Random Test Data Generation
Iteration 2 Report

Started 02/03/2026
Maksym Redchenko C00302240
SETU

Iteration Period: Weeks 7–10 (March 2026)

1. Changes to the Vision

The core vision of the project — a free, open-source, VS Code-native extension that generates Google Test boundary value tests for C functions on Linux — has remained unchanged during Iteration 2. The extension still targets the identified gap in the market: no free, VS Code-native tool generates Google Test code with boundary focus for C developers.

No features were removed from the vision.

2. Changes to the Specification

2.1 Additional Requirements Implemented

The following requirements were added beyond what was originally planned for Iteration 2:

#	Requirement	Rationale
1	Shared struct header pattern	Required when testing functions that accept two different struct types from different files. structs.h centralises all struct definitions and is included by both source and test files via CMakeLists.txt include_directories()
2	Per-field struct inputs in custom test UI	Custom test tab now renders separate input fields per struct field (e.g. p1.x, p1.y) rather than a single opaque input, making it practical to specify struct parameter values
3	Select / Deselect All in webview	Supervisor feedback from the Iteration 1 presentation explicitly requested this feature to allow bulk test inclusion/exclusion before saving
4	3-choice notification popup	After generation, users are offered Fill Expected Values, Build & Run, or View Tests — reducing clicks for the common case of immediately building without filling values

2.2 Requirements Deferred

#	Requirement	Reason for Deferral	Target Iteration
1	INFINITY / NaN float boundary auto-generation	Float-specific special values (INFINITY, NaN, -0.0f) are not yet auto-generated. They can be added via custom tests. Full auto-generation requires detecting guard patterns in the function body, which is beyond the current parser scope	Iteration 3
2	Header file (.h) parsing	Implementation continues to parse .c source files with function definitions. Header-only parsing was deprioritised in favour of deeper type support	Iteration 3
3	Extension unit tests (80%+ coverage)	Only sample tests exist. Development time was directed toward type support features this iteration	Iteration 3

2.3 Further Details for Existing Requirements

- **Void function testing**: Functions with void return type (e.g. scale_value(int *data, int factor)) now correctly call the function without capturing a return value, and assert side effects by reading the pointed-to variable after the call. Generated tests use the pattern: declare stack variable, pass its address with &, call function, assert variable value after the call. The fix was applied to both the generated test template and the custom test template in expectedValuesWebview.ts, which are separate code paths in the generator.
- **Overflow classification**: The generator was refined to distinguish genuine overflow cases (e.g. INT_MAX + 1) from deterministic cases where an extreme value is used as a single operand with a neutral second operand (e.g. INT_MAX + 0 = INT_MAX, or INT_MAX as a single element in max_array). The latter now receive firm EXPECT_EQ assertions rather than SUCCEED() placeholders.
- **EXPECT_FLOAT_EQ adoption**: All float/double return functions now use EXPECT_FLOAT_EQ / EXPECT_DOUBLE_EQ instead of EXPECT_EQ. Using EXPECT_EQ for float comparisons was a bug in Iteration 1 that caused false failures for values not exactly representable in IEEE 754.
- **Struct initialisation**: The generator was fixed to emit {0, 0} for two-field structs rather than {0}, which only initialises the first field and leaves the second undefined. Both fields are now always explicitly initialised.

3. Changes to the Research and Design

3.1 Additional Research

- **Boundary value semantics for non-int types**: Research confirmed that size_t is unsigned and cannot hold negative values, so negative size boundary tests are correctly omitted for size_t parameters. For char, CHAR_MIN and CHAR_MAX were verified as the correct boundary constants rather than 0x00/0x7F/0xFF, as the signedness of char is implementation-defined.
- **Google Test struct initialisation**: Research confirmed that C aggregate initialiser syntax {x, y} is valid in C++ test files for C structs, and that partial initialisation {0} leaves

subsequent fields undefined — informing the fix to always generate fully qualified initialisers {0, 0} for two-field structs [1].

- **Signed integer overflow in C**: Research into the C standard confirmed that signed integer overflow is undefined behaviour [2], meaning functions like absolute_value(INT_MIN) cannot have a correct portable assertion. The generator now emits GTEST_SKIP() with an explanation for cases where the operation itself is UB.
- **String parameter patterns in Google Test**: Research confirmed that char str[] = "..." is the correct declaration for mutable string parameters in test code, as opposed to const char * which cannot be passed to functions expecting char *.

3.2 Additional Design Decisions

- **Neutral operand pattern**: When testing one parameter at its boundary, all other parameters are set to their neutral value (0 for addition, 1 for multiplication, NULL for pointers). This prevents the other parameters from triggering overflow and ensures the boundary behaviour of the target parameter is isolated.
- **Struct boundary strategy**: Struct parameters are tested per-field — when one field is at its boundary, the other fields are set to 0. This prevents combined overflow from masking individual field boundary behaviour. Each struct parameter is also tested independently: when p1 is at its extreme, p2 is neutral, and vice versa.
- **Separate template code paths**: The fix for void function pointer handling was applied to both the generated test template and the custom test template in expectedValuesWebview.ts. These are separate code paths in the generator and must be updated independently — a lesson learned when the custom template continued emitting void result = f() after the generated template was fixed in boundaryValues.ts.

4. What Was Implemented and What Was Not

4.1 Successfully Implemented

Component	Description	Status
int[] array support	Size boundaries: size=0, size=1, size=10, negative size, size exceeds array length (GTEST_SKIP)	✓Complete
int* pointer support	NULL guard test (SUCCEED) + valid pointer test with stack variable and & address-of pattern	✓Complete
void return functions	Correct side-effect assertion pattern: declare variable, pass address, call function, assert variable value after call	✓Complete
Single struct param	{0,0} zero baseline, {INT_MIN,INT_MIN} minimum, {INT_MAX,INT_MAX} extreme — both directions tested	✓Complete
Two struct params	Each struct tested independently at boundary while other is neutral; custom tests with per-field inputs in webview	✓Complete

float / double BVA	FLT_MAX, -FLT_MAX; EXPECT_FLOAT_EQ assertions; deterministic cases correctly asserted instead of SUCCEED()	✓Complete
size_t boundaries	0, 1, 10, SIZE_MAX; correctly omits negative values (unsigned type)	✓Complete
char boundaries	CHAR_MIN, CHAR_MIN+1, CHAR_MAX-1, CHAR_MAX; custom tests use char str[] = "..." syntax	✓Complete
Test density config	cTestGenerator.testDensity setting: minimal, standard, exhaustive modes	✓Complete
3-tab webview UI	Boundary Tests, Custom Tests, Preview tabs; stats bar; Select/Deselect All; per-field struct inputs	✓Complete
Problems panel integration	Build errors parsed and surfaced with clickable file + line references	✓Complete
Shared struct header support	structs.h pattern for cross-file struct sharing; CMakeLists.txt include_directories() configuration	✓Complete
Overflow classification fix	Neutral-operand cases get firm EXPECT_EQ; only genuine two-operand overflow gets SUCCEED()	✓Complete

4.2 Not Implemented / Deferred

Component	Original Plan	Reason	Target
INFINITY / NaN auto-generation	Planned for Iteration 2	Requires function body analysis to detect guards. Beyond current parser scope	Iteration 3
Multi-param interaction tests	Not planned	Generator tests parameters independently; cases where two params interact (e.g. min_val > max_val in clamp) require semantic understanding of the function contract	Future
Header file (.h) parsing	Mentioned in Vision Document	Deprioritised in favour of deeper type support in this iteration	Iteration 3
Extension unit tests (80%+ coverage)	Planned for Iteration 2	Development time directed toward feature delivery	Iteration 3

4.3 Iteration 2 Summary Metrics

- Total TypeScript source files modified: 3 (boundaryValues.ts, testGenerator.ts, expectedValuesWebview.ts)
- New supported C parameter types: int[] arrays, int* pointers, struct, size_t, char strings

- Functions validated end-to-end: 16 (sum_array, max_array, find_first, scale_value, distance, rect_area, rect_perimeter, rect_contains, safe_divide, safe_strlen, count_char, clamp, add, multiply, absolute_value, weighted_average)
- Extension version: 2.0.3

5. Use of Gen-AI Tools

Generative AI tools were used during Iteration 2 in the following ways:

5.1 GitHub Copilot (VS Code Extension)

- **Code autocompletion**: Copilot was used for repetitive boundary catalog entries for new types (char, size_t, struct field patterns) in boundaryValues.ts.
- **Webview tab structure**: Boilerplate for the new Custom Tests and Preview tabs in expectedValuesWebview.ts was initially drafted with Copilot assistance, then reviewed and tested manually.
- **Per-field struct input rendering**: The per-field struct input logic in the webview was initially drafted with Copilot assistance and debugged manually.

5.2 Claude (Anthropic)

- **Iterative test review**: Generated tests were reviewed across 16 functions over multiple iterations, identifying bugs such as void result = f() in custom templates, INT_MIN misclassification as overflow in absolute_value, overflow comment copy-paste from sum_array into find_first, and wrong expected values for the max_array sentinel return.
- **Boundary value reasoning**: Provided boundary value reasoning for float (INFINITY, NaN, -0.0f) and size_t (unsigned — no negatives) that informed design decisions in boundaryValues.ts.

5.3 Important Notes on AI Usage

- All AI-generated code was **reviewed, tested, and understood** before inclusion in the project.
- AI tools identified bugs that were then fixed by the developer in the actual TypeScript source.
- The core design decisions — neutral operand pattern, struct boundary strategy, separate template code path isolation — required manual reasoning and were informed but not generated by AI.

6. Revised Iteration Plan

Iteration 1: Foundation ✓ (Weeks 3–6) — COMPLETED

All planned deliverables achieved plus additional features (global variable support, webview UI, cross-platform build).

Iteration 2: Complex Types ✓ (Weeks 7–10) — COMPLETED

All five goals delivered: pointer/array/struct support, smarter boundary values, test density configuration, webview UI improvements, and Problems panel error reporting. 16 functions validated end-to-end against the generator across multiple iterations of feedback.

Iteration 3: Production Polish (Weeks 11–14)

Goal: Marketplace-ready extension

Revised Deliverables:

- Error recovery with helpful messages for malformed C code
- VS Code settings panel (boundary customisation, output paths)
- INFINITY / NaN float boundary auto-generation (deferred from Iteration 2)
- Header file (.h) parsing support (deferred from Iteration 2)
- Extension unit tests targeting 80%+ coverage of parser and generator modules
- Performance optimisation: 1000+ tests in under 2 seconds
- Comprehensive documentation: installation guide, user manual, API docs
- Beta testing with 5+ users and feedback incorporation
- VS Code Marketplace package (.vsix) published

7. References

[1] Google (2025) googletest/README.md at main, GitHub. Available at: https://github.com/google/googletest/blob/main/googletest/README.md [Accessed: 28 January 2026].

[2] ISO/IEC 9899:2011 (C11 Standard) — Section 6.5: Expressions. Undefined behaviour for signed integer overflow.

[3] Wikipedia (2026) 'Boundary-value analysis', Wikipedia. Available at: https://en.wikipedia.org/wiki/Boundary-value_analysis [Accessed: 28 January 2026].

[4] Microsoft (2026) VS Code Extension API — Webview. Available at: https://code.visualstudio.com/api/extension-guides/webview [Accessed: March 2026].

[5] CMake (2026) CMake Documentation. Available at: https://cmake.org/documentation/ [Accessed: March 2026].