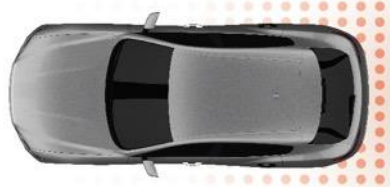
foretellix

Spec bugs

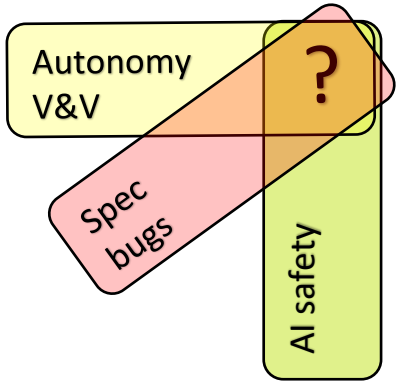
Lessons from V&V of AI-based autonomy



Yoav Hollander – 2-Oct-2025

Can autonomy-grade V&V make AI safer?

Introduction

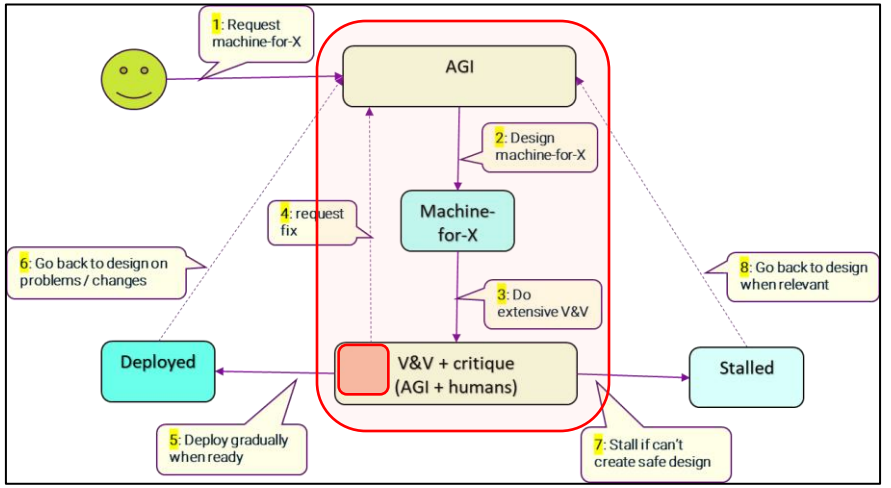


- About me
  - CTO & Founder of Foretellix: We help companies do autonomy V&V
  - We create V&V methodology, tools, content etc.
  - Did V&V of electronic systems => autonomy => AI-based autonomy

- I started worrying about AI safety / alignment around 10 years ago
  - But did not expect it to arrive so soon – yet here we are

- So in June I wrote a post + paper: The V&V method – a step towards safer AGI

- This presentation is just about *spec bugs* – not the whole paper
  - A spec bug is "a failure of the stated requirements to capture what we actually want"
  - Related to "unknown unknowns" and hard bugs in general
  - Perhaps the biggest killer in AI-based autonomy



- FM people already worry about incomplete specifications
  - But it may be worse than you think (especially with AI-based systems)

- I'll try to sketch how SOTA V&V of AI-based autonomy handles it
  - Will be happy to help if interested
  - You can contact me at yoav.hollander@foretellix.com

Can autonomy-grade V&V make AI safer?

foretellix

Design and V&V of AI-based autonomy (the single-slide version)

- Let's look at how people currently do design and V&V of AI-based autonomy
  - E.g. Autonomous Vehicles (AVs)

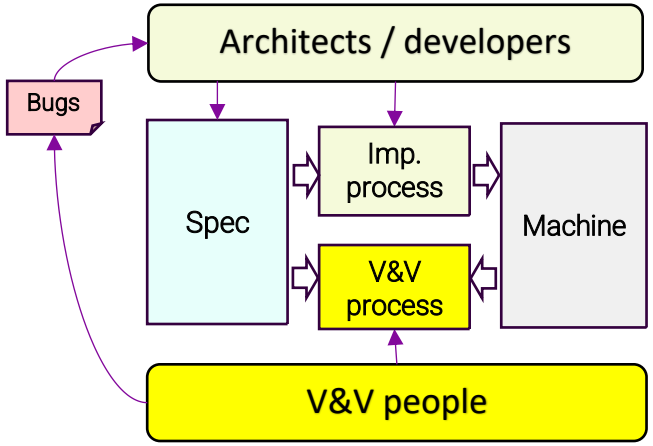
- The spec is about
  - Safety, functionality, performance, availability, security, …
  - (also about architecture / structure / interfaces)

- The machine / SUT (System Under Test) is the full AV
  - AI part: Most of perception, prediction, planning, control
  - Rule-based part: Monitor + backup system (mostly C++)
  - Other pieces: Sensors, mechanics, dynamics, …

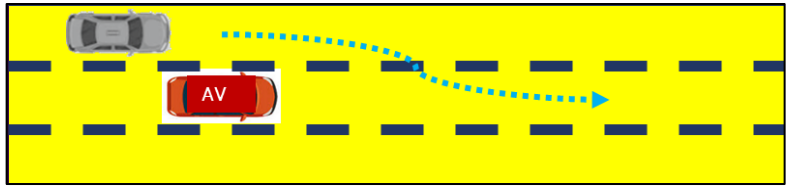
- The V&V process
  - Can use formal methods, Coverage Driven Verification (CDV), …
  - Note: I'll use "verification" as shorthand for "V&V + safety case"

- Main technique is massive, scenario-based CDV
  - Bombard the (simulation of) the machine with scenario instances
  - Auto-check how the SUT behaves, do triage / debug, document bugs

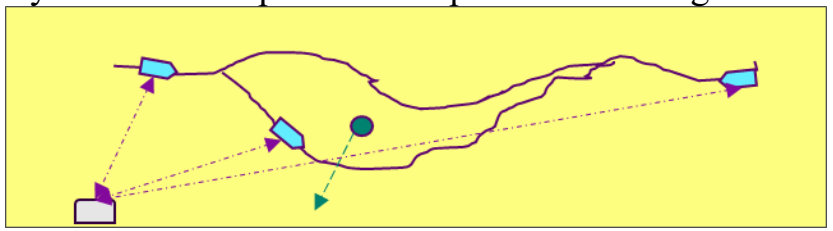
- The SUT can be an AV, a drone, a robot, a multi-AV mine system, …



An AV SUT with another vehicle doing a cut-in.

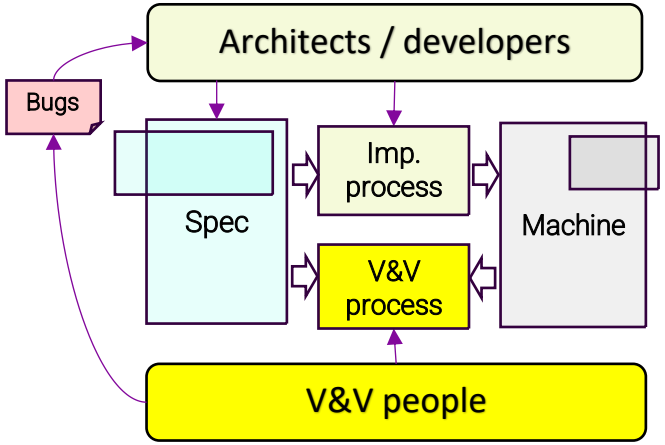


A mine SUT with three autonomous trucks controlled by a central computer. And a person is running.

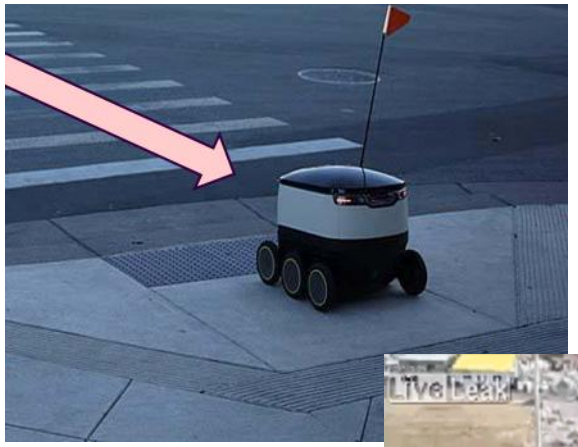


foretellix

Spec bugs – the problem



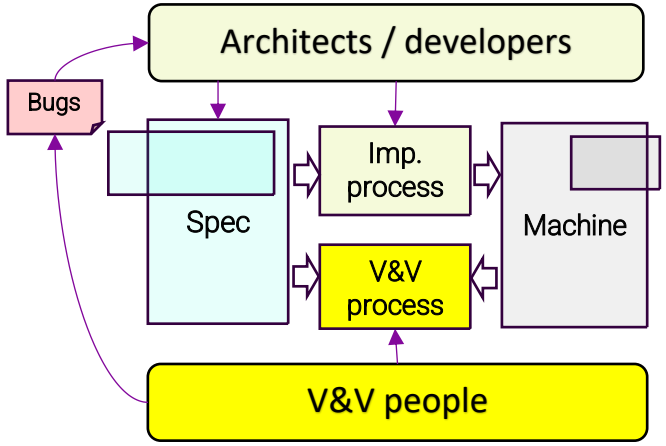
- Let's look again at that "design and V&V" diagram
- The "classical story" says: The V&V process finds bugs in the machine
  - And then developers fix the *machine* so it obeys the spec
- But in reality, there are lots of "spec bugs": The *spec* was wrong / partial
  - And this is getting worse with AI-based autonomy
- Example of a true spec bug
  - Stationary delivery bot [blocked a lady on wheelchair] from entering sidewalk
  - Probably nobody thought that "A bot doing nothing" could itself be unsafe



- There are ways to find spec bugs, and then generalize them
  - With luck you may be able to generalize the "stationary bot" bug to catch the (hypothetical) "stationary-AV Tsunami bug"
- Spec bugs are obvious *after* discovery – but you can't enumerate them upfront
  - "Obviously we should consider AVs moving from a left-driving to a right-driving country"

foretellix

Spec bugs – it's worse than you think



- Spec bugs often appear in "improved versions" (e.g. the Ariane 5 explosion was a "reuse bug")
  - So you need to fully verify each new version, each new ODD (Operational Design Domain), …
- Spec bugs are not just about safety. Consider the "bridge to Malmö" capacity planning bug:
  - Bridge designers failed to anticipate the influence of the bridge itself on usage patterns
- Or consider the Spectre / Meltdown security bugs
  - Security is often formally-verified, yet this cache side-channel danger stayed hidden for ~20 years

- Lots of other examples – see my post It's the spec bugs that kill you
  - E.g. a military targeting device caused soldiers to call a fatal air strike on themselves (because it was programmed to reset its GPS coordinates after a battery change)
- Spec bugs are a notorious failure mode of requirements-based-design
  - Initial requirements are always partial (unless you add the requirement "never do anything bad" ;-)
  - So you find a bug, then add the requirement "it should not happen"
  - Consider 26262 (talks about requirements) vs. SOTIF (talks about "unsafe unknown")

Spec bugs – even worse with AI-based autonomy

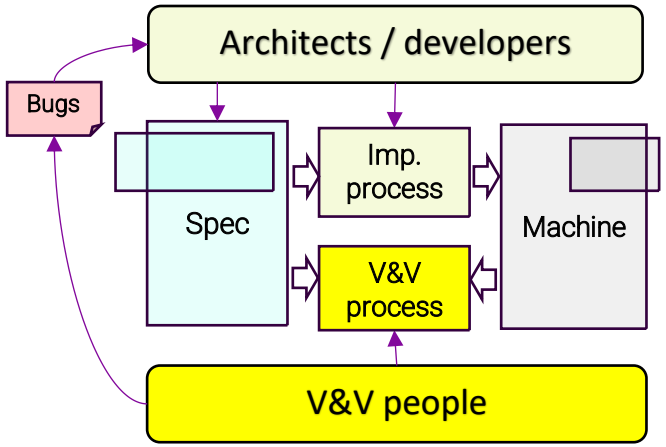
- In end-to-end autonomy you often start with almost no spec
  - By simply imitating e.g. “good human drivers” (using IL then RL)
  - So initial spec is effectively “Imitate good human drivers, but drive even safer”

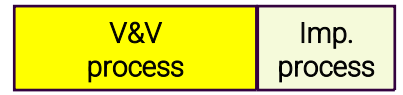
- You then incrementally add scenarios and checks (for safety, legal driving, …)
  - Those, in a sense, constitute the spec

- Spec writing seems less urgent: No need to pre-define components, protocols etc.
  - There are (almost) no components anyway

- Also, V&V becomes harder
  - No components means no modular verification – mostly black-box V&V
  - Non-determinism is hard (see discussion about local repeatability later)
  - People do add human-interpretable outputs (but they are suspect)

- So the *verification ratio* (relative size of V&V effort) *grows* for AI-based autonomy
  - AI makes implementation much easier (but often buggy)
  - V&V is harder (and more important)





foretellix

Spec bugs and reward hacking

- Reward hacking is a big issue in AI alignment / safety, and has two main variants
  - More “innocent” specification gaming / goal steering
  - More “serious” strategic deception / scheming

- Claim: A system which can *repeatedly* find bugs (including spec bugs), can (mostly) fix the *innocent* variant
  - (but not strategic deception and the danger of convergent instrumental goals)

- Examples of specification gaming / goal steering it could fix
  - The boat-race game hacking where the agent looped instead of racing
  - Robot fakes grasping by occluding the camera
  - A chatbot stopped speaking Croatian (because Croatians tended to express more negativity than Americans)
  - AGI killing us all when you request “make sure nobody dies of cancer” (chapter 4 here)

- To understand the claim, consider the *human* developers behind those spec bugs
  - Were they reward-hacking when they ignored <the-problematic-situation>?
  - No, it simply never occurred to them – so they settled on a simpler (but bad) solution

- Similarly, an AI is often trained (say via RL) to do various “bad shortcuts”
  - But repeated, automatic penalties for bad shortcuts will (often) generalize correctly
  - *“Good V&V aligns training pressure towards ‘what you really want’, and away from ‘Goodhart to a proxy’”*

foretellix

V&V system – main pieces

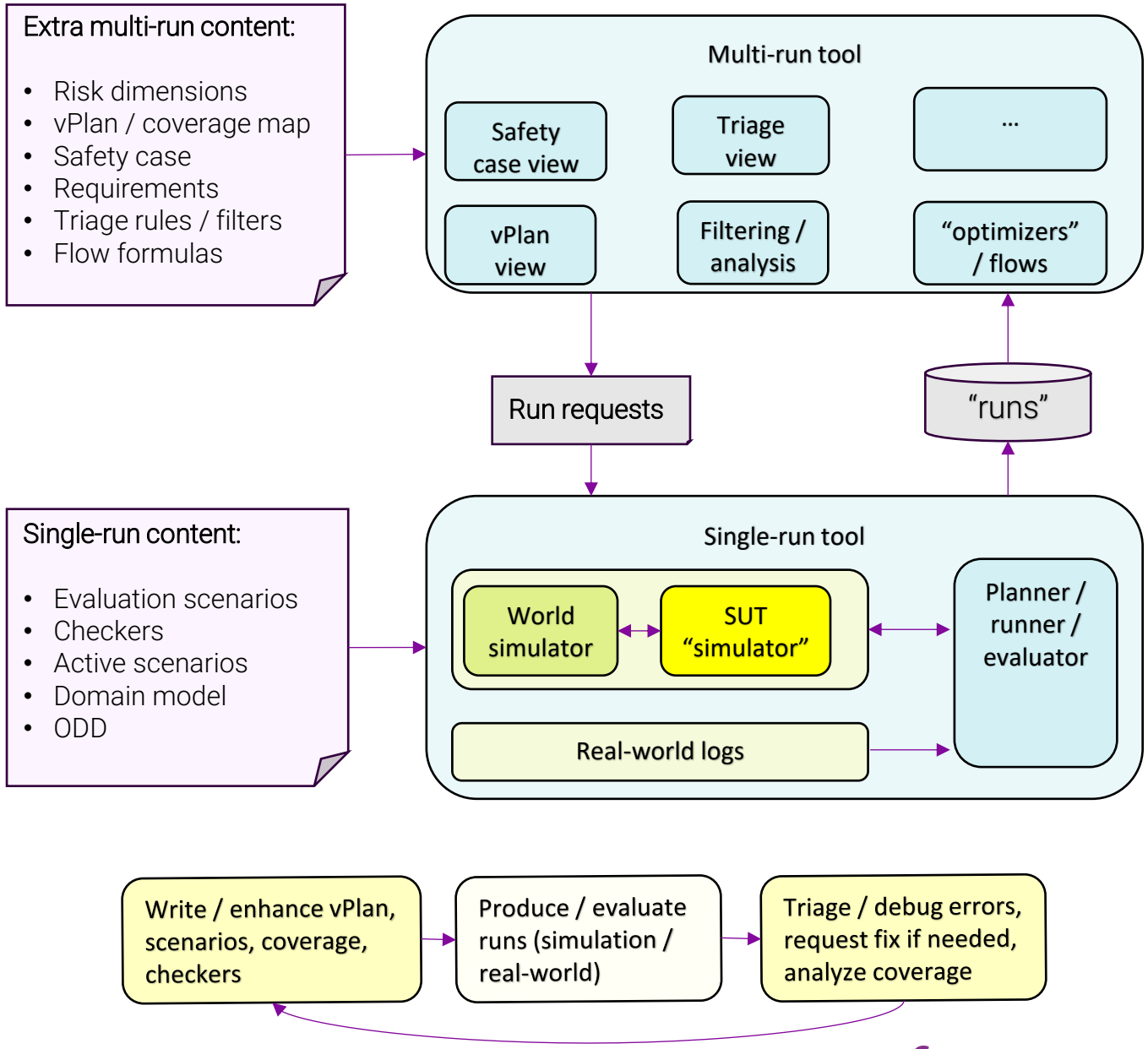
- V&V people (with increasing AI help) do
  - Flow requests, analysis, content modification
  - Goal: **Maximize risk-reduction-per-week** (with fixed resources, given where we are today)

- V&V content is mainly formal abstractions
  - And some structured English
  - Much of it is temporal expressions ++
  - E.g. scenarios, checkers, triage rules, ODD, domain model

- Content is SUT-specific
  - AVs, mines, drones, robots (for factory floor, for assisting elderly people, …), …

- "Runs" is mostly a database of typed intervals with attributes
  - Found by e.g. evaluation scenarios (with coverage attributes)
  - Or by checkers (with error severity and other attributes)

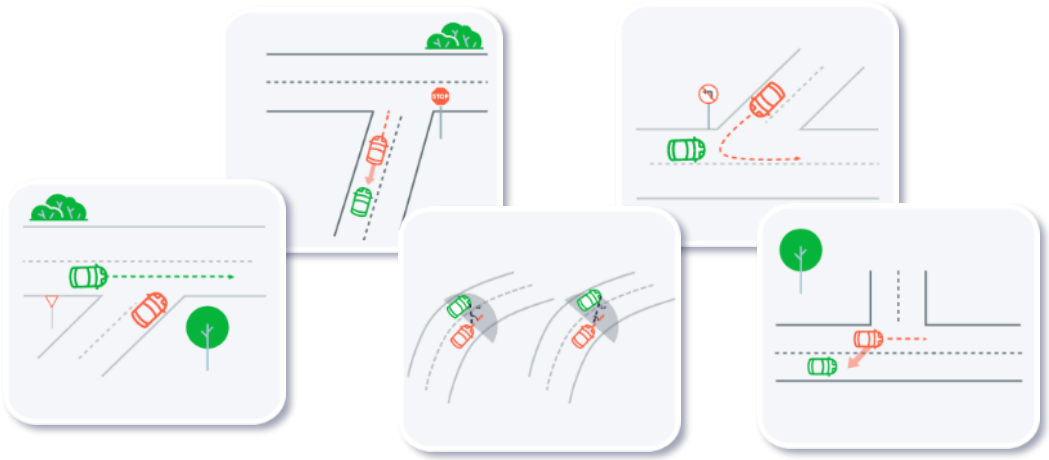
- On a typical Monday, V&V people look at the weekend's runs
  - Analyze which areas are still under-covered (in vPlan view)
  - Triage / cluster / understand checker issues (in triage view)
  - Refine the scenarios / checks (according to the above)
  - Start some new flows



Scenario-based-CDV basics

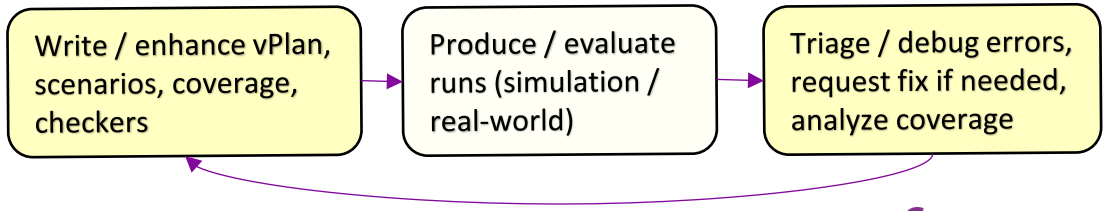
Main ideas (details here)

- Verification is never complete
  - We only aim for some "reasonable approximation"
- Verification is a resource-bound engineering process
  - You have limited resources, and so V&V is really an optimization game where your goal is to maximize *risk-reduction-per-week*, given some fixed compute and human resources
- Verification is influenced by current *priorities*
  - How mature is your SUT? Do you want all issues or just perception issues? Which model of the SUT?
- Verification needs random mixing of ***risk dimensions***
  - E.g. so as to catch spec bugs
- Verification is an ***iterative process***



- Behavior variability: Cut-ins, vehicle following, junction behaviors, …
- Other-actor variability: How law-abiding / assertive, density, …
- Road element variability: Freeways, streets, street width, curvatures, …
- Weather variability: Nice weather, rain / snow / blizzard, …
- Country variability: Left/right-driving country, different country rules, …
- Fault variability: Tire punctures, camera failures, communication failure, …
- Special request variability: Passenger emergency stop, re-route request, …
- …

- Example 1: After finding a bug, you realize there is a whole area you did not consider: *Create a new scenario*
- Example 2: You notice that many more of the bugs / suspect areas reside in (say) the Ego's junction behaviors: *Give more weight* to that area in the vPlan.
- Example 3: You plan on expanding to Japan: *Add left-hand-driving* to your runs



foretellix

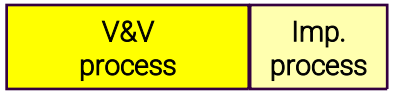
How to find spec bugs

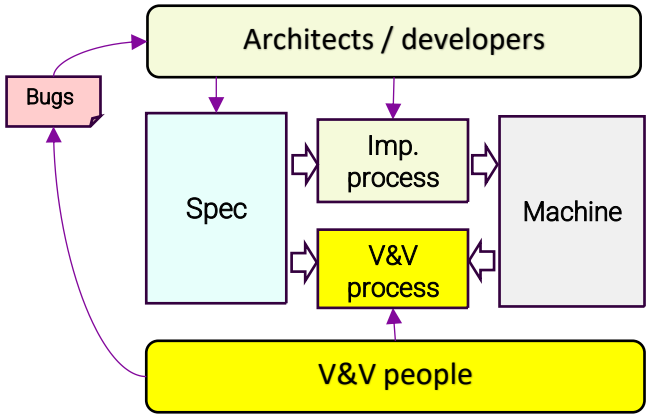
- Short answer: Scenario-based CDV (as described) is a big part of the answer
  - By mixing risk dimension + running search optimizers + methodology
  - See this post (and the spec bugs post)
- Good V&V includes not just looking for "bad things" (e.g. collisions)
  - But also for precursors (e.g. "low time-to-collision", and then "somebody running a red light")
  - Looking for spec bugs involves extending this *even further* backwards
- Emphasize "shallow" mixing of many risk dimensions
  - Example: Cruise AV hit a fire truck driving on wrong side *and* running a red light
- Use various kinds of generalizations
  - *Generalize* e.g. the Tsunami bug to other unexpected natural phenomena (like a mudslide)
  - *Instantiate* generic hazard patterns (mode change issues, identity issues etc.)
  - AI can help here (more on that later)

foretellix

“V&V is (almost) all you need”

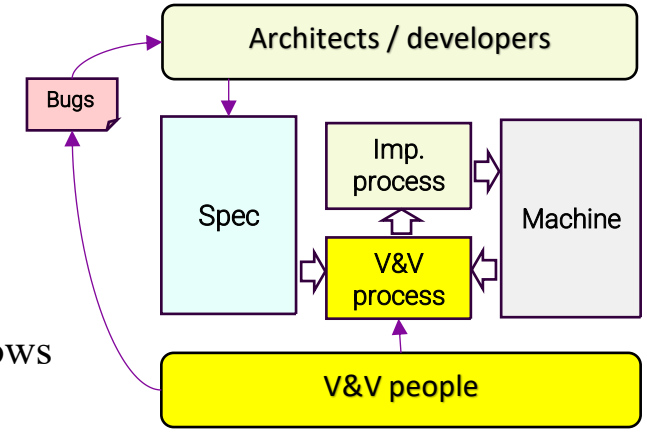
- As we move to AI-based autonomy, most of the work becomes “essentially V&V”
  - The verification ratio grows (as discussed before)
  - But also *implementation* (via training) is V&V-like





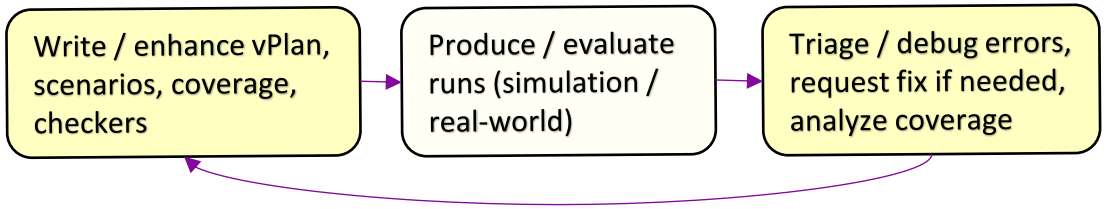
- Example: Say you find cut-in bugs – to fix them you train on additional cut-in snippets
  - Good cut-in examples for IL, both good and bad cut-ins for RL
  - Covering night/day, curved/straight, fast/slow etc. – you need variability just like V&V

- So implementation-via-training is a “continuation” of V&V
  - Use V&V to *find* problems (and later to *validate* the training)
  - Use V&V techniques (coverage, selection / test generation) to *fix* the problems
  - And much of that implementation process can be automated



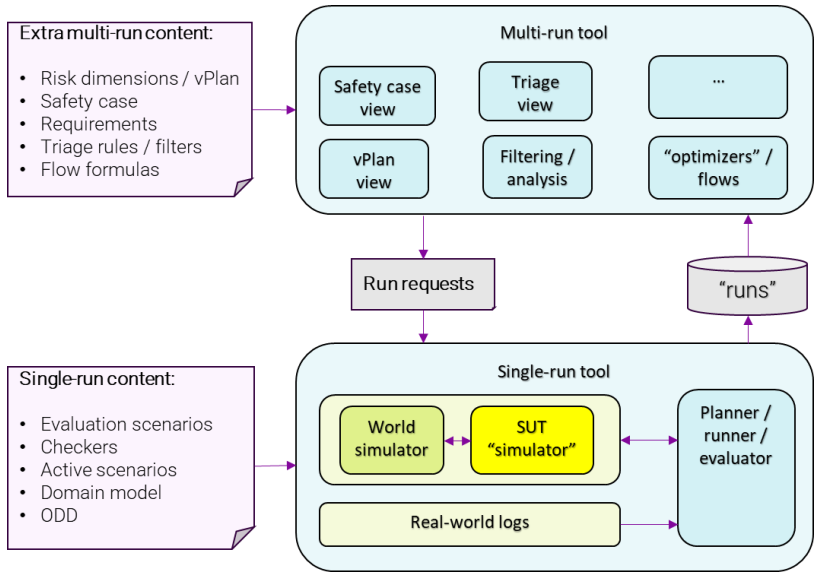
- Similarly, implementation-via-code-generation (e.g. Claude Code) increasingly turns to V&V-based flows
  - You give it a V&V automatic machine, it auto-improves the code

- If this trend continues, the majority of work will be in the V&V loop
  - And in determining if this is a spec bug, how to fix it etc.
  - “The spec is the source, and V&V is how you improve it”

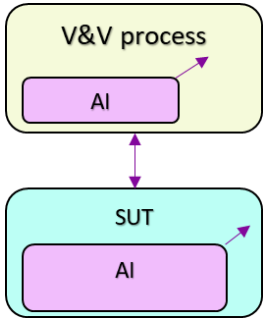


foretellix

Using AI in the V&V process



- It is mostly used to speed up what engineers already do
  - Mostly analysis and content creation / modification
- An important usage of AI is to help create readable, *formal* abstractions
  - "The main thing you save in Git are scenarios (with comments), not prompts"
  - V&V engineers need the format they trust (scenarios, not LTL, not prompts)
- There is also a growing interest in "informal abstractions"
  - And how to combine them with formal abstractions
  - E.g. generative world models for creating scenarios "from scratch"
  - E.g. multi-modal retrieval models for scenario recognition (e.g. "pedestrian hesitates", which is hard to formalize)
- AI can help in balancing realism and edge cases
  - You don't have many examples of edge cases, so hard (but crucial) to make them realistic



- AI can help in *generalizing* spec bugs (as mentioned before)
  - And *instantiating* generic hazard patterns (mode change issues, identity issues etc.)
  - Perhaps consider very-abstract-simulation (CoT-like) for initial exploration (e.g. see this excellent DeepMind paper)
  - This seems like an important (and practical) research area

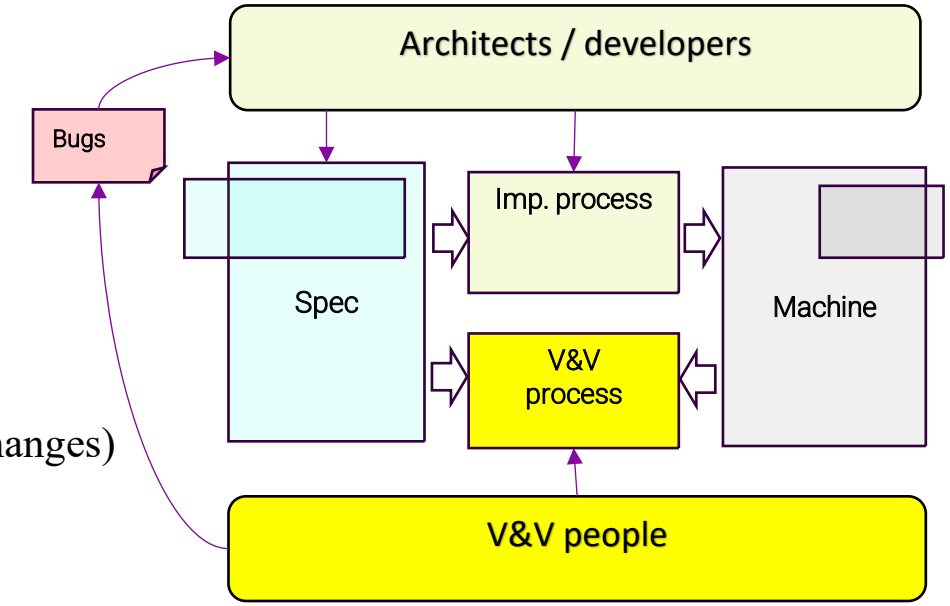
foretellix

Backup slides

See also a longer version of this presentation I gave to some ARIA / Safeguarded AI people (slides, 90-min recording)

Unintuitive traps from V&V practice

- Requirements-based design has big issues
  - As mentioned above (and "V&V is (almost) all you need")
- Testing in the ODD is not enough – need to test also at ODD *edge*
  - E.g. need to test what happens when it starts snowing on your no-snow ODD
- Local repeatability is hard but important (else can't rerun with more logging / small changes)
  - Ensure simulation repeatability via precise timing, setting the AI seed etc.
- Handling multiple configurations efficiently is hard
  - Multiple countries, SUT models, … (see details slide)
- Checking complexity is mostly about context / special cases
  - "Don't drive on sidewalk, except …"
- Much of checking is statistical / "analogue"
  - Resulting bugs are "tradeoff bugs" (see details slide)
- Triaging is *really* important
  - Clustering, debugging, assigning to people, setting bug status, … (see details slide)

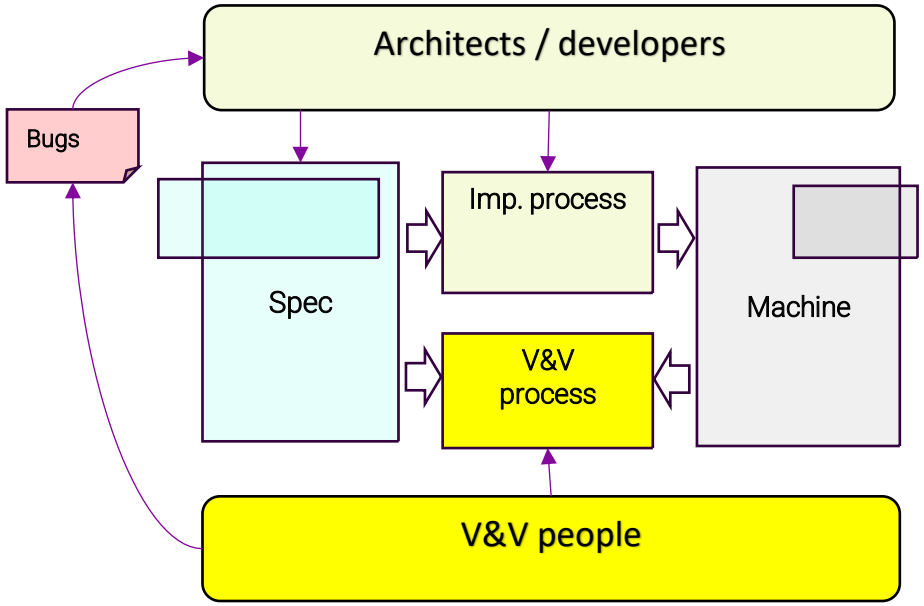


foretellix

Verifying multiple configurations

- "Machines" often come in multiple configurations
  - For AVs: multiple countries, multiple models, …
  - Each needs to be verified individually

- A common problem occurs *between* companies
  - Stack provider (supplier of the SW / AI) does extensive V&V
  - But the vehicle provider *cannot* just trust that V&V
  - Especially since the consequences of failure will mostly land on them



- They need to re-verify with the vehicle's *actual* sensors, dynamics, ODD, special quirks, …
  - Remember: One can build buggy systems out of perfect components (due to mismatched expectations)
  - This is even worse with black-box AI-based systems

- The ideal solution is to ship the stack with a full V&V environment
  - Which the vehicle provider can use / extend

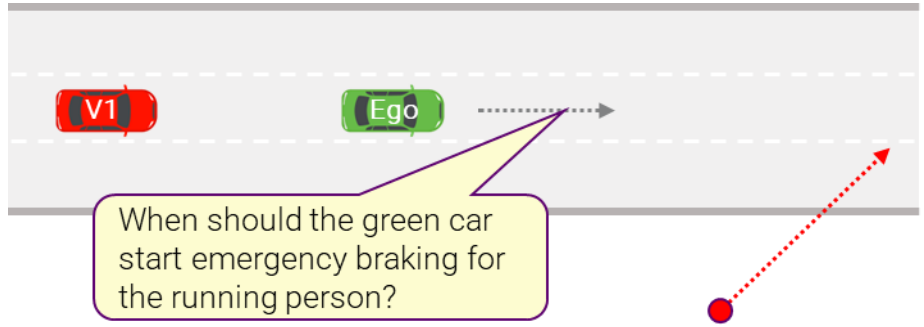
- This is similar to adding another risk dimension (e.g. multiple countries)
  - With the added complexity that V&V is done by another organization

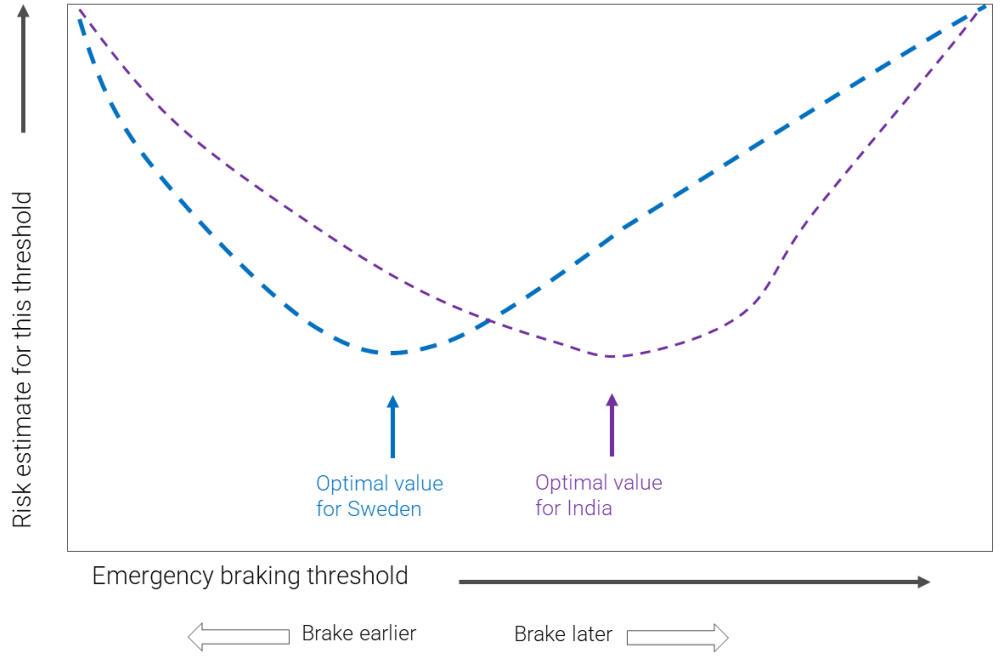
foretellix

Tradeoffs bugs

- Your autonomous system *is* going to cause bad things
  - Because it is interacting with humans, with all their faults
  - We just want to *minimize* overall risk
  - Thus many bugs are “tradeoff” or “analogue” bugs

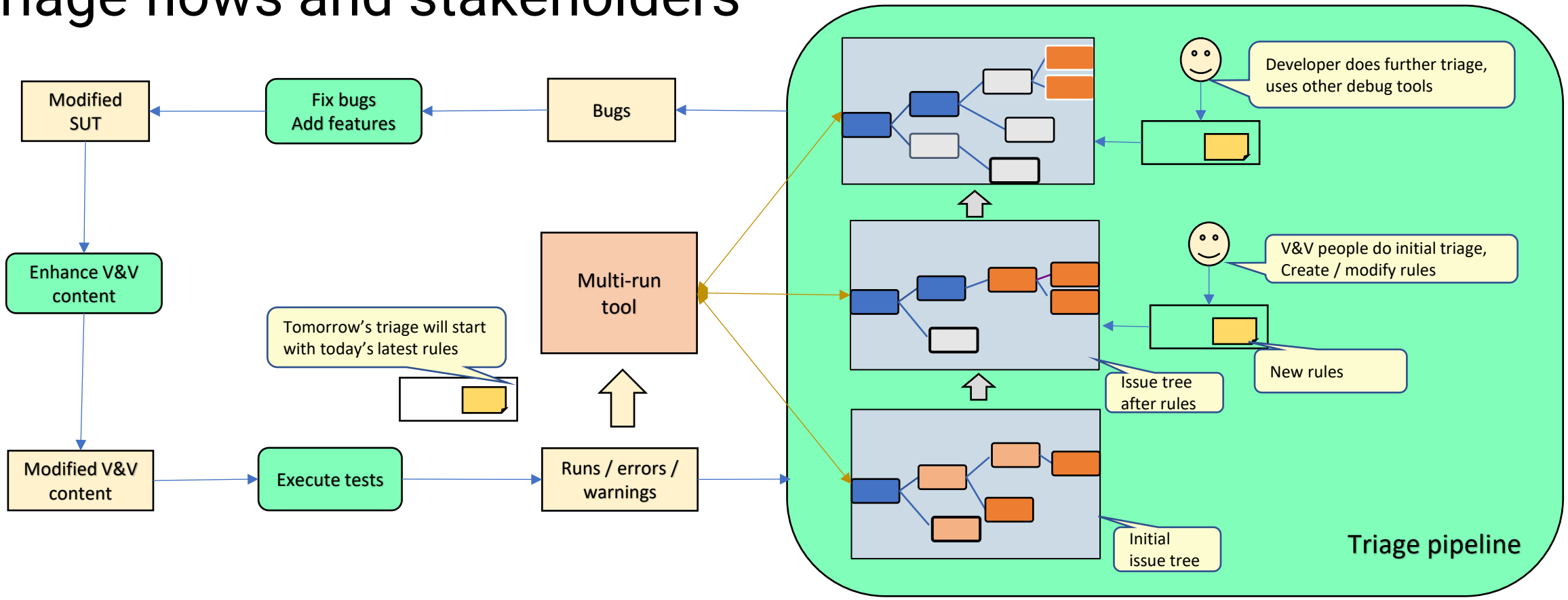
- Much of the checking is statistical
  - “Near-collisions per KM in (typical SF traffic) are *30% worse* this week”





foretellix

Triage flows and stakeholders



- Rules are used for
  - Clustering issues into groups / sub-groups
  - Assigning attributes (like severity, owner, Jira ticket, …)
- The issue tree is a really a tree-table
  - With the "important" attributes shown as columns
- Some issues take longer to fix (or even decide if bug)
  - Need to "actively" ignore them for now (for efficiency)
- Rule mgmt. is difficult and layered
  - Different stakeholders need somewhat-different rules