Plan Review: Requirements & Validation

- If we don't get the requirements right, it does not matter how good we build our system.
- The world of *requirements*, can be divided into a few parts:
  - **Eliciting** requirements: how do we **capture** the requirements
  - **Documenting** requirements: how do we **keep track** of the requirements
    - **Formal specifications**: well-defined syntax and semantics (usually mathematical)
    - **The 4 variable model**: relationship between requirements and design ← *** WE ARE HERE ***
  - How do we know if we elicit and document requirements **correctly**?
- *Validation*: determining whether or not we got the *right* requirements
  - Not to be confused with *verification*, which is determining whether or not we built the system *right* (i.e., if it complies with the requirements)
  - We often refer to Validation and Verification as V & V.

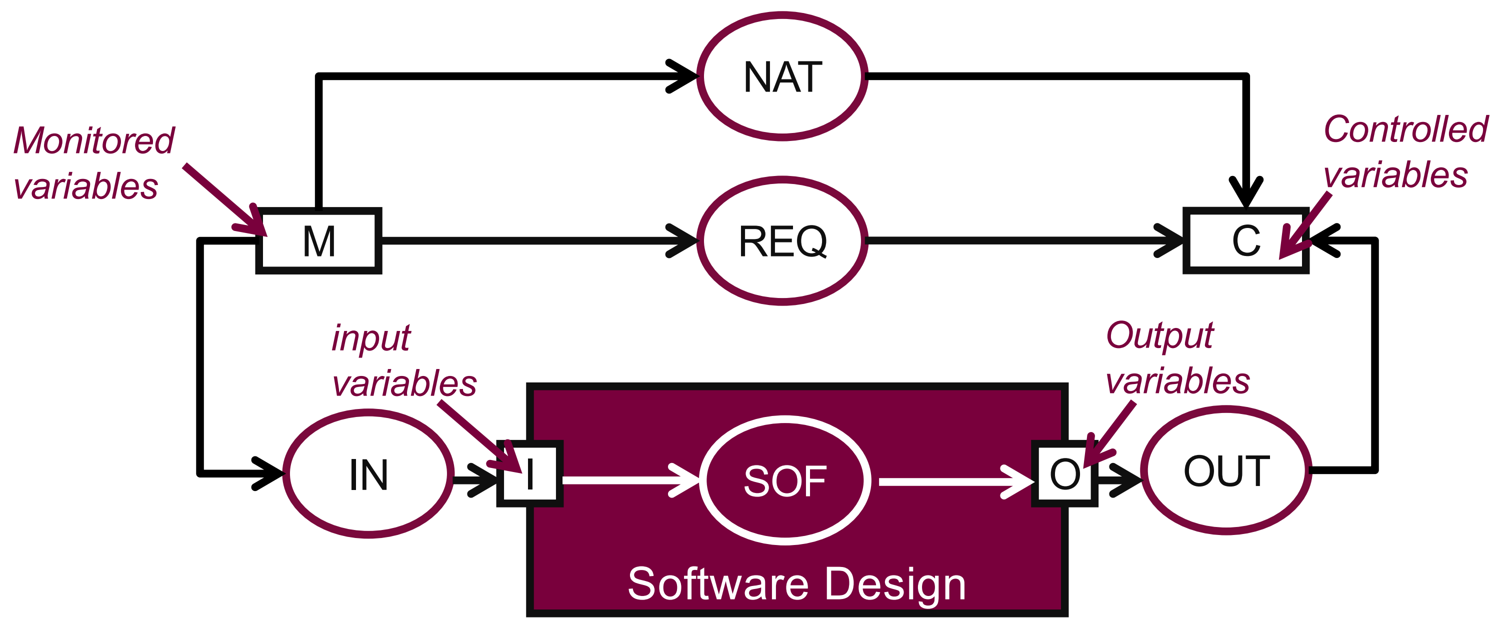
The 4 Variable Model

- Describes the essential relationship between requirements and design of embedded and safety-critical systems
- Illustrates why the software design has different input and outputs from the requirements
    - I.e. Way to keep requirements (what the system must achieve in the environment) separate from implementation details (how we make that possible)
- It is often used to show the relationship between a system and the real world.
- Parnas, D.L., Madey, J.: Functional documents for computer systems. Science of Computer Programming 25 (1995) 41-61

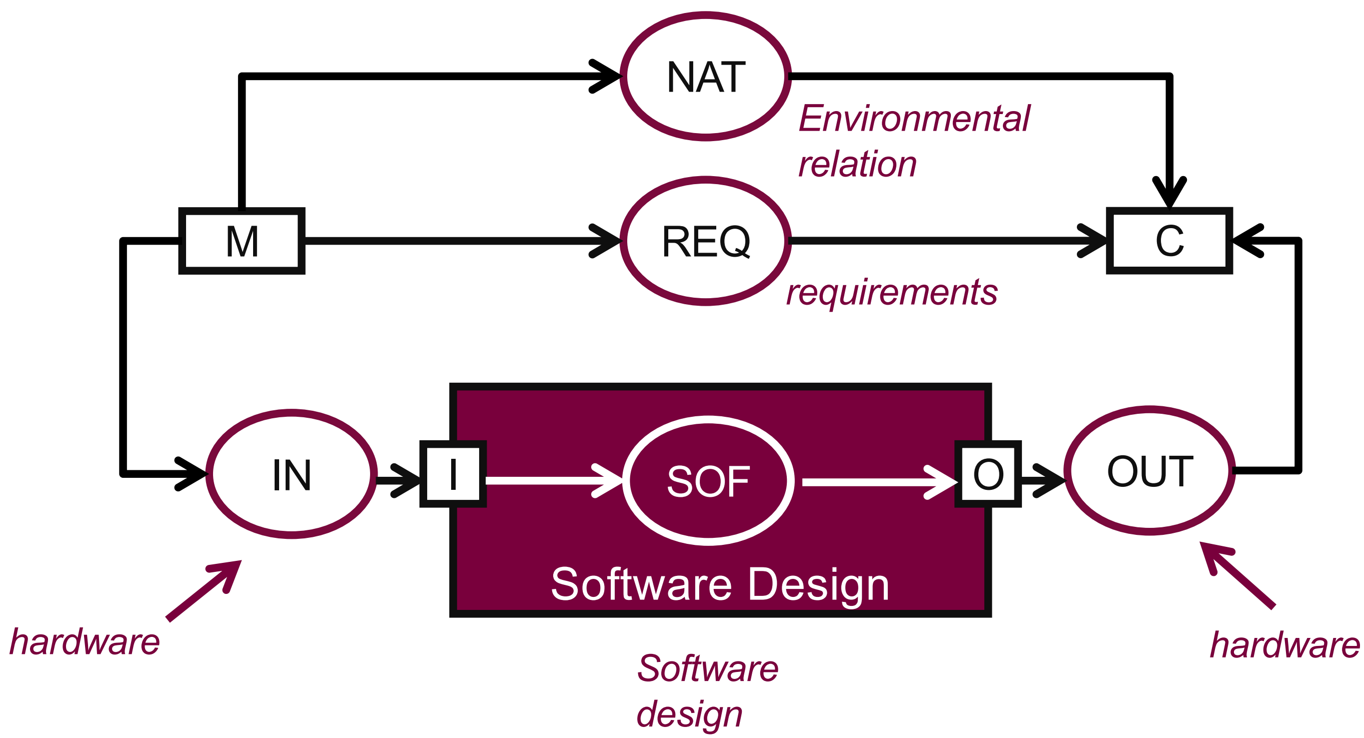
The 4 Variable Model

- Monitored variables (M): Properties in the environment that the software needs to watch
    - E.g., button pressed
- Controlled variables (C): Properties in the environment that the software is supposed to control
    - E.g., setpoint requested
- Input variables (I): What the software *receives* from sensors
    - Representations of M
- Output variables (O): What the software *sends* to actuators
    - Representations of C

The 4 Variable Model: Variables

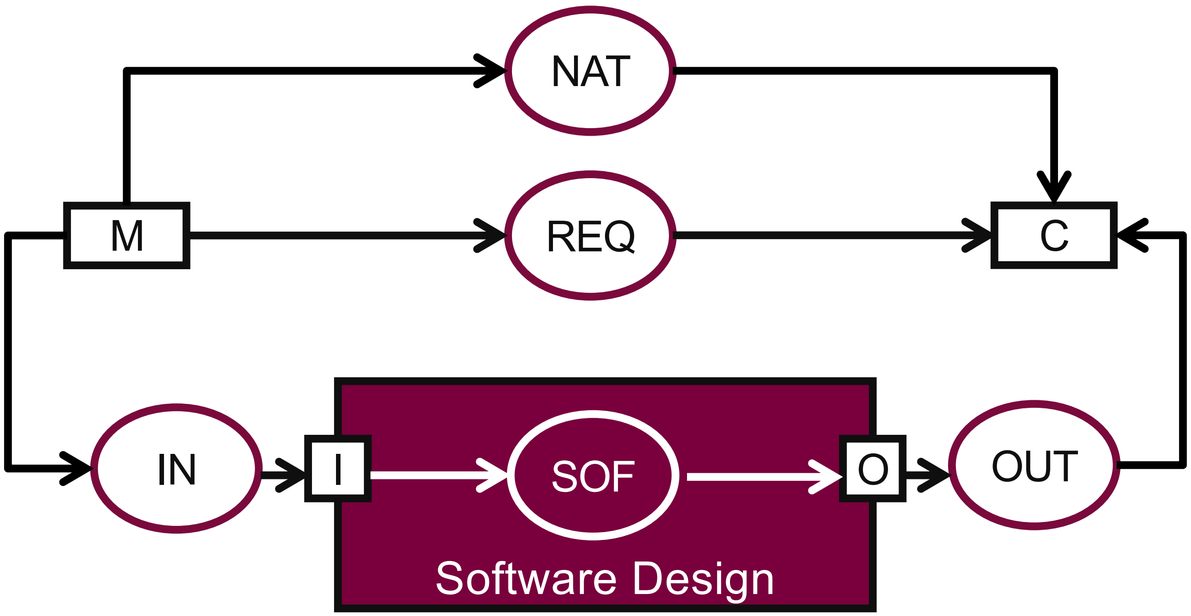


The 4 Variable Model: Relations/Functions



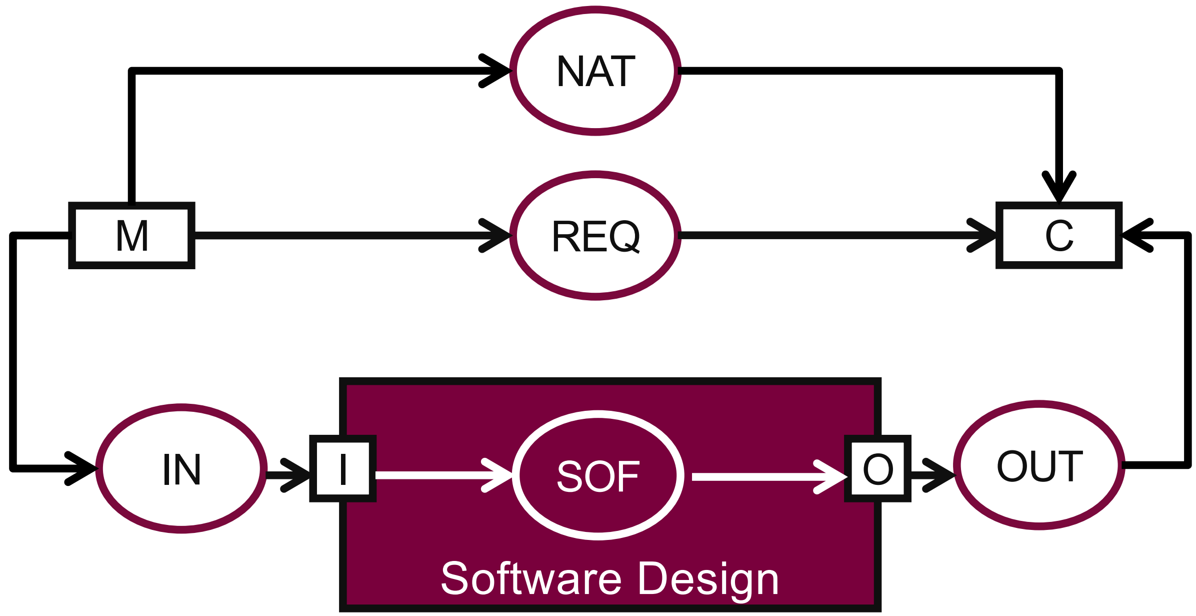
The 4 Variable Model: Formal Relationships

- IN (sensor model)
  - How real-world monitored values are turned into software inputs
- OUT (actuator model)
  - How software outputs affect the environment
- SOF (software design)
- REQ (requirement)
  - What we *want* the system to achieve
- NAT (natural laws)
  - How the environment *behaves*



The 4 Variable Model: Formal Relationships

- C = REQ(M)
  - The requirements describes the intended relationship between the environment and variables
- I = IN(M)
  - Input devices transform monitored variables into input signals
- C = OUT(O)
  - Output devices transform software outputs into real-world controlled variables



SOF: I -> O such that OUT(SOF(IN(M))) = REQ(M)

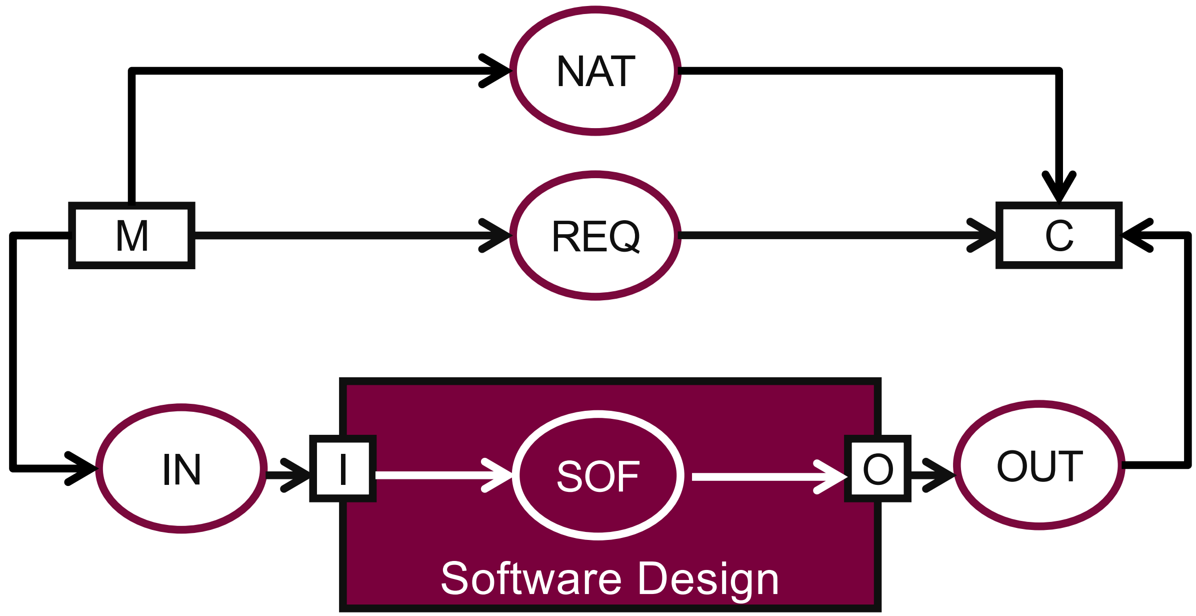
The 4 Variable Model: Example

Assume we have a sensor that converts physical temperature in Celsius to an unsigned 12-bit integer

We can also determine that *i* = round(*m**4/5), where *m* is the monitored variable (temperature) and *i* is the 12-bit integer that represents m in the software.

<u>Example</u>: if the room is 25 C, the sensor IN will convert that into *i* = 50.

The software will never see "25 C", only the encoded integer.

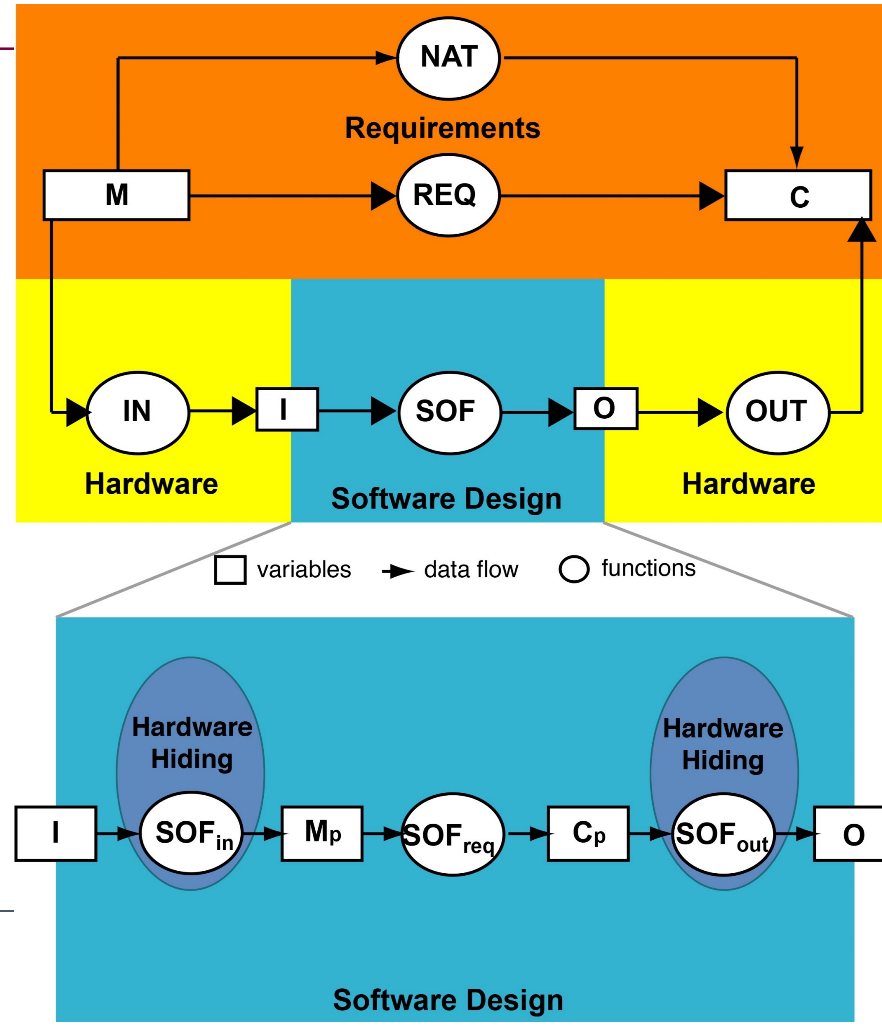


The 4 Variable Model in Safety-Critical Systems

- Regulators care about real-world requirements (M and C)
  - REQ is stated in terms of M and C
- Software engineers must translate these into implementations using I and O, while documenting assumptions about IN and OUT mappings
  - SOF works with I and O
- If the mapping is ignored, you risk dangerous mismatches
- TLDR:
  - Write requirements in terms of monitored (M) and controlled (C) variables
  - Write design in terms of input (I) and output (O) variables
  - Explicitly document the IN and OUT mappings (assumptions about sensors and actuators)
- We have to remember that in software, we use i (in I) instead of m (in M).

The Modified 4 Variable Model

- M_p and C_p are called pseudo-M and pseudo-C: software-usable versions of those variables
- SOF_{in}: Converts raw input variables I into a software-usable representation M_p
- SOF_{req}: Works on M_p and decides what the controlled properties C_p should be
- SOF_{out}: Converts the desired controlled property C_p into the actual actuator O
- Hardware hiding: Encapsulates the details of I and O, isolating the rest of the software from hardware-specific details.
  - This makes the design portable and testable (you can simulate M_p and C_p without real hardware)



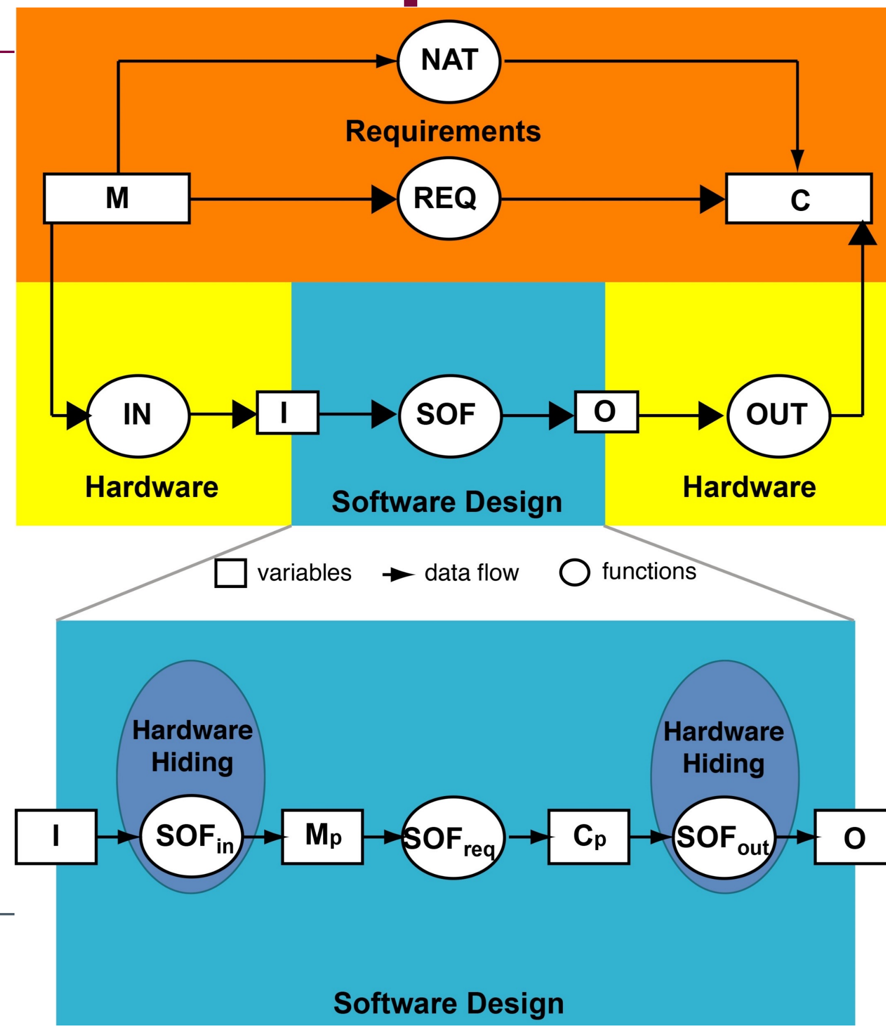
The Modified 4 Variable Model: Example

Assume we have a sensor that converts physical temperature in Celsius to an unsigned 12-bit integer

We can also determine that i = round(m*4/5), where m is the monitored variable (temperature) and i is the 12-bit integer that represents m in the software.

If we want to produce m_p so that it is close to the original m, we can multiple i by 5/4, so we get: $m_p = i*5/4$

$| m - m_p |$ < error bound



Time for a larger example

Home Alarm System: Background Details

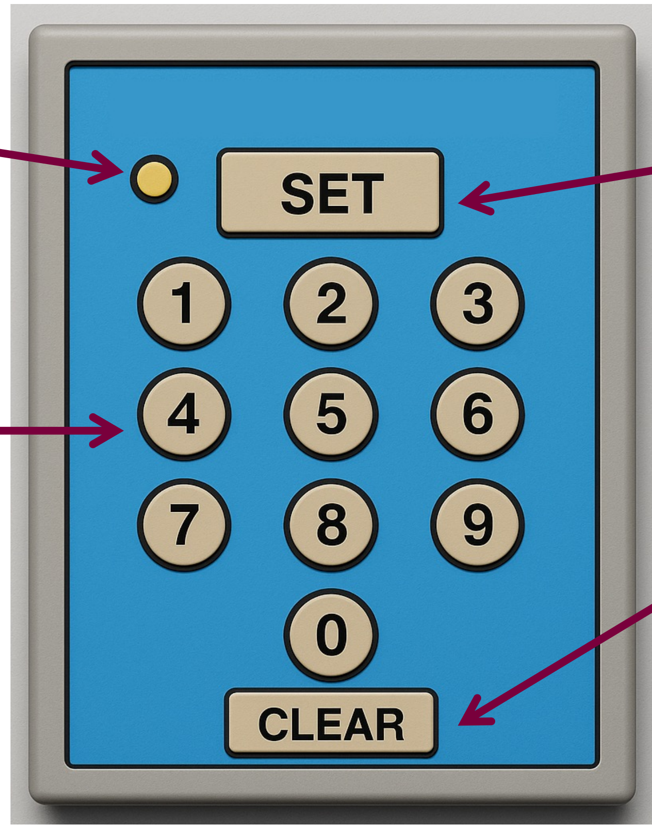
- Who is asking for an alarm system?
  - Who will use the alarm system?
  - I.e., stakeholders
- Why do we want this alarm system?
  - The 5 whys technique: https://archive.org/details/toyotaproduction0000onot
  - On average, it takes 5 whys to get to the root of a problem
- What will the alarm system do?
- Where will the alarm system go? Where will it be used?
- When will the alarm system be used?

Home Alarm System: Background Details

Light on when the system is live

Button to make system live

Buttons to enter 3-digit code to stop the alarm sound and deactivate alarm



Button to clear erroneous digit entry and re-enter all digits

Sensors are connected to the panel and will make the alarm sound when motion is detected and system is live

Home Alarm System: Requirements

1. The security system has a detector that sends a trip signal when motion is detected.
2. The security system is activated by pressing the SET button.
3. The SET button is illuminated when the system is active.
4. If a trip signal is detected while the system is active, an alarm is sounded.
5. A three-digit code must be entered to turn off the alarm sound.
6. Correct entry of the three-digit code deactivates the device.
7. If a mistake is made when entering the code, the user must press the CLEAR button before the entire code can be re-entered.

Are these requirements complete? Unambiguous? Clear?