CLANG AST ANALYSIS AND REPAIR

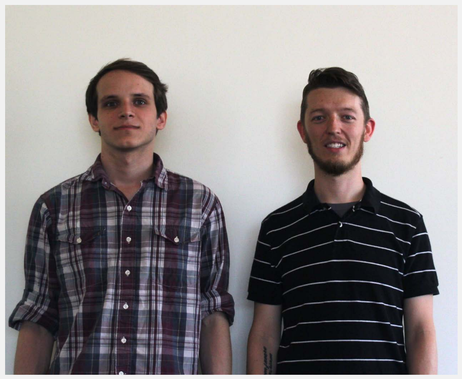
A Code Quality and Compliance Development Tool

The Design, Planning and Collaboration Stages

Our Client and their Goals

Garmin AT develops navigational components for aircraft, and must meet Federal Aviation Administration (FAA) guidelines for safety and operations certification prior to allowing any component into any operational aircraft. Garmin AT currently uses regular expressions to locate, log and fix code that does not comply with FAA code standards. We are developing a more robust code check and fix system using Clang AST libraries in conjunction with regular expressions to broaden the capabilities of the system to accommodate all coding standards. Ease of maintainability and extensibility are key factors for our client.

The system utilizes discrete rule sets that use a combination of capabilities in the Clang AST and Python regex libraries. The system continues to be implemented within Python and is geared toward evaluating C code. The goal is to meet all code format and structure requirements mandated by FAA regulations and Garmin internal standards. The client has provided a list of rules/issues for each requirement that must be met. All flagged elements will either be automatically corrected, or noted in a log file for future manual corrections; run-time option flags set by the user determine which rules are evaluated and which actions are taken.



Charles Santos Nicholas Nelson

Thanks for the great year, GARMIN™

Development Roadmap and Work Division

Development on issues began in late November 2014, with version 1 released on February 11. From there, we expanded functionality for issues and infrastructure code. Version 2 has been released on May 8.

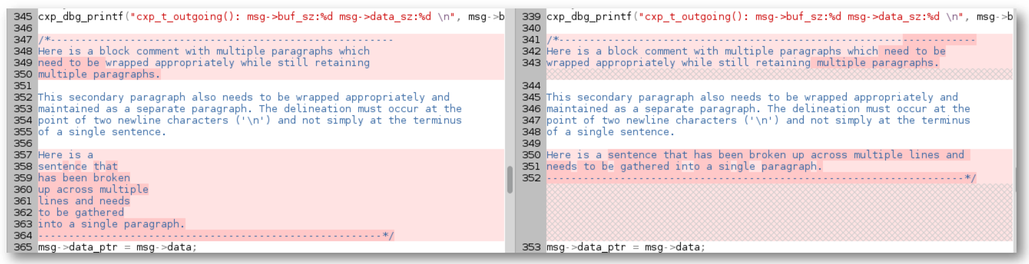
We have been working independently on issues because each issue is small enough to only warrant one developer. However, we discuss our designs and final solutions regularly between ourselves and with our client to ensure we are producing a cohesive final product. Final integration has been a team effort.

Growing Pains

Medium-early in development, we ran into a few issues with how the architecture was structured. Between the two of us, we developed on Linux and Windows which lead to some problems because the tool deals with files and file paths frequently. The solution was to implement extensive platform-dependent path normalization.

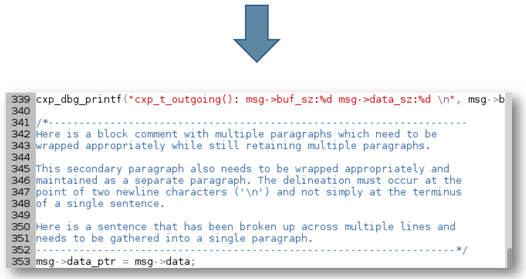
Included Rules and Selected CLANG Examples

- Comment length and line wrapping
  Uses CLANG to easily locate and modify comments
- Local variable aggregation and sorting
  Uses CLANG to locate local variable declarations which would have required complicated regular expressions
- Include statement aggregation and sorting
  *Uses CLANG to locate **#include** directives*
- Automated function prototyping
  Uses CLANG to identify function definitions and then generates a matching prototype signature
- Full-file function definition sorting
  Uses CLANG to isolate function definitions then reorganizes larges pieces of code in the file
- Comment indentation matching
- Title header comment formatting
- Tab character conversion
- Function argument stacking
- Brace and full-file alignment

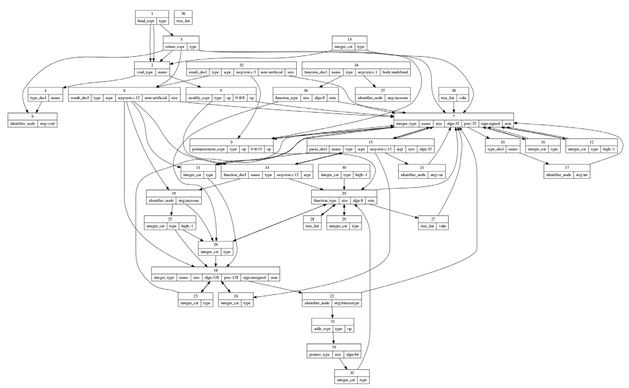


```
345 cxp_dbg_printf("cxp_t_outgoing(): msg->buf_sz:%d msg->data_sz:%d \n", msg->b
346
347 /*------------------------------------------------------------
348 Here is a block comment with multiple paragraphs which
349 need to be wrapped appropriately while still retaining
350 multiple paragraphs.
351
352 This secondary paragraph also needs to be wrapped appropriately and
353 maintained as a separate paragraph. The delineation must occur at the
354 point of two newline characters ('\n') and not simply at the terminus
355 of a single sentence.
356
357 Here is a
358 sentence that
359 has been broken
360 up across multiple
361 lines and needs
362 to be gathered
363 into a single paragraph.
364 ------------------------------------------------------------*/
365 msg->data_ptr = msg->data;
```

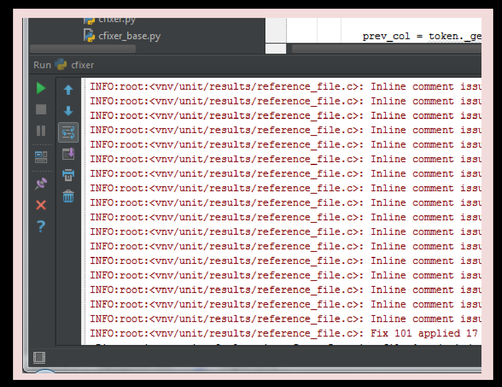
```
339 cxp_dbg_printf("cxp_t_outgoing(): msg->buf_sz:%d msg->data_sz:%d \n", msg->b
340
341 /*------------------------------------------------------------------------
342 Here is a block comment with multiple paragraphs which need to be
343 wrapped appropriately while still retaining multiple paragraphs.
344
345 This secondary paragraph also needs to be wrapped appropriately and
346 maintained as a separate paragraph. The delineation must occur at the
347 point of two newline characters ('\n') and not simply at the terminus
348 of a single sentence.
349
350 Here is a sentence that has been broken up across multiple lines and
351 needs to be gathered into a single paragraph.
352 ------------------------------------------------------------------------*/
353 msg->data_ptr = msg->data;
```



```
339 cxp_dbg_printf("cxp_t_outgoing(): msg->buf_sz:%d msg->data_sz:%d \n", msg->b
340
341 /*------------------------------------------------------------------------
342 Here is a block comment with multiple paragraphs which need to be
343 wrapped appropriately while still retaining multiple paragraphs.
344
345 This secondary paragraph also needs to be wrapped appropriately and
346 maintained as a separate paragraph. The delineation must occur at the
347 point of two newline characters ('\n') and not simply at the terminus
348 of a single sentence.
349
350 Here is a sentence that has been broken up across multiple lines and
351 needs to be gathered into a single paragraph.
352 ------------------------------------------------------------------------*/
353 msg->data_ptr = msg->data;
```



The Final Product and a Retrospective



The Application of CLANG

CLANG's strength comes from its internal representation of the input source file. This means it is great for dealing with difficult to parse items like local variable definitions, function parameter definitions or handling large chunks of code as one unit, like function definitions. However, CLANG's greatest weakness is that it is an AST. It does not keep track of whitespace, for example, which is central to formatting. In a number of issues, we implemented CLANG-Regex hybrid solutions because they were ideal. CLANG isolated the areas of the file we were interested in and then Regex let us pick out the minute pieces that needed modification.

The Product as a Whole

This product is undoubtedly very useful even with the relatively few issues that have been implemented so far. While its primary purpose is to find and fix problems in code, its other major strengths include its ability to find and fix an number of unrelated issues all in one batch run and is designed to integrate into a number of different IDEs.

Oregon State UNIVERSITY