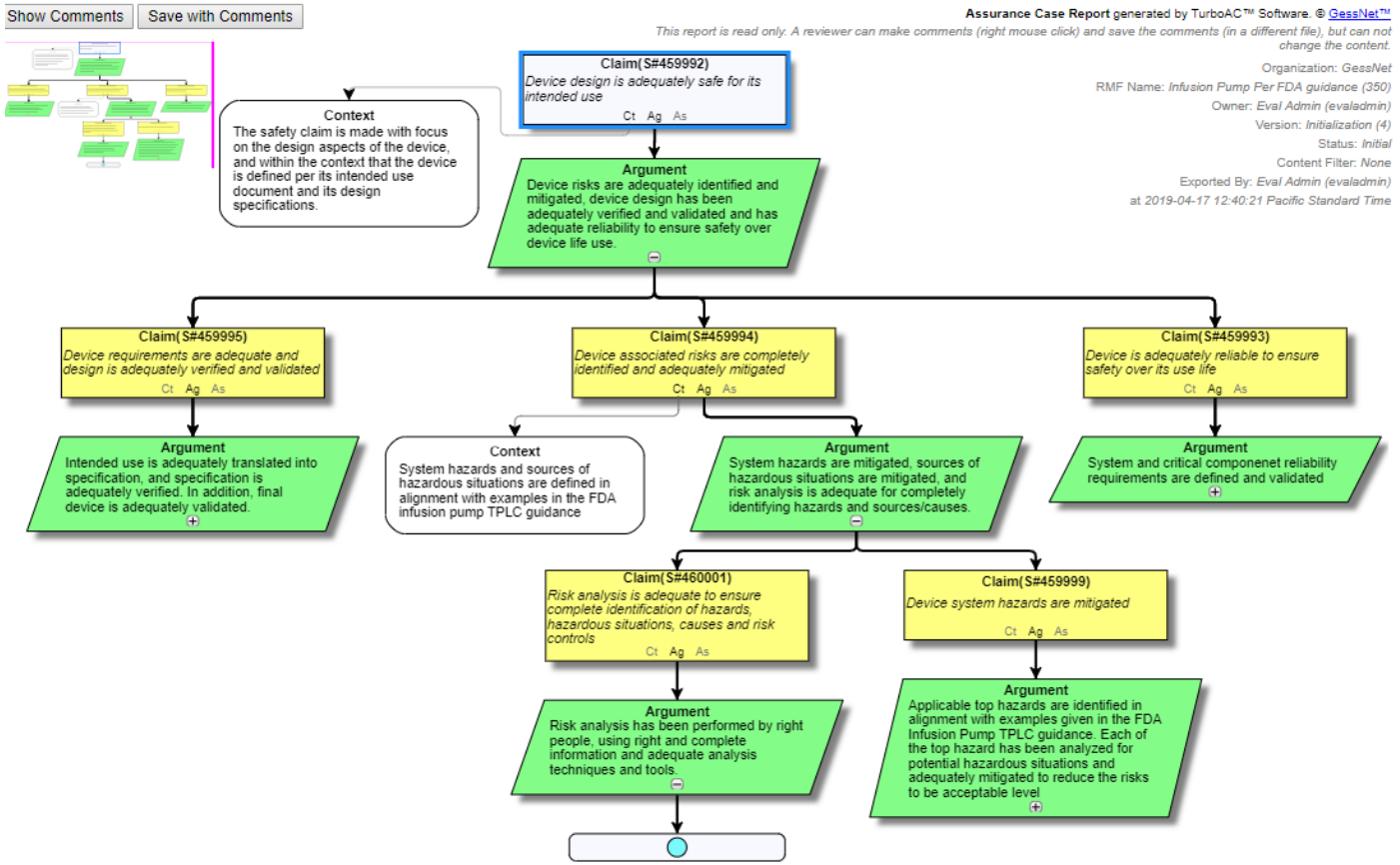
Assurance Case: Delivery Robot



Assurance Cases in Practice

https://blog.aurora.tech/safety/aurora-unveils-first-ever-safety-case-framework

Assurance Cases in Practice



Introduction of Assurance Case Method and its Application in Regulatory Science. Fubin Wu (2019).

Assurance Case: Benefits & Limitations

- Provides an explicit structure to a design argument
  - Encourages the designer to articulate why their design works
  - Easier to navigate, inspect, and refute for reviewers
  - Provides traceability between system-level claims & low-level evidence
- Challenges and pitfalls
  - **Completeness**: How do I know whether it's missing any sub-claims?
  - Effort in constructing the case & evidence: How much evidence is enough?
  - System evolution: If system changes, must also recreate the case & evidence
- Recall: **Risk-driven design**!
  - Build an assurance case for the most important functionalities or quality attributes

Exercise: Assurance Case for IntelliGuard

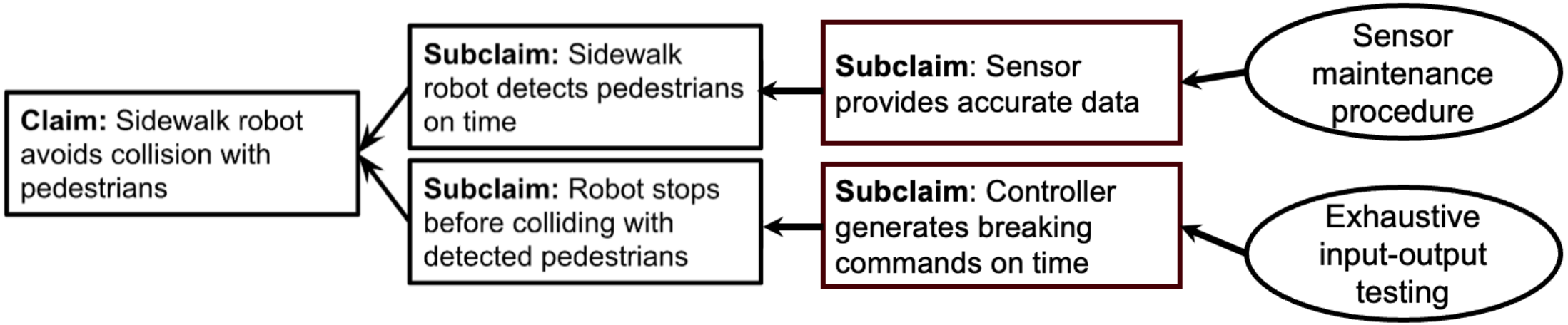
- Recall **IntelliGuard** from HW1
- Break into groups; pick one person's design from HW1
- For that design, develop an assurance case for the following top-claim: "The intrusion detection system notifies the homeowner in time when a stranger appears around the house"
- For evidence, include **hypothetical** pieces of evidence that you would include (assuming you had implemented & tested the system)
- Make sure the assurance case is legible; you will share it with your classmates later

Design Review

Design Review

- **Goals**
  - Improve the quality of a design by identifying and addressing flaws or weaknesses
  - Communicate and align the understanding of the design with other teams and stakeholders of the system
  - Indicate that the product is ready for release or the next phase of development
  - Track changes and improvements to the system design over time
- There are no ”standard” practices or methods for design reviews
- We will use an assurance case as the basis for reviewing a design

Criteria for Reviewing an Assurance Case

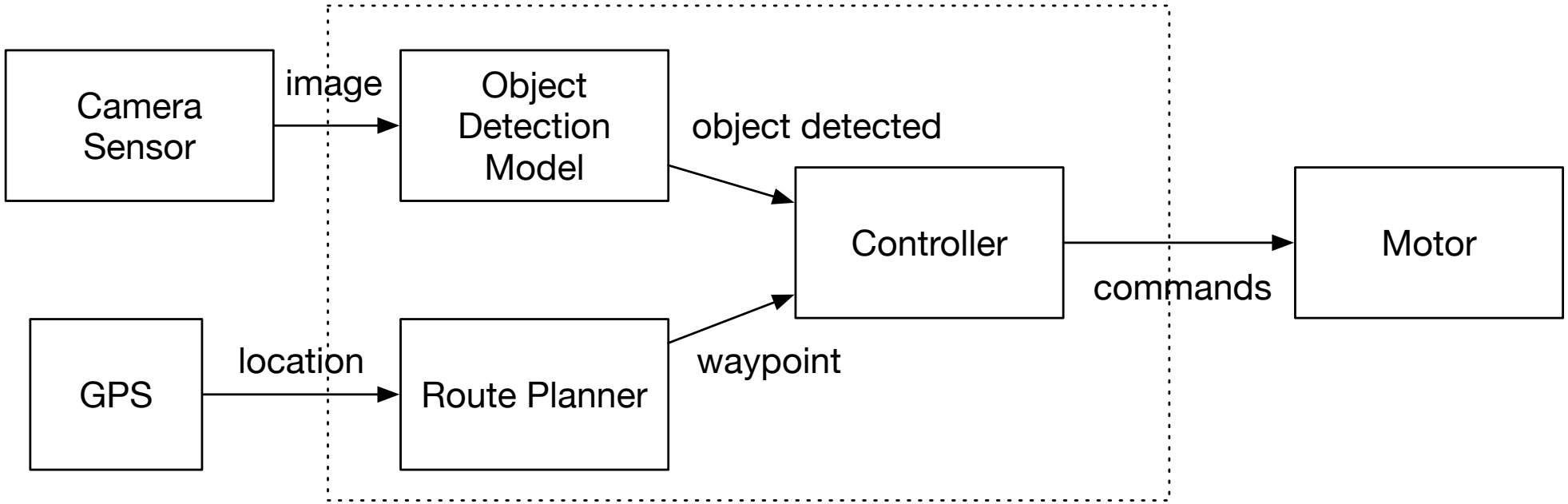


- **Soundness**: Do the sub-claims imply their parent claim? Are there any missing sub-claims?
- **Validity**: Is the evidence strong enough to support a leaf claim? Can the evidence be independently verified (e.g., by re-running the test cases)?

Adversarial Thinking

- Think like an “attacker”, not the designer of the system
- As a reviewer, your goal is to **invalidate** the argument; i.e., show how the system may fail to satisfy the claim in certain scenarios
- For each **leaf sub-claim**: Think of a scenario where it fails to hold due to insufficient evidence (**validity flaw**)
- For each **non-leaf sub-claim**: Think of a scenario where all its children sub-claims hold but it does not (**soundness flaw**)

Component Diagram for Delivery Robot



- Consider domain entities & components that are involved in achieving the desired functionality/quality attribute
- Is there an assumption or responsibility missing from the argument?

Reviewing Evidence

- **For testing & program analysis reports**: Re-run the tests or analysis under the identical conditions (if possible) and compare the output. Check whether the result (e.g., test coverage) is strong enough to support the sub-claim.
- **For design decisions**: Review the design document (e.g., component diagram) and the code to ensure that the documented decisions are implemented properly in the system.
- **For procedures**: Check the procedure; often requires domain knowledge!
- **For empirical data**: Apply proper statistical methods to ensure the validity of the presented data

Sample Review Comments

- **Soundness flaw**: The sub-claim “Sensor provides accurate data” is not enough to ensure “Sidewalk robot detects pedestrians on time”, since the object detection model may fail to detect a pedestrian even if it’s given an accurate image from the sensor.
- **Validity flaw**: There is not enough evidence to support the sub-claim “Sensor provides accurate data”. Sensor might fail during deployment between maintenance procedures and cause the robot to ignore a pedestrian.

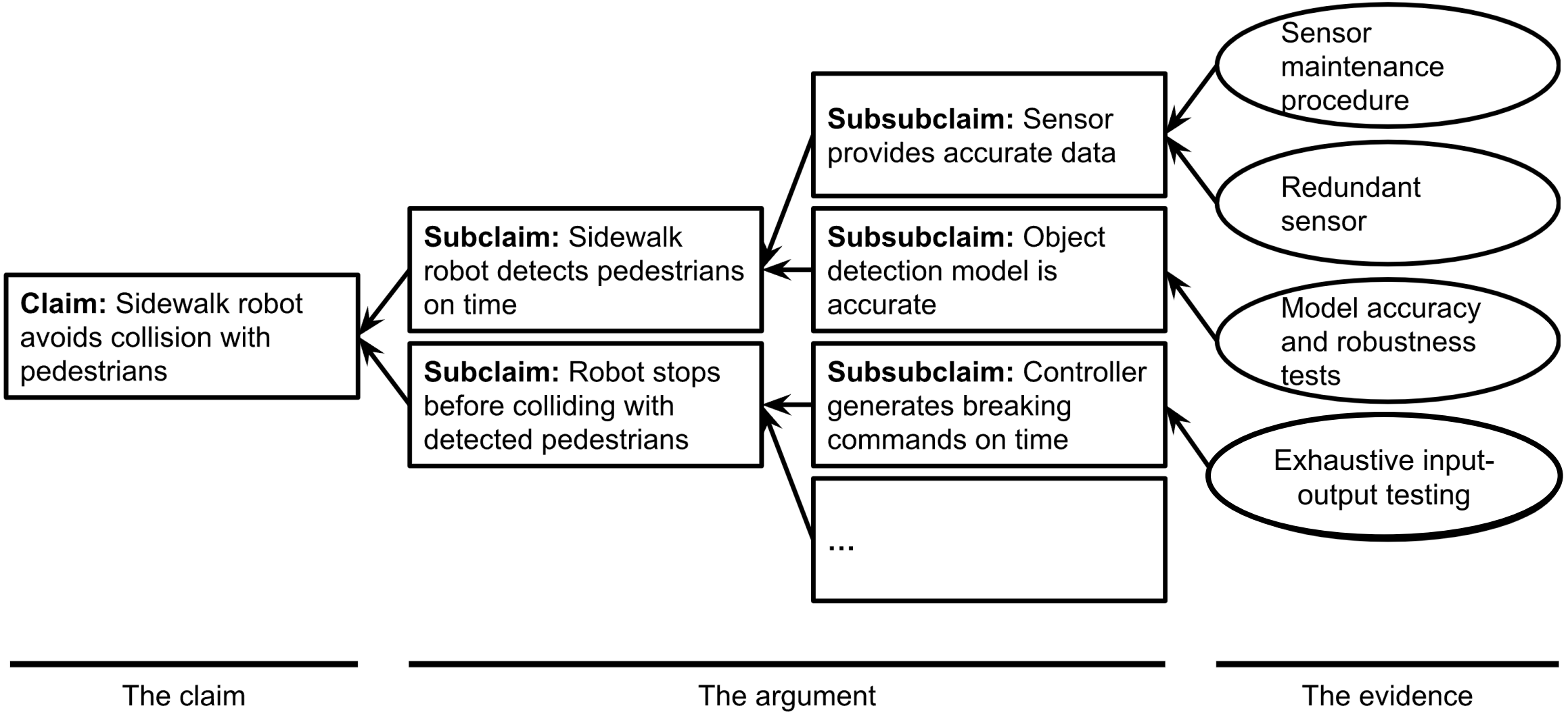
Responding to Review Feedback

- **Refute the feedback!** It is possible that the reviewer misunderstood the design/argument. Explain why the feedback is incorrect.
- **Do nothing but put on backlog**: The identified flaws might not be significant enough to be addressed now, but can be revisited later
- **Improve the argument**
- **Improve the design, if the former is not possible.**
- Send the revised assurance case back for a further review; repeat until no more feedback

Improving the Argument

- For each **leaf sub-claim**: A scenario where it fails to hold (due to insufficient evidence)
  - Add additional pieces of evidence to support the sub-claim
- For each **non-leaf sub-claim**: A scenario where all its children sub-claims hold but it does not
  - Add a new sub-claim(s) to ensure that the parent claim is implied by its children
  - The sub-claim **must correspond** to a domain assumption or a responsibility of an existing software component
- If no further evidence or sub-claim can be added to fix the argument, then a valid argument does not exist – the design itself must be fixed!

Improved Assurance Case for Delivery Robot



Exercise: Assurance Case for IntelliGuard

- Take (1) an assurance case and (2) a component diagram for IntelliGuard from another group
- Review the assurance case and identify potential flaws with respect to soundness and validity
- Discuss the flaws identified by the other group: (1) refute if they are not flaws or (2) devise ways to improve the argument or design to address those flaws

Design Review: Tips

- Be constructive! The goal is to help improve the design, not to shoot it down
- Don’t nitpick; look for larger problems that could lead to significant risks for the project
- Take a risk-driven approach! Focus on claims about most important functionalities or quality attributes
- Recruit outsiders (e.g., customers, engineers from another team) for review, to reduce bias
- Keep a record of suggestions from the reviewers; track which of those suggestions have been implemented
- Do design reviews regularly, after each project milestone or iteration

Summary

- Exit ticket!