Navigation Field Triage Doctrine

(go/proxy-nav-triage-doctrine)

Status: Draft
Authors: ammarh@google.com
Reviewers: x-proxy-sw@, x-proxy-ops@
Last Updated: 2019-10-08

Objective

The purpose of this document is to codify & formalize a principled process that defines the interaction between the design & engineering of the Navigation system on Proxy robots with its deployment, testing & operations on the field.

Background

Proxy robots currently operate for several hundreds of hours every month gathering large amounts of interesting data samples. The robots perform various tasks in slightly different environments that exercise its functional subsystems of manipulation & navigation. These systems inevitably have certain failure modes that are all, for lack of a better term, collectively referred to as a “bug”:

(i) Capability limitations:
These are limitations that the currently deployed system is not designed for. Ex: detecting glass doors.

(ii) Regressions due to code changes:
These are failures that might have inadvertently gotten introduced while fixing a bug or adding a new capability.

(iii) Trickle down failures:
These are failures that originate upstream and propagate down the software stack to exhibit symptoms downstream. Ex: The stereo camera publishes noisy data and the robot keeps stalling. These could recursively be caused by one of the same 4 failure modes in the upstream subsystem.

(iv) True failure modes:
These are failures where we expect the robot to work however it clearly failed, not related to any

of the 3 categories mentioned above. Swiftly getting to & acting upon such bugs (via a fix or a design consideration) is the essence of designing this process.

Overall, each bug is a valuable statistic and when properly aggregated over time should guide feature development and prioritization. Therefore, instead of a whack-a-mole approach to act upon each bug/data sample, it is imperative to clearly define the process on how that feedback data gets addressed & incorporated into the engineering design. Given the chaos & clutter that robots encounter in the unstructured real world we must strive to make this process as structured, precise & predictable as a well-oiled machine. While nothing outlined in this document is particularly complex or “rocket science”, the hope is to in the end come up with a process that is something akin to science.

Current Process

There are 3 main teams that log several hours operating Proxy robots: (a) Continuous Operations, (b) QA, and (c) Robot Support. In the current operations flow when an odd robot behavior is observed, a bug is created. These bugs are **approximately & symptomatically** categorized into planning, localization or perception and directly assigned to one of the engineers. The first response of the assigned engineer is to delineate between the 4 categories the bug falls into. For (i) the bug usually gets closed, while for (ii) & (iii) the hot potato is passed, usually based on a quick Piper history search. In order to make this process slightly more palatable, debug screenshots are requested following instructions in this doc. However the context information is mostly missing, hard to capture on the field or not entirely sufficient.

Bug Triaging Resource

Currently Perception SWEs triage all incoming bugs. On a weekly basis, bug rates and resource required are as follows:

- New incoming bugs: 40 (30 Navigation, 10 Mapping/Localization)
- Bugs backlogged or closed without proper triage: ~50%
- Engineering hours spent per week: 10-15 hours

Limitations

While the current process got us quickly off the ground, it is very evident that it does **not scale**. Some observed limitations are as follows:

- Bug fixing is subjective and a category (i) to one engineer might be category (iv) to another.
- Since bug diagnostics & categorization is partly handled by engineers, it's hard to accurately scope & allocate time for design & engineering. This is due to the unpredictable nature of bug frequency as well as the time penalty induced by frequent context switching.

- Due to a piled up backlog of bugs from all 4 categories, category (iv) at times gets buried or delayed in getting attention.
- Given the distributed nature of bug fixing, we lose out on consolidating vital field data such as what are the most critical capability limitations that the robots constantly run into.
- We lack requisite tooling & infrastructure since the feature requests are not unified, making it difficult to prioritize and focus on what's truly important.

Tool Limitations

Our current set of triage tools have significant functional and performance limitations. When a user needs to play back a log, the log needs to first be downloaded, which takes 20-25 minutes. After the log is downloaded, the user then needs to pinpoint to a specific time to start log playback (1-5 minutes). If the playback time is incorrect (i.e., need to fast forward or rewind), the user needs to kill the playback service and reload the log (1-5 minutes), and iterate until the right point in time is found. Root-causing the problem typically takes an additional 5-10 minutes after the log is correctly loaded. Overall, >80% of time in triage is spent loading/reloading logs in the visualizer.

Overview

On a high level the process is straightforward and involves a coordinated & synergistic interplay between 4 units: *Operations, Triage, Tools & Navigation*. *Ops* will make observations regarding anything deemed noteworthy from the field. These can be either positive or negative. In either case, the observation needs to be triaged. The triage process involves dedicated & trained navigation *Triage* champions taking Ops' observations, deriving further context data and binning the observation. The binning is based on a taxonomy that is defined by the *Navigation* engineers. This taxonomy alone dictates (a) what the observation is, (b) what is to be done about it, and (c) how it will be addressed (filed or fixed). The branches in the taxonomy identify the domain expert who will fix the bug, if a fix is indeed warranted. Through this process the Triage champions will make feature requests for additional *Tools* that maximizes their triage productivity, accuracy and bug throughput.

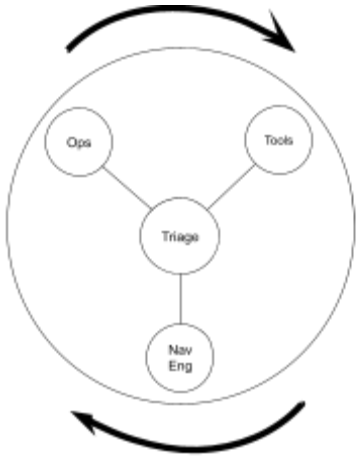

Figure [1] : Triage champions are the hub in this wheel and they provide Proxy with operational momentum to move 10X faster.

The fundamental guiding philosophy for this process is the fact that each group is able to most efficiently & productively operate at different cadences. While it is inevitable to occasionally operate across a broad cadence spectrum, doing so under normal conditions not only results in lost productivity, it leads to normalization of deviance whereby we miss critical failures. The following section dissects the proposed process into further detail.

Detailed design

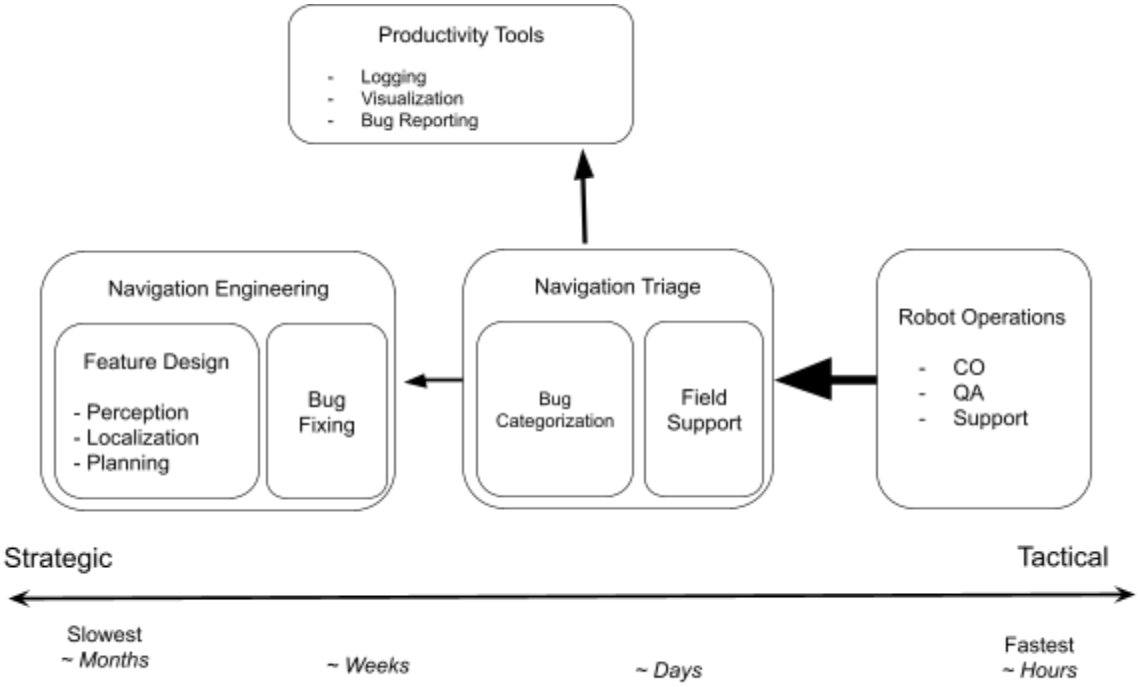

Figure [2]: Each unit operating at their natural cadence.

Figure [2] above is an illustration of this process design. Each rectangle above is a functional unit of personnel with domain expertise. The arrows in between is an established contract that establishes a common interface through which data flows. Examples: Navigation Behavior Taxonomy and Triage Tools Feature Requests . The thickness of each arrow indicates the throughput on that channel. Consequently the arrows become thinner as you go from right to left in this process diagram.

Personnel

Navigation Triage Champions:

People

X@, Y@, Z@

Responsibilities

Shall:

- Ramp up time for a new triage champion to become fully functional should be about 4 weeks.
- Absorb all bugs that have been filed while operating the robot by CO, QA, Support or other individuals.
- Provide field support for any navigation related questions or issues while running the robot.
- Categorize every navigation bug into the lowest level of granularity as determined by the Navigation Behavior Taxonomy with a brief diagnostic. Assign bugs to the appropriate hotlist defined by the branches of the taxonomy.
- Categorization must be completed within 72 hours from when the bug was filed.
- Provide guidance on feature requests that maximize productivity by owning & augmenting Triage Tools Feature Requests.
- Populate aggregate statistics & histograms of bugs in every category of the taxonomy and present to navigation-eng@ on a weekly cadence.

Navigation Engineering:

People

ammar@ (Perception), joergm@ (Mapping & Localization), mpersson@ (Planning)

Responsibilities

Shall:

- Own & maintain the Navigation Behavior Taxonomy.
- Train Navigation Triage champions.
- Address all bugs that were not categorized in the taxonomy with a more detailed diagnostic.
- Fix bugs of categories (ii) & (iv) incorporating the diagnostics from the Triage champions.
- Prioritize feature requests for new capability through field aggregated statistics.

Triage Tools:

People

sarahcoe@ (Logging), jrundquist@ (Visualization), X@ (Bug Reporting)

Responsibilities

Shall:

- Build features & tooling support as outlined in Triage Tools Feature Requests.
- Provide development timelines for each outstanding request.

Contracts

Navigation Behavior Taxonomy

Owner

navigation-eng@

Purpose

- Document a set of known navigation failures tying them to their symptoms as well as potential sources of error.
- Minimize tribal knowledge or worse yet miscategorization of failures.

Triage Tools Feature Requests

Owner

navigation-triage@

Purpose

- Provide guidance for the most impactful tooling to maximize productivity & accuracy.
- Achieve bug categorization in under 5 minutes per bug with less than 10 mouse clicks.

References

- Launius, R. (2002), Managing NASA's Complex Space Flight Programs: The Apollo Experience
- DeMarco, T. (2001), SLACK: Getting past burnout, busywork, and the myth of total efficiency.
- Allen, D (2002), Getting Things Done: The Art of Stress-Free Productivity
- Vaughan, D (1996): The Challenger Launch Decision