Five Defects in NHTSA's Federal Policy on Automated Vehicles (Sept. 2016)

David Gelperin
david@clearspecs.com

This note identifies five significant defects in the Vehicle Performance Guidance of the recent HAV policy (https://www.transportation.gov/sites/dot.gov/files/docs/AV%20policy%20guidance%20PDF.pdf) and suggests changes.

Defect 1. The policy's **vague description of a framework** for developing and verifying complex, safety-critical software **is inadequate**.

Effective development and verification of complex software that is NOT safety-critical is a hard job. Adding "safety-critical" makes the job much harder. If AI is involved, it's harder still.

So far, the automotive industry has developed the following capabilities:

- GPS-guided navigation
- Self-parking
- Adaptive cruise control
- Intelligent speed adaptation
- Automatic high-beam control
- Remote vehicle shutdown
- Accident response system
- Vehicle and roadway monitoring, warning, and mitigation
    - Forward-collision warning with automatic braking
    - Reverse-collision warning with automatic braking
    - Brake assistance
    - Curve speed warning
    - Left-turn crash avoidance
    - Pedestrian detection with night vision
    - Cross-path detection in rear-mounted radar
    - Blind-spot and cross-path warning
    - Lane-departure warning and prevention
    - Roll-over avoidance
    - Temperature warning
    - Tire pressure, fuel level, and battery monitoring

None of these capabilities is complex, from a software perspective.

Integrating these capabilities and new capabilities that deal with the situations listed on pages 28 and 29 as well as:

- Changing lanes in heavy traffic
- Aggressive behavior by other drivers (e.g. not yielding)
- Avoiding unexpected flying or roadway objects e.g. falling rocks
- Emergency parking with few or no safe parking spots
- Dealing with poor control
    - Poor traction e.g. ice, snow, oil
    - Poor stability
    - Driving into the setting sun (on a cloudless day)
    - Poor visibility e.g. dust storm, very heavy rain, thick fog, or whiteout
    - Poor power e.g. engine stopping in traffic

    - Poor braking
- Dealing with deep water on the roadway
- Stopping for school buses
- Dealing with failure
    - Communication
    - Sensor
    - Image classification

is **very complex from a software perspective**.

The likelihood of accidents involving HAVs is significantly increased by the fact that neither automobile manufacturers (AMs) nor NHTSA have experience developing, verifying, and regulating software of this complexity. Vague references to standards and best practices won't help the inexperienced.

The safest way to manage the risk of this unfamiliar task is to clearly define a set of disciplined activities and their associated documentation and then require compliance. This can be achieved (1) by requiring compliance with ISO 26262, MIL STD 882E, RTCA DO-178C, and the MISRA coding standards or (2) by developing a modified version of the proposed process model in "Safe Software Acquisition in a Nutshell" (appendix).

Defect 2. The policy leaves each AM and NHTSA to develop **separate** problem definitions.

The first two steps of safety engineering entail identifying safety goals, harmful events, and hazards. These activities result in a clear problem definition. These activities do not develop solutions. There are no IP concerns.

It does not serve the public interest (nor promote HAVs) to have AMs or NHTSA overlook a hazard that results in "needless" crashes. NHTSA must perform hazard analysis to develop effective tests.

Google overlooked a "failure to yield" hazard resulting in a low-impact collision. What else may be overlooked?

The solution is to have NHTSA lead development of a single clear problem definition. NHTSA should lead a consensus process, including AMs, to define safety goals and identify harmful events and hazards. A curated problem definition can improve with experience as well as providing the most comprehensive initial definition using public comment.

Defect 3. The "Safety Assessment letter" is **dangerously incomplete**.

a. There are no references to safety goals, harmful events, or hazards. If NHTSA doesn't address "Defect 2" as suggested, it will need to perform its own HA and then compare it to the HA results of each AM.

   Each "safeguard" must be linked to its associated hazards. Each test of a safeguard must be linked to it associated safeguard.

b. There are no references to most quality attributes that support software safety. Figure 1 (appendix) claims there are up to 39. The assessment letter only references 3.

   After identifying the relevant supporting attributes, measurable goals must be defined and achievement tactics identified, implemented, and verified. For example, comprehensive coding standards must be acquired (e.g. MISRA standards) and compliance verified using automated analysis and code reviews to assess "understandability".

c. There are no references to several critical verification methods. The policy only references test and simulation. Neither method can directly verify "internal quality attributes" – see all attributes in the Basic quality group of Figure 1 (appendix).

   Technical reviews, measurement, and various forms of analysis are not mentioned as part of a comprehensive verification strategy. As software complexity increases (in response to the increasing number of situations that must be handled), the verification effectiveness of test decreases. Technical reviews including inspections (Fagan or High-Impact), Planguage-style measurement, and automated and manual analysis are also required.

   References to compliance reports for coding standards should be included.

d. There is no mention of a **code coverage criteria** for AM testing, nor reporting of its achievement.

The content of the "Safety Assessment letter" needs to be expanded with the items identified above. The term "verification" should replace "test" in most of the policy.

Defect 4. **Self-certification** of complex, safety-critical software by developers inexperienced with such software **is dangerous.**

Self-certification (pages 41, 42, 71, 72, 74) by the inexperienced is a dangerous strategy. Most developers have a poor understanding of software quality attributes and software quality verification. Toyota's unintended acceleration problem illustrates this danger.

NHTSA is a surrogate "vendor manager". In the era of complex, safety-critical, embedded software, NHTSA must use best industry practice to assess the quality of each AM's embedded software. At a minimum, this entails measurement of compliance with MISRA coding standards, cyclomatic complexity requirements, and code coverage requirements. This assessment has no IP issues and will detect gross code quality defects.

NHTSA can assist learning by minimum participation in development reviews and by providing process assistance when necessary. For example, NHTSA should witness inspections of requirements specs and verification strategies.

Defect 5. **Vague terminology is dangerous**.

How should "reasonably safe under real-world conditions" (page 11) be measured?

How should "unreasonable safety risks" (page 20) be identified?

Who or what are the authorities for "industry best practice" (page 11), cybersecurity best practices (page 13), HMI design best practices (page 13), and "established best practices" (page 21)?

Are "best practices" and "**best practices for complex, safety-critical systems**" the same?

What if a "minimal risk condition" (page 14) can't be safely created? For example, what if a safely parked state can't be attained or can't be safely attained?

Appendix

Safe Software Acquisition in a Nutshell

Map of High-level Activities and Results of Mixed Software Acquisition

Phased Activities	Phase Results	Continuous Activities			
		6 Verification	*7 Software Quality Support*	*8 Configuration Mgmt*	*9 Project Mgmt*
1 Project Planning	Acquisition strategy and project plan				
2 SW Reqts	Reqts specs and verification strategy				
3 SW Acquisition	Capabilities, Characteristics, and Test suites				
4 SW Acceptance	Acceptance reports and release strategy				
5 SW Release	Release reports				

Notes –

1. **Acquisition** means "to come to have in an unspecified manner". Software can be acquired via Software as a Service (SaaS), Commercial Off The Shelve (COTS), outsourcing, open sourcing, reuse, refactoring, custom development, or a mixture.
2. An **acquisition strategy** includes project-specific verification, quality support, configuration mgmt, and project mgmt strategies**.**
3. **Phased activities may be repeated and phase results changed** because of learning or redirection.
4. **Software requirements** should include domain functions, quality goals, constraints, and supplier characteristics.
5. **Verification** of phase results entails various forms of technical review, analysis, measurement, testing, and operational monitoring.
6. **Software quality support** is not a testing functions, but refers to support for all project activities and entails acquisition and maintenance of standards and guidelines for requirements, verification, design, and coding, of tools (e.g. code compliance analyzer and code coverage analyzer), and of skills training, as well as project mentoring and participation in verification activities.

Unnecessary defects are defects that are easy to prevent or easy to detect e.g. by measuring test coverage, by using mutation analysis, or by running a coding standards analyzer.

Effective formality implies less ambiguous and less defective results at a higher cost. Effective formality should prevent or remove all unnecessary defects. The challenge is to balance the risk of defective software against the cost of increased formality.

Assumptions

1. **Predictable software safety** requires (1) a safety culture and (2) a safety-oriented acquisition process.
2. Effective acquisition methodologies should require a minimal number of "just enough" and "just in time" activities and results. They should describe optional activities and results to include when failure risk is higher than normal.
3. For software safety, all acquisition activities in the model above are necessary to some extent and can be carried out by individuals or groups in a range from informal to formal. For example, planning can take hours, days, or weeks.
4. The failure risk associated with safety-critical (or mission-critical) functionality and quality goals (e.g. security goals) justifies a "significant amount" of effective formality. This amount should be defined by professionals in system safety.

The acquisition process model above can be used to organize the major documentation items of RTCA DO-178C and ISO-26262.

RTCA DO-178C **Scope:** Safety-Critical SW	**ISO-26262** **Scope:** Functional Safety	**Proposed SW Safety Process** **Scope:** Safety-Critical SW
1. Project Planning	1. Project Planning	1. Project Planning
a. SW development plan	a. SW elements in the safety plan	a. SW acquisition strategy including project-specific quality mgmt, safety assurance, configuration mgmt, and project mgmt strategies
b. SW QA plan	b. SW design guidelines	b. Project plan
c. SW CM plan	c. SW coding guidelines	c. Requirements guidelines
d. SW verification plan		d. Verification standards
e. Plan for SW aspects of certification		e. Design guidelines
f. Requirements standards		f. Coding standards
g. Verification standards		
h. Design standards		
i. Coding standards		
j. Project mgmt plan		
k. Reqts mgmt. plan		
l. Test plan		
2. Software Requirements	2. Software Requirements	2. Software Requirements
a. SW requirements spec	a. SW elements in safety goals	a. List of harmful outcomes
b. SW interface specs	b. SW elements in hazard analysis results	b. Hazard and Fault tree analysis results
	c. SW safety reqts	c. SW hazard mitigation strategy
	d. SW interface specs	d. SW Reqts specs including all safety-related reqts
	e. SW verification plan	e. Verification strategy

3. Software Acquisition	3. Software Acquisition	3. Software Acquisition
a. SW design descriptions	a. SW design descriptions	a. Acquired functionality, and characteristics
b. Code and data sets	b. Code and data sets	b. Test suites
c. Test suites	c. Test suites	
	d. Safety case	
4. Software Acceptance	4. Software Acceptance	4. Software Acceptance
a. Acceptance test reports	a. SW Safety verification report	a. Acceptance verification reports
		b. Release strategy
5. Software Release	5. Software Release	5. Software Release
NA	NA	a. Release report
6. Verification	6. Verification	6. Verification
a. Verification reports	a. Verification reports	a. Verification reports
7. Software Quality Assurance	7. Software Quality Mgmt	7. Software Quality Mgmt
a. SQA report		b. SQM report
8. Configuration Mgmt	8. Configuration Mgmt	8. Configuration Mgmt
a. Configuration and control logs		a. Configuration and control logs
9. Project Mgmt	9. Project Mgmt	9. Project Mgmt
a. PM report		b. PM report

Observations

1. 178C has too much planning too early. Need to find a balance between baroque planning and no planning
2. 26262 doesn't focus on software safety.
3. Task-adequate software requirements are essential to safety. Neither guidance document adequately deals with quality goals including safety (see Figure 1 below).
4. Both guidance documents focus only on custom development.
5. Neither guidance document mentions verification and reporting of compliance with design or coding standards/guidelines.

Figure 1: The 39 quality attributes that may support safety

Not only do measurable safety goals need to be specified and verified, but measurable goals for supporting qualities do too.