# Requirements for self-driving guidance software: *A discussion of alternatives*

Moderator

**David** Gelperin, CTO
ClearSpecs Enterprises

# Goals for the discussion

- **To identify major hazards for requirements quality in the current competitive approach**
- **To explore safer strategies and tactics**
- **To identify tactics for raising awareness of the need for safer approaches among manufactures and others**

# Proposed Agenda

1. **Moderator task description – *promote participation***
   - ***Share knowledge, experience, & opinions you brought***
   - ***Share reactions to what I brought***
   - ***Share your thoughts and those of colleagues afterwards at david@clearspecs.com***
2. **Quick attendee intros: *Name, Organization, Responsibilities and Interests***
3. **Agenda modifications**
4. **Position statements (3-5 minutes) on current approach to autonomous software**
5. **Discuss (and record) hazards in and mitigations for current approaches**
6. **Discuss (and record) safer approaches and recommendations**
7. **Identify (and record) areas of disagreement**
8. **Discuss (and record) tactics for raising awareness in various communities**

# Why I'm Here – 1 of 2

# Dave's Goal

**To share ideas that raise your awareness of extremely dangerous requirements practices for potentially pervasive safety-critical software**

**I hope that each of you will spread these ideas until they influence the developers of this software**

# Autonomous vehicles may have significant benefits

- May save many lives
- Many increase freedom of movement for physically challenged
- .

**Are an admirable objective**

# Innocents will die

**Group 0 – Wrong place, wrong time**

**Group 1 – System defects**

**Group 1a – Best efforts, but human error**

**Group 1b – Not best efforts e.g., poor reqts**

Current situation will significantly increase Group 1b due to **deeply flawed development strategies** for **extremely complex software**

**Competitive and unmonitored development strategies will kill many more than necessary**

# Competitive requirements are a very bad idea

**I assume safe software entails excellent requirements**

**Competitive artifacts will be seriously flawed**
- **Glossaries**
- **Hazard analysis results**
- **Quality attributes e.g., safety, security, …**
- **Function sets**
  - **particularly hazard mitigation functions**
  - **e.g., all behavior must be monitored**
- **Design constraints**
  - **e.g., no single points of failure**
- **Implementation constraints**
  - **e.g., must have and monitor coding standards for clear and safe software**

# Hazards and Mitigations

**Review candidate hazards and mitigations for requirements quality and public safety**

**Modify and add others**

- e.g., **No experts in the safety of AI applications**
- e.g., **Worst will be first i.e., max defects, no support**

Purpose:

***To identify current, but unnecessary, hazards to self-driving guidance (SDG) software and suggest mitigations.***

Scope:

**Safety of the guidance software**.

Neither the sensor processing software nor the effector control software is included in this initiative.

**Candidate hazards** for requirements quality and public safety:

1. SDG software will be at least an order of magnitude more complex than any embedded automotive software in production use today.
2. As is well known to [some] software engineers (but not to the general public), by far **the largest class of problems** arises from errors made in the eliciting, recording, and analysis of requirements. [2007 National Research Council report on **Software for Dependable Systems**, page 40]
3. Most US software engineers have never taken a requirements course. Only four of the "top 25" software engineering colleges in the US teach such a course. It is likely that most embedded software developers, some of whom are EEs and system engineers, have never taken a requirements course either.
4. Most software engineers, system engineers, and embedded software developers do not understand the nature of software quality requirements i.e., requirements for safety, security, reliability, and understandability. They do not know how to specify, achieve, nor verify them.
5. In the US, regulators of autonomous vehicles (AVs) know less about software requirements than software developers.
6. There are more than 40 companies worldwide "racing" to create components and vehicle guidance software with a poor understanding of software requirements in general and software quality requirements in particular.
7. While the potential benefits of AVs are very desirable (i.e., the ends), risks in the current competitive development approach (i.e., the means) are significant and unnecessary.
8. When manual tasks are automated by a team, a detailed glossary is essential. Otherwise, something worse than a "tower of babel" is created within the team. The same words will have many different meanings. For example, the meaning of "erratic driving of the preceding vehicle" must be defined in detail along with actions to take when detected.
9. Having more than one (1) glossary, (2) set of hazard analysis results, and (3) set of basic SDG requirements is dangerous.
10. Competitive requirements suggest unsafe cultures.
11. An industry-supported edge-case simulator for software verification will also increase safety.

12. People will die needlessly, especially early adopters, without industry-consensus work-products because significant confusion will be added to the significant complexity of this task.
13. The volume of early deaths may endanger acceptance of the technology and thus its benefits would be lost or delayed due to risky development tactics.

**Candidate mitigations**:

1. Encourage industry to cooperate in the development of industry-consensus, "open-source", requirements information including a glossary, hazard analysis, and specification of basic SDG requirements.
2. Develop a standard on "meta-requirements for critical software requirements".
3. Encourage regulators to require compliance with the meta-requirements standard.
4. Encourage quality-aware development.
5. Encourage specification of quality attribute requirements using a tailored 3D quality model.
6. Encourage staffing of a Quality and Productivity function (part 3 of Quality Goals chapter).
7. Encourage development and use of an industry-supported edge-case simulator for SDG software verification.

**Note**: Details of mitigations 4-6 can be found at **www.quality-aware.com**

# More Potential Mitigations

**Open requirements artifacts**

**Intentionally imprecise requirements**

**Meta-requirements and monitoring**

**(SEI) software architecture evaluation methods**

**(CISQ/OMG) code quality standards and monitoring**

**(Misra) coding standards and monitoring**

# What is “erratic driving”?

# Is this creature “near the roadway”?

# Glossaries are critical

**Tower of Babel**

**Tower of Unrecognized Ambiguity**

**For new domains or applications**, terminology management is very important

# System Hazards

**Review system hazards and glossary definitions**

**Modify and add others**

# Developing requirement artifacts from identified hazards

Identify:

- potential system hazards that must be detected and responded to
- glossary definitions for hazard or response terminology e.g. driving erratically

Various forms of hazard analysis might identify:

**System failures**

- all/some sensors fail
- classifier fails e.g., misclassifies
- effector logic fails
- cpu with guidance software dies
- GPS signal lost
- software fails e.g., causes unanticipated acceleration (UA) or erratic behavior
- software enables hacking
- software fails to adapt to new environments e.g. passage from England to France or entry into school zone

**Vehicle conditions**

- motor dies
- tires go flat
- brakes fail
- battery fails
- vehicle catches fire
- top is sheared off
- vehicle skids (e.g., on ice)
- vehicle is submerged

**Roadway conditions**

- weather conditions
- different zones e.g., school, hospital, or work
- large, heavy, stationary object (e.g., firetruck) blocking the roadway or lane
- roadway collapses (e.g. bridge or sink hole)
- kangaroo on or near roadway
- traffic light outages
- stale yellow light

**Traffic conditions**

- wrong way driver
- car ahead/beside drives erratically
- car behind tailgates

Such hazard lists can be used to guide requirements development or check the completeness of existing requirements. To guide requirements development, each hazard including some system failures, can be put into the following template and then factored into specific situations.

> If <hazard>, then the system must safely respond.

For example:

> If the “motor dies” and the vehicle is stopped, then …
>
> If the “motor dies” and the vehicle is moving and safe stopping is feasible, then …
>
> If the “motor dies” and the vehicle is moving and safe stopping in not feasible, then …

# Intentionally Imprecise Specs

**Intentionally imprecise** requirement specifications state complete, necessary, possibly achievable, and subjectively verifiable restrictions on an implementation.

**Subjective verification** entails agreement by a "sufficient majority" of the stakeholders, where the composition of the "sufficient majority" is precisely defined.

For example:

> Any autonomous vehicle approved for use outside Australian cities must "recognize kangaroos" on or *near* the roadway and *take proper action*.

# Another example of Intentional Imprecision


Develop an elevator control system for tall buildings.

When fire is detected, the control system must **help to safely evacuate as many people as possible**.

It must also **enable responders to safely reach the fire and endangered locations as quickly as possible**.

# Building imprecise requirements from hazards

If the **motor dies on a roadway** and the vehicle is moving
and a *safe and traversable glide path is available*,

then turn on the hazard lights, signal appropriately, steer onto the glide path, decrease speed as needed, and brake when stopping is safe.

# What's an autonomous vehicle to do?

# Meta-requirements

**Review draft specification and suggest modifications**

**Meta-requirements for critical software requirements**

A straw-draft specification

Definitions:

1. A **mission statement** specifies the goals of a mission.
2. **High-risk missions** may result in significant financial loss and/or serious injury or death.
3. **Critical software** supports the goals of high-risk missions.
4. The **general public** are those not directly involved in a mission.
5. **HRGP missions** are those whose failures are likely to endanger the general public e.g., self-driving vehicles.
6. A **quality goal** is a quality attribute requirement e.g., a security requirement.
7. **Basic qualities** (as described in the LiteRM quality model – www.quality-aware.com/daves-q-a-stuff.php) are internal qualities including understandability, verifiability, and compliance.
8. We consider the following **types of software requirements**:

| Types and subtypes of software requirements | Primary Sources | Understanding Risk |
|---|---|---|
| Functions | | |
| Domain functions | | |
| interactive | | |
| happy paths | Customers | Medium |
| unhappy paths | Developers | Medium |
| autonomous | Developers | Low |
| System functions e.g., backup | Developers | Low |
| Quality support functions | | |
| external quality supports e.g., exception handlers | Developers | Medium |
| mixed quality supports e.g., encrypting routines | Developers | High |
| Quality goals (i.e., quality attribute requirements)[1] | | |
| Internal qualities e.g., coding standards compliance | Quality Management | Low |
| External qualities e.g., reliability | Customers & Developers | Medium |
| Mixed qualities e.g., security | Customers & Developers | High |
| Constraints | | |
| Technical | | |
| design e.g., no single point of failure | Developers | Medium |
| implementation e.g., coding standards | Quality Management | Low |
| verification e.g., test coverage | Quality Management | Low |
| deployment e.g., secure packaging | Developers | Low |
| Societal e.g., regulations | Quality Management | Low |
| Project e.g., deadlines | Project Management | Low |
| Supplier attributes | Quality Management | Low |

9. Each requirement has a set of **common properties** such as an identifier, type, primary sources, state, and priority.
10. **Technical inspections are skeptical** if inspectors assume the work is defective and diligently search for the defects.

[1] Details about quality goals can be found in Chapter 3 of **Understanding Requirements** and in the LiteRM 3D quality model. Both are freely available at www.quality-aware.com/daves-q-a-stuff.php.

11. **Meta-requirements** are restrictions on a requirement or set of requirements.
12.

Assumptions:

1. We don't fully understand the complex software we are building.
2. Effective development of complex software requires humility and an emphasis on understandability.
3. When software endangers the general public, our loved ones deserve our best efforts.
4. "As is well known to [some] software engineers (but not to the general public), by far the largest class of problems arises from errors made in the eliciting, recording, and analysis of requirements."[2]
5. Quality goals are poorly understood by developers.
6. Legacy norms (i.e., that's the way we have always done it) can be dangerous when dealing with significantly increased complexity.
7. The rationale for each meta-requirement is to maximize understanding and/or minimize defects in software requirements.
8. The rationale for the set of meta-requirements is to prevent unnecessary harm.
9.

Meta-requirements:

1. Software requirements must be organized by types and subtypes.
2. Each requirement must have:
    a. an identifier showing its type or subtype
    b. properties identifying priority, status, sources, and associated requirements
    c.
3. Functional requirements must be specified by rules specifying trigger conditions and associated actions.
4. Trigger conditions must be skeptically inspected to assure that each condition is neither too broad nor too narrow and is effective in capturing intent. Each action must be skeptically inspected to assure that it is safe and effective when its trigger conditions exist.
5. Trigger conditions for a set of functional requirements must be analyzed and skeptically inspected for correctness, feasibility, completeness, consistency, and necessity.
6. All basic qualities must be required (and aggressively verified).
7. The production and logging of assurance evidence during operation (e.g., by self-checking) must be required.

[2] Daniel Jackson, Martyn Thomas, and Lynette I. Millett, Editors
**Software for Dependable Systems: Sufficient Evidence?**
National Research Council 2007 [page 40]

8. The properties of a quality goal must specify its
    a. definition
    b. required level of achievement
    c. measurement strategy
    d. hazards
    e. mitigations
    f. achievement strategy
    g. verification strategy
    h. relationships to quality functions and other quality goals
    i.
9. The set of requirements must have:
    a. A r**equirements glossary or ontology** to precisely define conditions, actions, states, and any other words, phrases, acronyms, and abbreviations used in requirements statements. Actions must be defined by constant, pre, and post conditions. The glossary entries must be complete and consistent. The glossary must be configuration managed and readily accessible to any stakeholder.
    b. A set of user type specifications, if the software is interactive
    c. A set of domain object specifications
    d. A set of domain facts and assumptions
    e. A set of facts about values, conditions, and relationships
    f. A set of associated regulations
    g.
10. Each requirement must be discrete, understandable, unambiguous, valid, necessary, complete, concise, conformant, feasible, and verifiable.
11. The set of requirements must be task-adequate, concise, consistent with each other, prioritized, and achievable.
12. The requirements and their associated information must be skeptically inspected.
13. The set of requirements and their associated information for critical software supporting HRGP missions must be **made freely available to the general public** no later than the start of software design based on these requirements.
14.

# Unintended Acceleration Again

**'My gas pedal is stuck' BMW driver tells 911 as he barrels down interstate at nearly 100 mph**
**Feb 14, 2018**

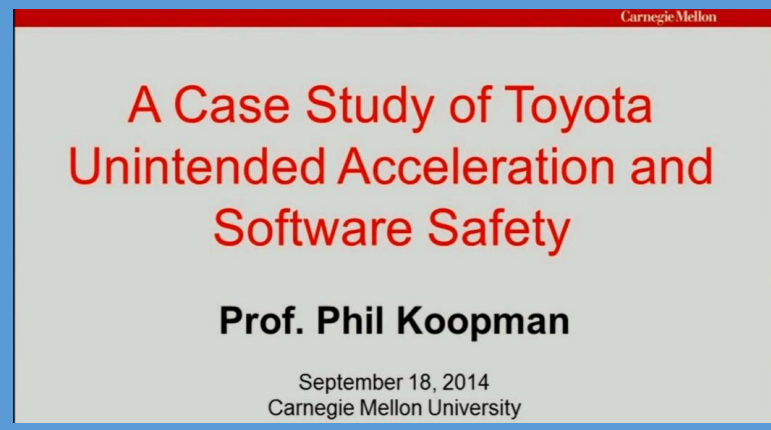
**BMW spokesperson called the scenario "implausible"**

"All BMW vehicles, .., employ an electronic accelerator pedal which uses software logic **to override the accelerator whenever the brake pedal is pressed while driving**. This fail-safe software means that if the vehicle detects that both pedals are depressed, the on-board electronics will reduce engine power so that the driver may stop safely.

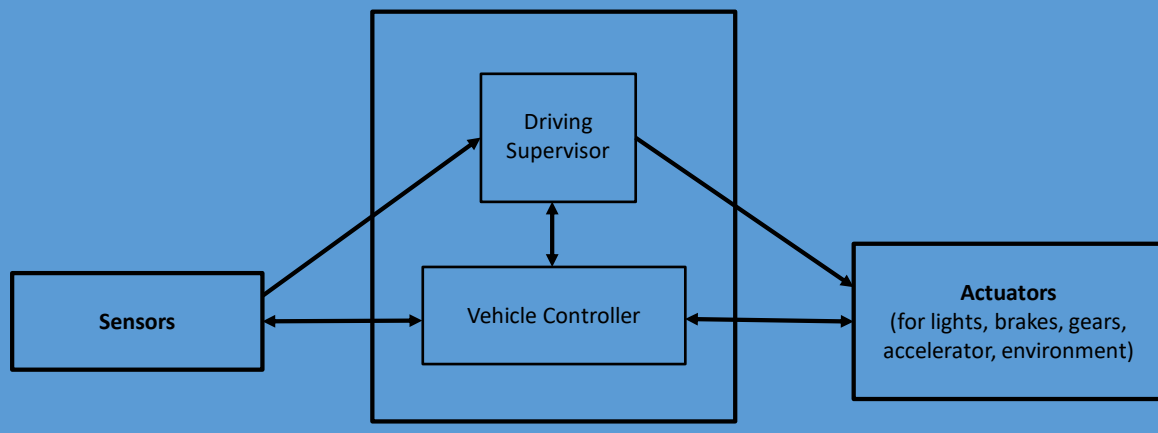
**No vehicle behavior monitoring requirement**

# Unintended Acceleration (UA)

Reports of UA
- Toyota
- BMW
- Audi
- Ford



# Software architecture for vehicle guidance




Driving **Supervisor** and Vehicle **Controller** both include
emergency parker, emergency stopper and logger
(no single point of software failure)

14

# Raising Awareness – Directly & Indirectly

- **manufacturers**
- **regulators**
- **liability insurers**
- **public interest groups**
- **press**

- **RE community**
- **software safety community**
- **software quality community**
- **lawyers**
- **others?**

**If you're NOT part of the solution, ...**

# Final Thought

**Hope is not a tactic**

Steve Jablonsky,
Deepwater Horizon