SFWRENG / MECHTRON 3K04 Software Development

Today: More Requirements

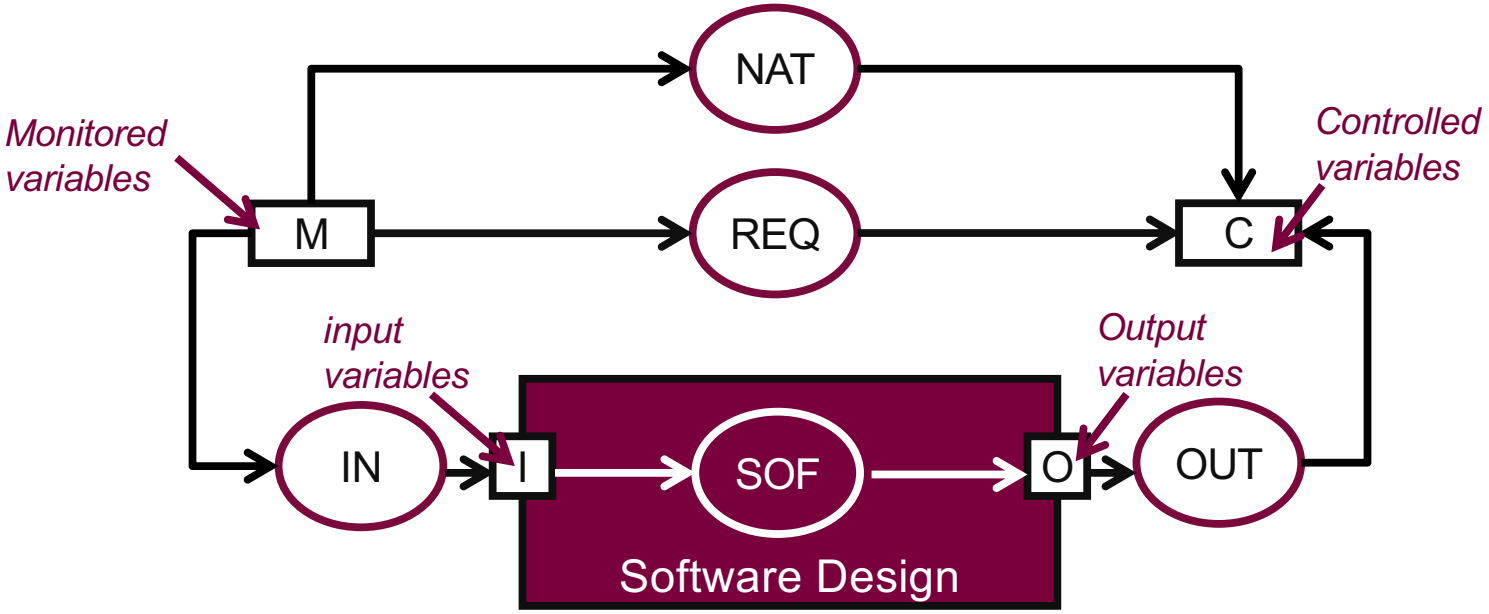
McMaster University | Engineering
Computing & Software

September 22nd, 2025

Dr. Angela Zavaleta Bernuy

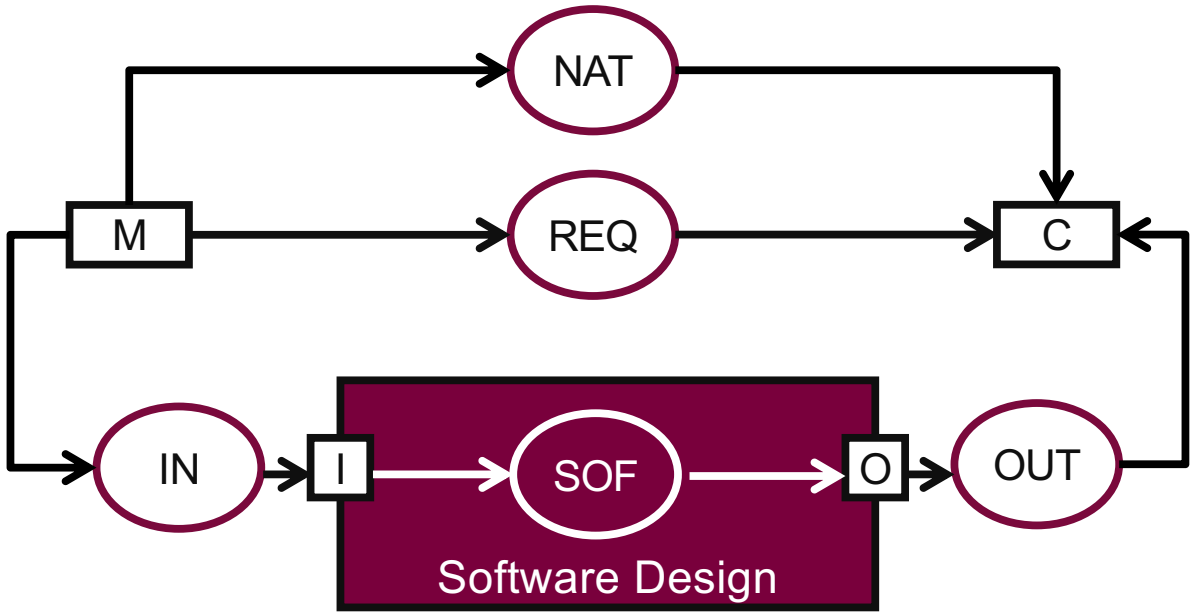
Let’s review a bit of what we did last week!

The 4 Variable Model: Variables



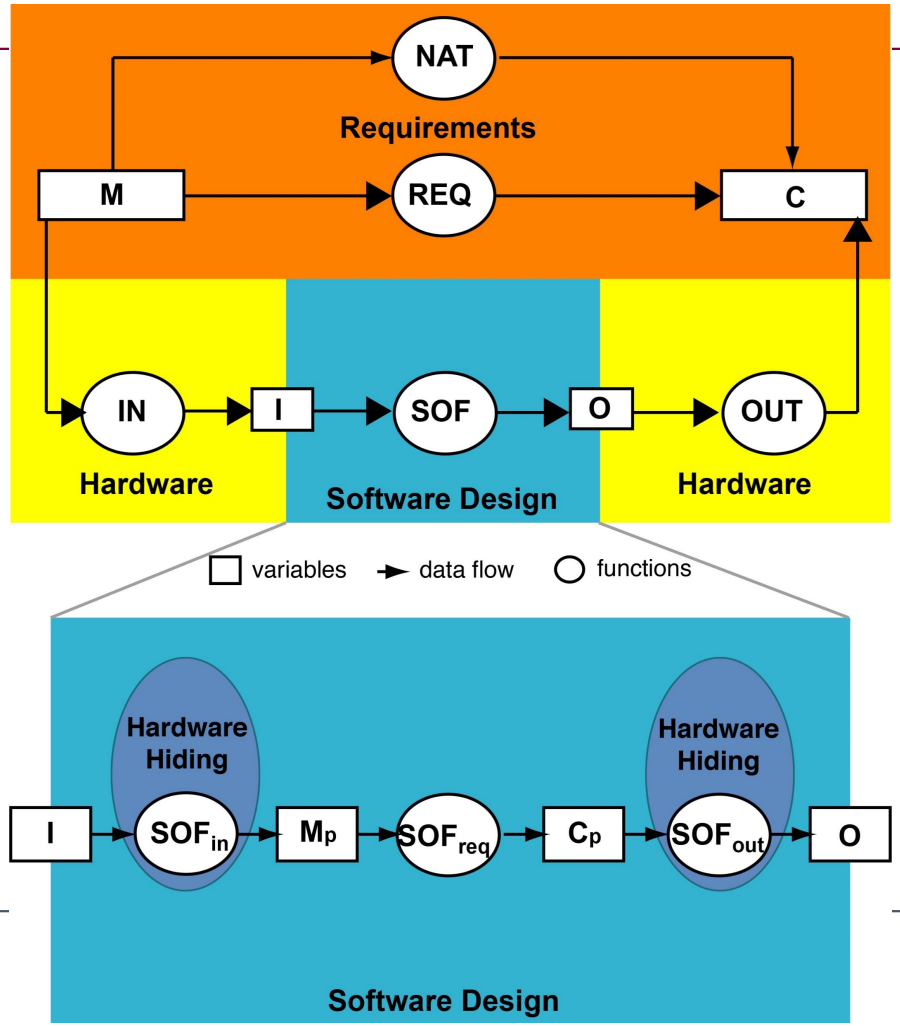
The 4 Variable Model: Formal Relationships

- 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 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)

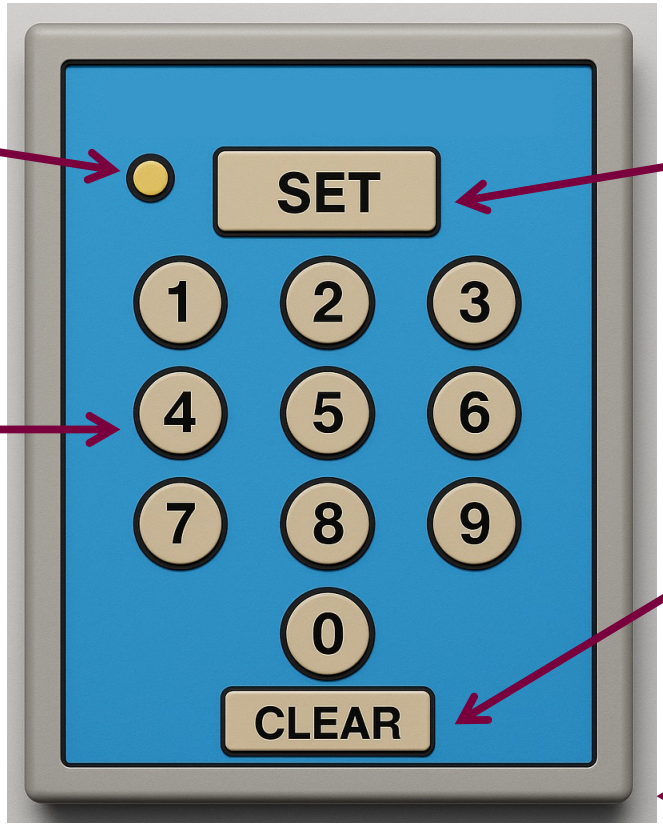


Time for a larger example

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

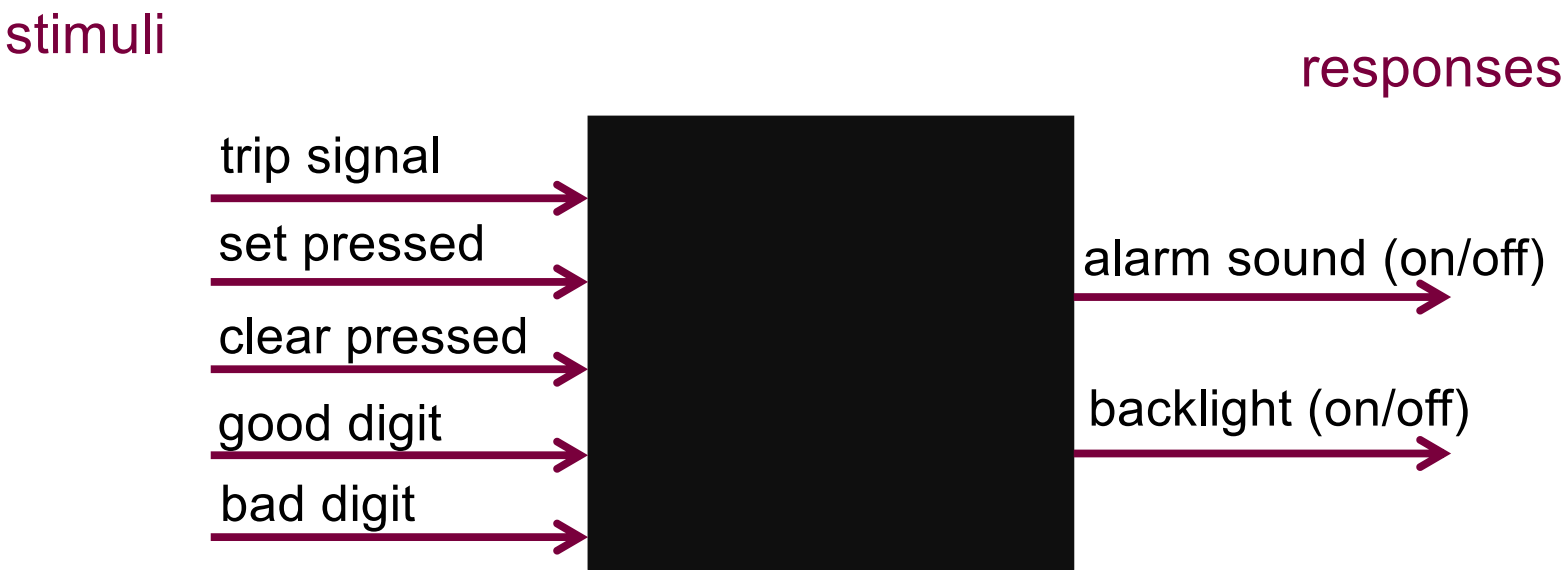
Sensors are connected to the panel and will make the alarm sound when motion is detected and system is live

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?

Home Alarm System: Notation



Home Alarm System: Notation (4-variable)

Home Alarm System: Input Sequence

- No stimulus
- M_set
- M_set . M_trip
- M_set . M_badDigit
- M_set . M_goodDigit
- M_set . M_trip . M_badDigit
- M_set . M_trip . M_goodDigit
- M_set . M_trip . M_goodDigit . M_goodDigit
- M_set . M_goodDigit . M_goodDigit

For example: M_set . M_badDigit . M_clear = M_set

What if we append any of the stimuli to any of these sequences?

These are called *canonical* sequences, we can append any legal stimulus to any of these sequences and you get one of these sequences as a result.

Home Alarm System: Now what?

1. We understood our problem space
2. We defined the requirements in natural language
3. We transformed the requirements into a 4 variable model notation
4. We have listed canonical sequences
5. Now, we need to assign response values for each canonical sequences
    1. Why does that help?
    2. Because we need to know that for any sequence of stimuli, we know what the response should be.

Home Alarm System: Responses

	c_alarmSound	c_backlight
no stimulus	e_off	e_off
M_set	e_off	e_on
M_set . M_trip	e_on	e_on
M_set . M_badDigit	e_off	e_on
M_set . M_goodDigit	e_off	e_on
M_set . M_trip . M_badDigit	e_on	e_on
M_set . M_trip . M_goodDigit	e_on	e_on
M_set . M_trip . M_goodDigit . M_goodDigit	e_on	e_on
M_set . M_goodDigit . M_goodDigit	e_off	e_on

Is that now sufficient to specify the required behaviour?

No! We need to show effect on ANY new stimulus!

One way to do that is to show the current *state* and effect of each stimulus.

Home Alarm System: Effect of each stimulus

Notation:

Sequence	Name/State
no stimulus – initial condition	device off
M_set	device on
M_set . M_trip	alarm
M_set . M_badDigit	error
M_set . M_goodDigit	1 good
M_set . M_trip . M_badDigit	alarm & error
M_set . M_trip . M_goodDigit	alarm & 1 good
M_set . M_trip . M_goodDigit . M_goodDigit	alarm & 2 good
M_set . M_goodDigit . M_goodDigit	2 good

Behaviour:

Current state

Current stimulus	device off	device on	alarm	error	1 good	2 good	alarm & error	alarm & 1 good	alarm & 2 good
M_badDDigit									
M_clear									
M_goodDigit									
M_set									
M_trip									

New state shown in the cells. - means no change

Home Alarm System: Let's do some thinking...

- Why did we use M_, c_, e_, etc. in our **notation**?
- Does it make sense to have **M_badDigit and M_goodDigi**t as our stimuli?
- How does the sensor value that **detects** the motion get into our software?
- How about the **secret code** to switch off the device or alarm sound? How is it conveyed to the device?
- Let's answer these!

Home Alarm System: Notation

- We already saw Parnas and Madey's 4 variable model that describes how requirements are related to the software design.
  - Stimuli are *monitored variables*, and responses *are controlled variables*.
- Our notation simply uses prefixes to describe the "kind" of identifier
  - M_ or m_ for monitored variables (M_ for time discrete, m_ for time continuous),
  - C_ or c_ for controlled variables
  - e_ is used for enumerated tokens (think "enum" in C, C#, etc.)
- These are conventions. They are not standard across all fields, but they are usual a local (to a domain, company, team etc.) customs to make communication easier.
- This enforces discipline: we don't confuse **variables** with **values**, or confuse **requirements-level environment variables**(M, C) with **software signals** (I, O)

Home Alarm System: Let's do some thinking...

- Why did we use M_, c_, e_, etc. in our **notation**?
- Does it make sense to have **M_badDigit and M_goodDigi**t as our stimuli?
- How does the sensor value that **detects** the motion get into our software?
- How about the **secret code** to switch off the device or alarm sound? How is it conveyed to the device?
- Let's answer these!

Home Alarm System: M_badDigit & M_goodDigit

- It does not make sense to use M_badDigit and M_goodDigit as monitored variables. Why not?
  - They are not really monitored variables
  - The true monitored variables are the digits ‘1’, ‘5’ etc
  - We used M_badDigit and M_goodDigit because it simplified our description
  - We should now determine how to “derive” badDigit and goodDigit given a monitored variable M_digit for instance, where M_digit is one of ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘0’

Home Alarm System: M_badDigit & M_goodDigit

- Given that the system knows the secret code, assume a string of 3 characters called *secretCode*.
- Assume we know M_digit represents a character *c*.

		Result
No digit since initialization or since M_clear		No Change
One digit since initialization or since M_clear	c == secretCode[0]	goodDigit
	c != secretCode[0]	badDigit
Two digits since initialization or since M_clear	c == secretCode[1]	goodDigit
	c != secretCode[1]	badDigit
Three digits since initialization or since M_clear	c == secretCode[2]	goodDigit
	c != secretCode[2]	badDigit
> Three digits since initialization or since M_clear		No Change

Home Alarm System: M_badDigit & M_goodDigit

- So, for current state, given c from M_digit:

		Result
device off || error || alarm & error		No Change
device on || alarm	c == secretCode[0]	goodDigit
	c != secretCode[0]	badDigit
1 good || alarm & 1 good	c == secretCode[1]	goodDigit
	c != secretCode[1]	badDigit
2 good || alarm & 2 good	c == secretCode[2]	goodDigit
	c != secretCode[2]	badDigit

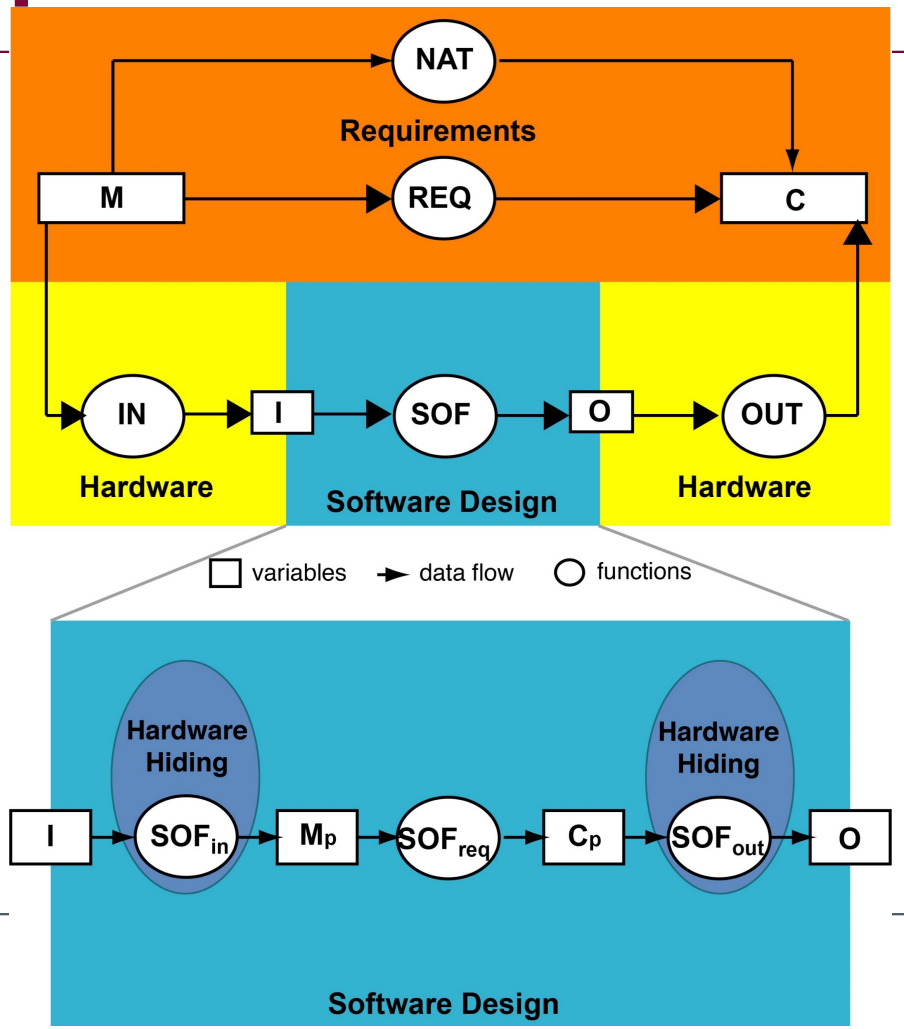
- One way to do this, is to make a different monitored variable for each digit.
  - For instance, M_one ==> ‘1’, etc.

Home Alarm System: Let's do some thinking…

- Why did we use M_, c_, e_, etc. in our **notation**?
- Does it make sense to have **M_badDigit and M_goodDigi**t as our stimuli?
- How does the sensor value that **detects** the motion get into our software?
- How about the **secret code** to switch off the device or alarm sound? How is it conveyed to the device?
- Let's answer these!

Home Alarm System: M_trip

- In a real device, there would be hardware piece that detects motion.
- There is also an interface to the software that delivers the value (the equivalent of a boolean, in this case) so that the software can work with it.
- To test, we can just make a button that the user can click to trigger M_trip
- *Think back to hardware hiding and the modified 4 variable model. Do we care about how we get the Boolean value?*



Home Alarm System: Let's do some thinking...

- Why did we use M_, c_, e_, etc. in our **notation**?
- Does it make sense to have **M_badDigit and M_goodDigi**t as our stimuli?
- How does the sensor value that **detects** the motion get into our software?
- How about the **secret code** to switch off the device or alarm sound? How is it conveyed to the device?
- Let's answer these!

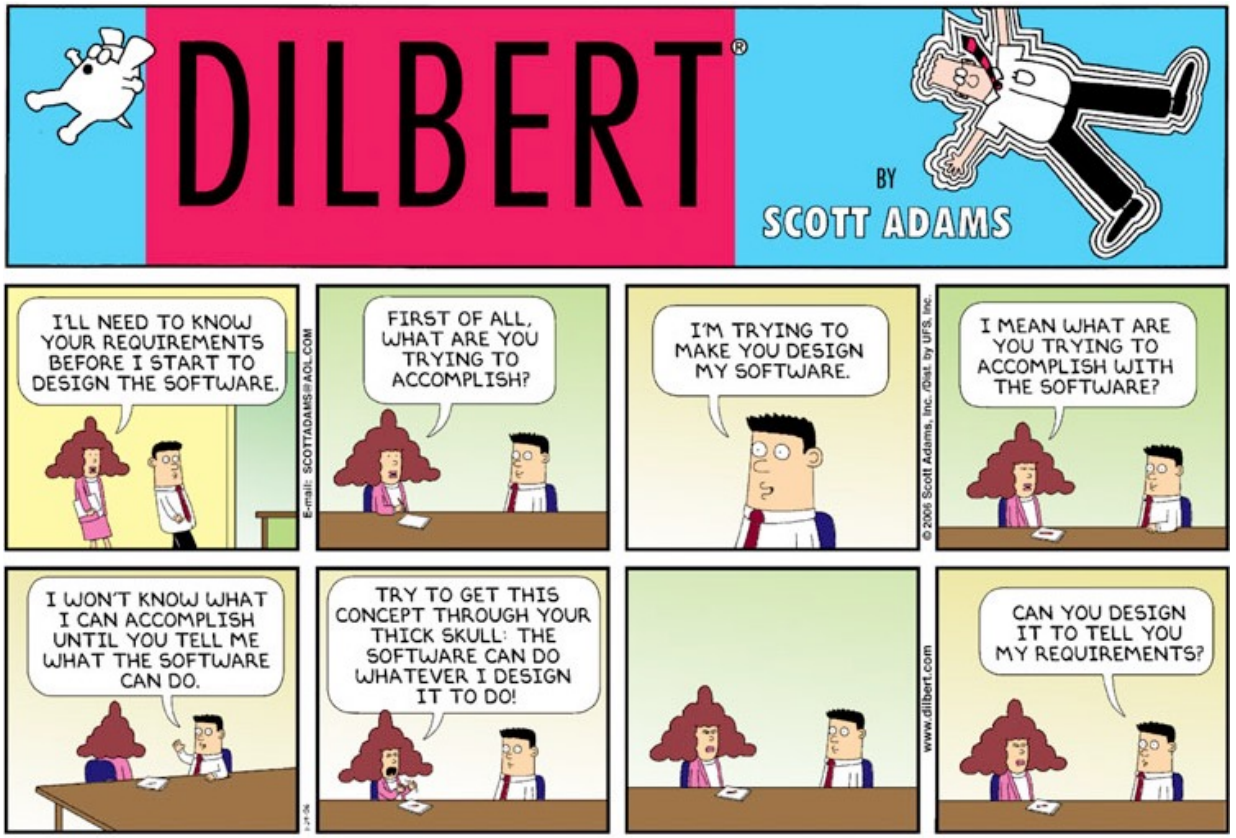
Home Alarm System: The Secret Code

- There are many ways of entering the secret code
- One way is to require that the string of 3 numeric characters must be entered when the Set button is pressed.
  - Therefore, pressing the Set button will result in entering the secret code and then switching the device on
- Making the requirements more realistic, the Set button can have 2 modes: Set and Enter Code.
  - Set will start the device. Code will let the user enter the secret code by pressing the digit buttons
  - M_mode . M_digit(c0) . M_digit(c1) . M_digit(c2) -> M_set, secretCode = c0+c1+c2
    - Replace M_set by M_mode in the transition table
- This is part of prototyping and trying things out!
  - Requirements can change!

Home Alarm System: Evolution of Requirements

- First attempt:
  - Used M_goodDigit and M_badDigit as inputs
- Second attempt:
  - Defining how to set up the alarm, separating the setting alarm and entering code modes, etc.
  - Refined the state table
- That's why we need various iterations of requirement gathering and specifications, as well as constant input of stakeholders
- *Who checks that our requirements (and our system) are good enough, and how do we prove it?*

Requirements



Who? Regulatory Environments

- In safety-critical domains, you just don't build and deploy, all systems are *regulated*
- One or more government agencies are mandated to act as *regulators* and are then empowered to *licence* or *approve* systems/devices before they can be deployed
- Typical domains:
  - **Nuclear power:** Canadian Nuclear Safety Commission (CNSC), US NRC.
  - **Civil aviation:** FAA (US), Transport Canada.
  - **Railways:** Transport Canada, FRA (US).
  - **Medical devices:** Health Canada, FDA (US).
- Legal accountability: if you violate regulations (e.g., deploy an uncertified system), your organization can face fines or legal action.

How? Standards

- There are many international standards that govern the development of software-dependent systems
- The three main international standards bodies in this regard are: *IEEE*, *ISO* and *IEC:*
  - IEEE: Institute of Electrical and Electronics Engineers
  - ISO: International Organization for Standardization
  - IEC: International Electrotechnical Commission
- They aren't laws, but they are **accepted best practices** that regulators often require compliance with.
- They were designed to serve as a foundation for achieving and maintaining quality products
- Two types:
  - **Process standards**: how to do development: lifecycle, documentation, testing
  - **Functional safety standards**: how to show that risk is reduced to acceptable levels

How? Standards

- Examples:
  - **IEC 61508** (umbrella standard for functional safety), **ISO 26262** (automotive), **IEC 62304** (medical devices), **DO-178C** (aviation software).
- **Safety Integrity Levels (SILs):** introduced by **IEC 61508**.
- SIL = a way of classifying *how safe* a function needs to be, based on risk.
  - SIL 1 = lowest integrity requirement.
  - SIL 4 = highest integrity requirement (rare, extreme risk situations).
- Different domains have their own variations (e.g., DAL in aviation, Class in medical devices).

Regulation vs. Standardization

- **Regulation:** mandatory, enforced by government/regulator, carries legal accountability.
- **Standards:** industry consensus on good practice, often required by regulators but not laws by themselves.
  - Typically, software standards are (too) process focused – and it is difficult to describe process in enough detail to ensure the outcome is what we need, eg safe!
  - Minimum requirement!
- Typically:
  - *Regulators say what you must achieve* (safe, secure, reliable).
  - *Standards say how you should go about it* (processes, SILs, documentation).
- Requirements engineering tells us how to write **clear specifications**.
- Regulation and standards tell us how to make sure those specifications, and the systems built from them, are **acceptable in safety-critical domains**.