COM4506/6506: Testing and Verification in Safety Critical Systems

Dr Ramsay Taylor

Contents

“The vast majority of accidents in which software was involved can be traced back to requirements flaws.”

–Nancy Leveson, Safeware: System Safety and Computers

What is a Requirement?

Three main categories:

- Basic function or objective.
- Constraints on operating conditions.
- Prioritised quality goals to support trade-off decisions.

User Requirements

“I want a website that’s really cool”

<weeks later>

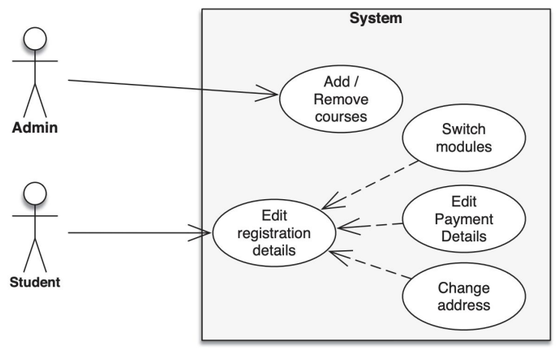
“Why isn’t this green!?”

- Humans are bad at really knowing what they want
- Humans don’t always articulate everything they think (“That’s common sense”)
- Human languages are *ambiguous* even when we do want to say something.

Requirements Capture

How we record requirements can help with their capture.

- User Stories
- Use Case diagrams
- Flow Charts
- State Machines
- Formal Methods (Z/CSP)



Requirements Capture

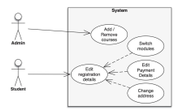
We can improve requirements with some better structured capture processes.

- Stakeholder Interviews
  - Captures the things they know they want
  - Misses things they don’t know about, or that they think are “obvious”
  - Misses requirements from people/things not identified as stakeholders
- Ethnography (*i.e. watching things happen)*
  - Works well for implicit or “common sense” requirements
  - Less useful for things that don’t exist yet
  - Hopefully doesn’t observe safety issues!

Requirements Capture

Requirements capture with clear documentation makes for better *iterative* processes.

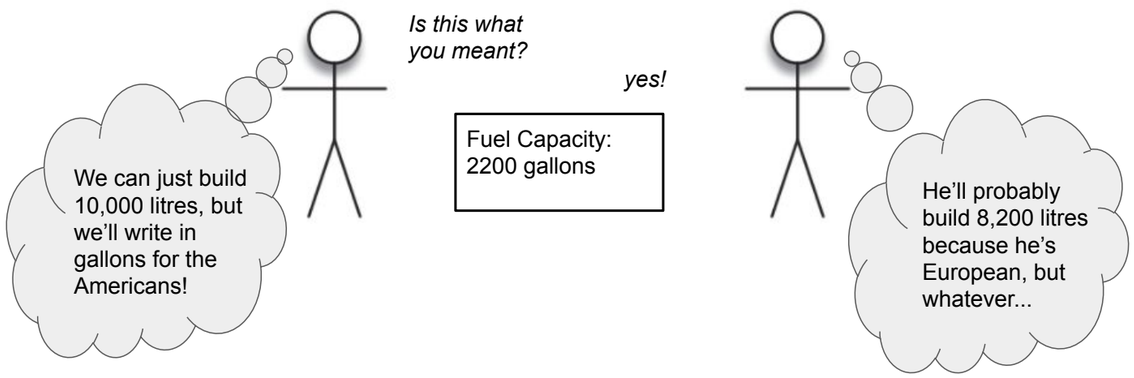
Is this what you meant?



Which bit will be green?

Requirements Capture

Iterative processes are also good for reducing ambiguity, but they don't remove it.

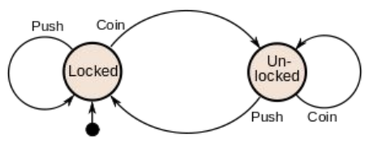


Completeness

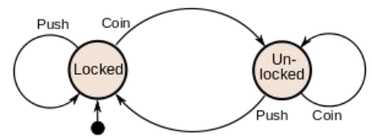
Various capture methods can prompt us to write down all of the things we know or think of.

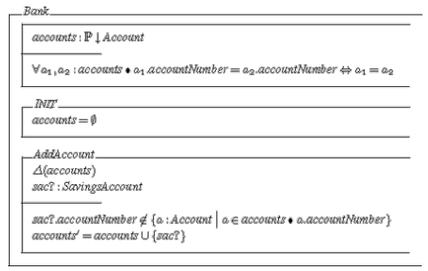
But what about the things we *didn't think of*?

We can use iterative requirements processes and structured documentation to measure and improve *completeness*.



Formal Languages





Bank

$accounts : \mathbb{P} \downarrow Account$

$\forall a_1, a_2 : accounts \bullet a_1.accountNumber = a_2.accountNumber \Leftrightarrow a_1 = a_2$

INIT'

$accounts = \emptyset$

AddAccount

$\Delta(accounts)$

$sac? : SavingsAccount$

$sac?.accountNumber \notin \{a : Account \mid a \in accounts \bullet a.accountNumber\}$

$accounts' = accounts \cup \{sac?\}$

UML and Flow Charts are *structured* but they don't have *Formally Defined Semantics*. This doesn't matter to normal people, but if its Safety Critical it can be significant!

The formal definitions remove (most) ambiguity.

Languages with *Formal Semantics* can support *Formal Verification* later.

Completeness

Formal languages have better defined notions of completeness, and can ask questions about what we've written - (e.g. Model Checking)

We can *define properties* in formal languages too, which makes them less ambiguous and easier to *Verify* later.

Name	Formal Statement
tP_1	$\texttt{NOT}(\texttt{cInDoor} \neq \texttt{closed} \wedge \texttt{cOutDoor} \neq \texttt{closed})$
P_2	$\texttt{cInDoor} \neq \texttt{cInDoor}' \Rightarrow \texttt{cChPres} = \texttt{InPres} \wedge \texttt{cChPres}' = \texttt{InPres}$
tP_3	$\texttt{cInDoor} \neq \texttt{closed} \Rightarrow \texttt{cChPres} = \texttt{InPres}$
P_4	$\texttt{cOutDoor} \neq \texttt{cOutDoor}' \Rightarrow \texttt{cChPres} = \texttt{OutPres} \wedge \texttt{cChPres}' = \texttt{OutPres}$
tP_5	$\texttt{cOutDoor} \neq \texttt{closed} \Rightarrow \texttt{cChPres} = \texttt{OutPres}$
P_6	$\texttt{cChPres} \neq \texttt{cChPres}' \Rightarrow \texttt{cInDoor} = \texttt{closed} \wedge \texttt{cOutDoor} = \texttt{closed}$

Summary

- User Requirements can be vague and ambiguous
- Structured capture processes, especially with structured languages can make this a bit better.
- Formal languages are even better for this, but you do have to work with Formal methods people!
- Completeness can be evaluated on structured languages, which can prompt you to think of things
- Defining formal properties will help later, but also helps with capturing things now.