

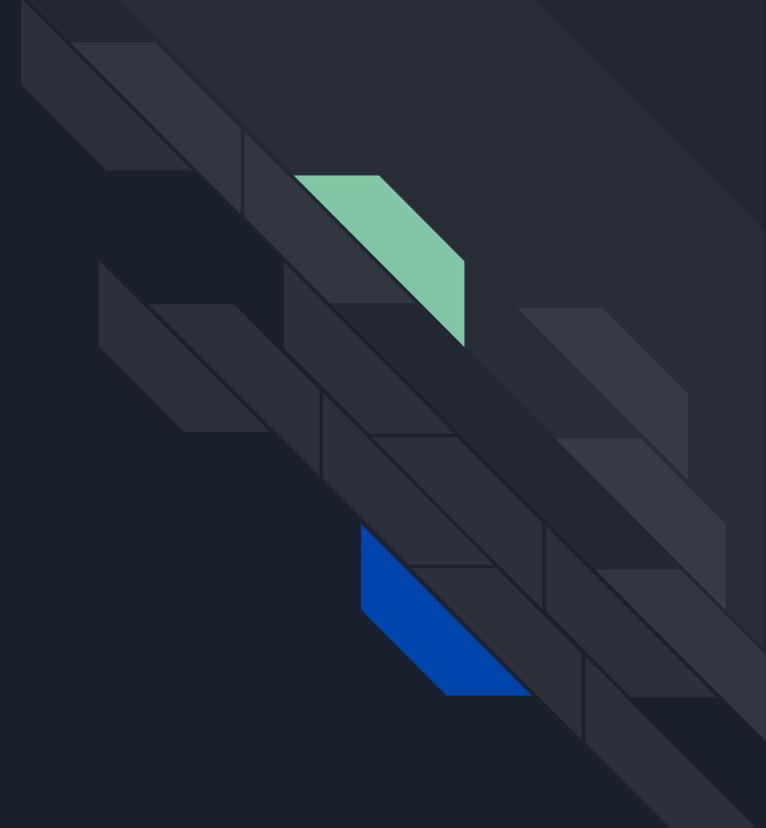


# Safely in time

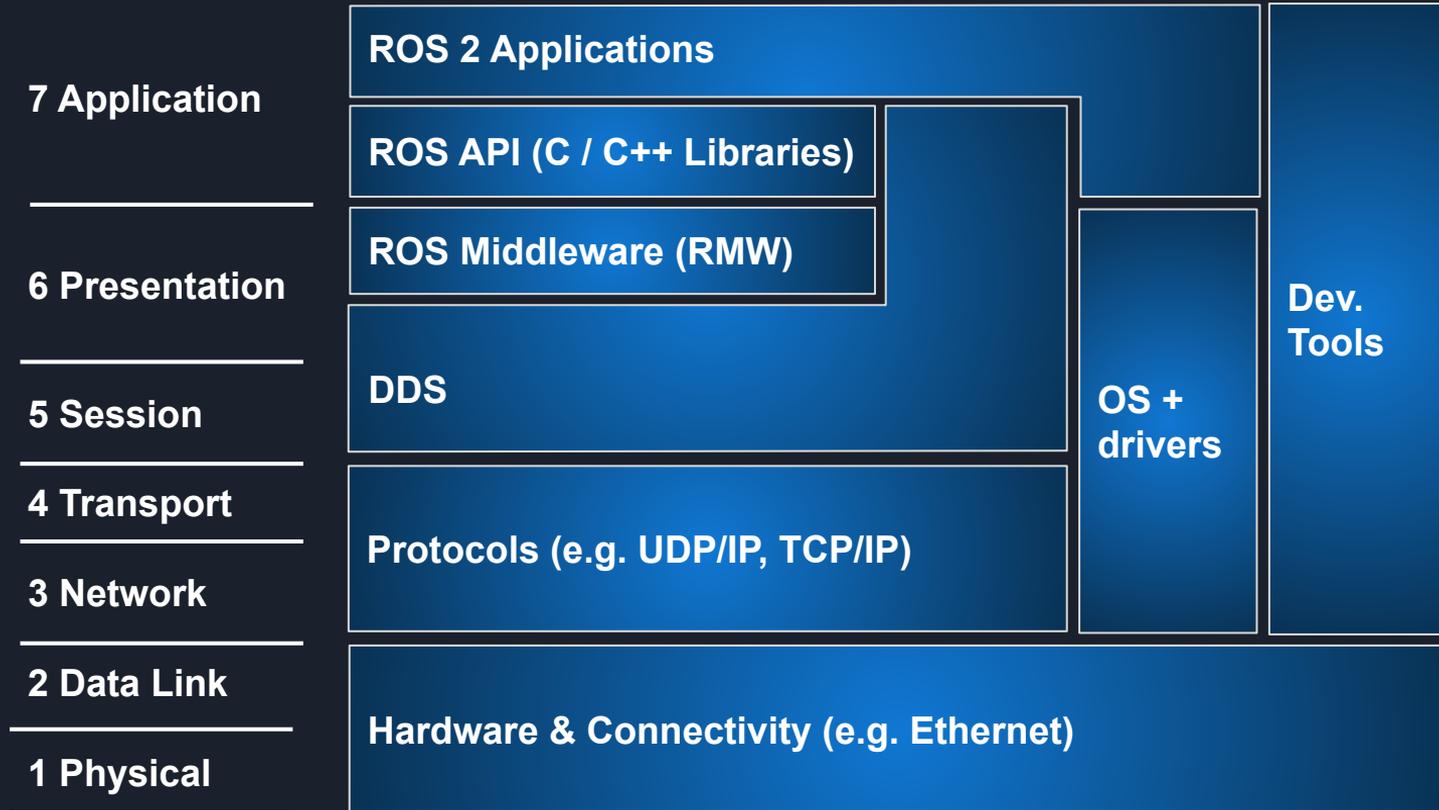
Real-time and safety-critical software  
development

Geoffrey Biggs  
Tier IV, Inc.

Warning: Incoming text



# What this is relevant to





# Safety-critical systems

Freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.

- IEC 61508 definition of “safety”



# Safety-critical systems

Freedom from **unacceptable risk** of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.

- IEC 61508 definition of “safety”



# Safety-critical systems

Safety-critical systems are not:

- “The manipulator will never stab someone with its tool.”
- “The autonomous car will never run someone over.”

Safety-critical systems are:

- “The manipulator will correctly stop motion within 100ms when it detects a person within its reachable space, with no more than one failure in 1,000,000 hours of operation.”
- “The autonomous car will miss-identify a pedestrian within the lane no more than once in every 10,000,000,000 detections.”



# What makes a system safety-critical?

In general, if there is a chance that it could cause harm in some way, it is safety-critical.

Is your mobile robot safety-critical?

- Does it operate near people?

- Is it large or heavy enough to cause injury?

- Is it a trip hazard? (Hello, Roomba!)

- How great is any risk presented?

What if your robot isn't moving?



# How is safety achieved?

## Passive safety

Safe by its nature, e.g. shielded battery contacts

We are not particularly concerned about this approach

## Functional safety

Must function correctly at the correct time or within the correct time period to be safe

This is what interests us



# How is safety achieved?

## Passive safety

Safe by its nature, e.g. shielded battery contacts

We are not particularly concerned about this approach

## Functional safety

Must function correctly at the correct time or within the correct time period to be safe

This is what interests us



# Safety requirements and real-time

Safety requirements typically have a real time aspect.

REQ042: The brakes must be applied **within 100ms** to reduce the severity of harm caused to a pedestrian below an unacceptable level

Safety requirements often drive other real-time requirements

REQ042.1: The sense-plan-act cycle must **operate at 100ms or faster** to provide sufficient reaction time

Sometimes a real-time failure can lead to a safety failure later on



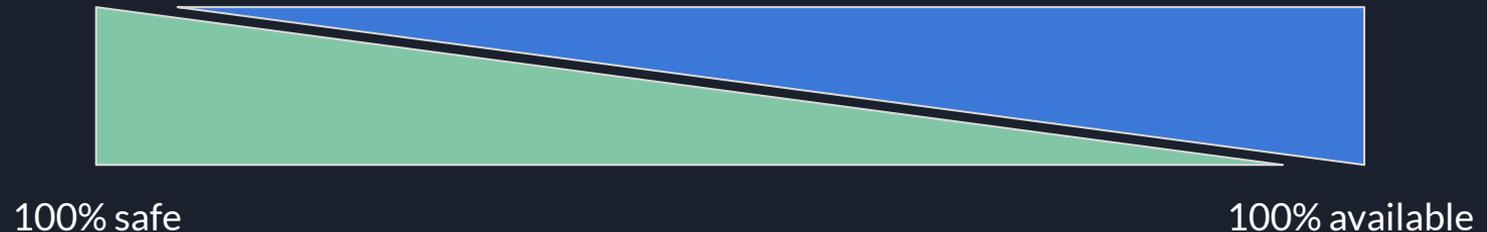
# Safety requirements and real-time

Not all real-time requirements are safety-critical

REQ043: The manipulator must move the spoon from the bowl to the eating position within five seconds

This relates to the balance between **availability** and **safety**

If not meeting the deadline is merely an annoyance, then all that will happen is no one will buy your robot.





# Finding the safety-critical real-time requirements

Not always obvious

The easy ones

If violating the real-time deadline means harm is caused, then it is safety-critical



# Finding the safety-critical real-time requirements

Not always obvious

The easy ones

If violating the real-time deadline means harm is caused, then it is safety-critical

The requirements that support those are also probably safety-critical, e.g. motor control loops



# Finding the safety-critical real-time requirements

The hard ones are very application- and design-specific

How is behaviour affected if control signals are late?

What if all control signals are within the deadline, but still irregular?

A robot with a long mission span may need rejuvenation periodically

How much time is available for rejuvenation?

How much harm is caused if rejuvenation takes too long?

Is that harm greater or less than the harm caused by not rejuvenating?

(Can you acquire your resources in deterministic time?)



# Specifying real-time requirements

Guaranteeing deadlines is typically impossible on most modern hardware

e.g. Large-scale x86 and ARM CPUs

Guarantees are broken by the complex nature of instruction execution

Out-of-order execution, branch prediction, etc.

Real-time guarantees instead must be given as probabilities

e.g. “Probability of missing the deadline is  $10^{-6}$  per hour of operation”



What do you do when you have found your safety-critical real-time requirements?

You must:

- Develop a system that meets those requirements

- Prove to the necessary level of confidence that your system meets those requirements

This means following a safety-critical development process



# The four steps of safety-critical software development

1. Design system behaviour to satisfy the requirement
2. Verify that the design will satisfy the requirement
3. Implement software that meets the design
4. Verify that the software does meet the design



# Design system behaviour to satisfy the requirement

When doing the design, consider factors of the design that will impact execution time

- Longest execution path time

- Whether each execution cycle is identical in execution time or not

- How error cases impact deadline achievement

- Time required to recover from an error (or to achieve a safe state)

Also consider factors that will impact whether an action that must be performed by a deadline can be performed at all

Don't forget to design a system that knows if it is failing a deadline



## Verify that the design will satisfy the requirement

You must provide evidence to back up this claim

Timing diagrams, process scheduling plans, etc.

Formal proofs are particularly useful for real-time claims

Linear Temporal Logic can help prove the liveness aspects of algorithms

Tools such as TLA+ can be used to show there are no deadlocks

Don't forget to consider undesired conditions

Do failure analyses (FMEA, FTA, etc.) to know when something might break and impact a deadline



# Implement software that meets the design

In general, follow software engineering best-practices.

- Follow a coding style

- Test your code properly (not just unit tests)

- Use static analysis tools

In particular relevant to coding practices that may impact real-time performance

- e.g. Memory allocations in the wrong places - there are tools that can find these but you need style rules as well



# Verify that the software does meet the design

Understand what you are testing against

- The hazards, not the safety requirements derived from them

Use fault injection to verify real-time behaviour, real-time failure, and the impact of error paths on meeting deadlines

- Inject random delays to test deadline failures

- Inject delays in communication, execution, memory access, ...

Verify against models of the system that can simulate execution time and process scheduling

All verification results need to specify a confidence level



# Know your tools

Many tools can impact the achievement of deadlines, and not all are obvious

## Programming languages

Does your programming language have the necessary constructs for real time?

How easy is it to create priority inversions?

## Compilers

Is the timing of constructs predictable?

What impact does optimisation have on execution time?



# Know your tools

## Tools for measuring execution

Are the tools you use accurate?

How do you know that? Have you verified it yourself?

## Tools for verifying execution

Does your memory analyser work correctly?

Does your test environment accurately match the production environment?

## Tools for generating implementations (e.g. Simulink)

How do you know they produce the correct code?



# Know your tools

How do you know you can trust your compiler?



# Know your tools

How do you know you can trust your compiler?

Hint: You can't.

“Every compiler we tested was found to crash and also to silently generate wrong code when presented with valid input.”

- Yang *et al.*, “Finding and understanding bugs in C compilers”, ACM SIGPLAN 2011



# Summary

If your robot not reacting in time means it might squash the cat, you have to think about safety.



# Summary

If your robot not reacting in time means it might squash the cat, you have to think about safety.

If you don't want your robot to squash the cat, it must be able to react in a timely manner.



## Summary

If your robot not reacting in time means it might squash the cat, you have to think about safety.

If you don't want your robot to squash the cat, it must be able to react in a timely manner.

If you want to show that it (probably) won't squash the cat, you must develop such that you can prove with sufficient confidence that it will react in a timely manner.



# Summary

If your robot not reacting in time means it might squash the cat, you have to think about safety.

If you don't want your robot to squash the cat, it must be able to react in a timely manner.

If you want to show that it (probably) won't squash the cat, you must develop such that you can prove with sufficient confidence that it will react in a timely manner.

This is safety-critical development of real-time software.