# Introducing static analysis to a mature project

Jacob Sparre Andersen

JSA Research & Innovation, Jægerparken 5, 2. th., 2970 Hørsholm, Danmark

Static analysis tools (eg. AdaControl, CodePeer) are accepted as a useful way of identifying potential problems in source text before they turn into system failures or make a project prohibitively expensive to maintain.

I have previously been presented with anecdotal evidence that – as good as such tools may be – if they aren't used on a project from day one, it is very hard to get any benefit from them, since they have a tendency to need the developers to stay within a limited set of patterns, which the tools can recognise as correct.

Since January 2016 I have been working with a client, maintaining an application initially developed in 1987, which includes of more than one million lines of Ada source text. This application has traditionally been developed without the use of static analysis tools (besides a small selection of enabled compiler warnings).

As the customer decided to focus on increasing quality (to reduce support costs), we experimented with using AdaControl to identify problems in the source text. The good news was that AdaControl had a single rule matching a significant part of the registered incidents. The bad news was that AdaControl reported many more (potential) issues than we had resources to do something about[1].

We are handling this in several ways:

- My client funded an extension of AdaControl, such that the rule recognises the most common, correct implementation pattern in the application.
- For a start, we decided only to run AdaControl on units which are touched anyway; either due to a change request or due to an incident.
- We track individual rule violations on a per-file basis in our continuous-integration system, to ensure that the number of violations of each rule is not increasing.

Extending the tool reduces the number of false positives on checks of our most critical problem.

Only checking units, which are touched by a developer anyway, limits the effort spent fixing sources to where we are likely to introduce new problems.

Tracking rule violations will allow us to increase quality, even with an explicit non-zero-warnings policy. If we have zero-warnings, we are probably not checking our sources for enough kinds of problems.

As an indication that the focus on increasing quality already has paid off, the frequency of incidents has been reduced significantly since the beginning of 2016[2].

The presentation will:

- Report the effect of each of the three attacks on the problem listed above.
- Contrast zero-warnings and non-zero-warnings policies.

Throughout the presentation both business needs and software engineering "needs" will be discussed.

I hope to close with a lively discussion including the experiences of the audience on the subject.

[1] Matching the above-mentioned anecdotal evidence.

[2] This change is not **only** due to the work presented here.