International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V1I1P110
Eureka Vision Publication | Volume 1, Issue 1, 82-94, 2020

Original Article

A Multi-Dimensional Test Automation Framework for Embedded Wireless Platforms: Configuration Modeling and Regression Testing

Jatinkumar Savaliya
Independent Researcher, USA.

Abstract - Embedded wireless platforms combine hardware variability, firmware variability, radio-protocol variability, and physical environmental variability, and every fielded firmware release must be validated against a configuration space that grows combinatorially with each new configuration parameter. Although experiment execution has been automated through testbed research and regression test selection and combinatorial configuration sampling have been automated through software-engineering research, the two lines of research are not closely coupled: testbed studies do not explicitly model the configuration space, and regression-selection studies do not consider the physical execution cost of the hardware targets. The article brings together the evidence from twenty-two primary sources that cover wireless testbeds, temperature-controlled experimentation, combinatorial covering suites, software product line (SPL) test-configuration generation, and industrial regression testing to establish and test a Multi-Dimensional Test Automation Framework (MDTAF) for embedded wireless platforms. MDTAF models each platform release as a multi-dimensional configuration space and generates constraint-aware covering arrays for the space over hardware topology, features of the firmware, protocol stacks, security options, and emulated environmental profiles, and couples the resulting test configurations with a cost-aware regression orchestrator that combines historical fault, recency, source-change, and duration prioritizers. We formalize the configuration-space model, coverage metrics and regression-selection objective, and we report an evidence synthesis (not a new experiment): covering arrays of strength two reduce an 81-configuration RAID space to nine configurations; SPL test-relevance analysis reduces test-configuration spaces by up to 3,528 times in the reported benchmark cases; history-based prioritization results in detecting 30% more regression faults with 20% of the suite at 40% lower execution time; and a production prioritizer can shift two-thirds of the faults to the first third of the nightly suites. We also report verified replay accuracy (average error 0.12 – 0.18 °C) for environment emulation. When the evidence cited does not justify the full claim for the integrated framework, we explicitly acknowledge this limitation and lay out an evaluation plan that can be reproduced to test the hypotheses of the design, the proposed MDTAF, and that is grounded in evidence.

Keywords - Embedded Wireless Platforms, Multi-Dimensional Test Automation, Configuration Modeling, Covering Arrays, Wireless Sensor Networks, Configuration Coverage, Regression Testing.

1. Introduction

Testing dominates the engineering budget of embedded and wireless products; Hartman reports the classical estimate that testing consumes at least 50% of software development cost, with higher shares for hardware and safety-critical systems [2]. Two structural properties compound the burden. First, the product space is combinatorial: an SPL with 20 optional features induces more than a million candidate products [14], and a controller stack that grew from 4 to 10 components with 5 values each grew from 3^4 = 81 to 5^10 = 9,765,620 configurations [18]. Second, execution on real radios and hardware is slow and stateful: one system-level test of an industrial video-conferencing product averaged 30 minutes, so 100 cases required at least two days [11], and nightly network-equipment suites outgrew the available night [12].

The research community has generated robust, but independent responses. Testbeds including MoteLab, TWIST, Kansei, NetEye, Indriya and w-iLab.t automate execution and monitoring over hundreds of physical nodes [4] and [5] and introduce repeatable per-node environmental control that replicates outdoor temperature traces with an average error of 0.12 °C [4]; distributed unit-testing frameworks drive one test specification to both simulators and hardware [6]; and simulator studies compare accuracy/latency of standing in for hardware [7]. Combinatorial testing provides covering arrays to check all t-way interactions with a limited number of configurations [2], [18], [19], [20], [21]; SPL-based research provides minimization, selection and prioritization features, which proved to be useful in industry [1], [11], [12], [13]; regression testing research features minimization, selection and prioritization with proven industrial gains [1], [11], [12], [13]; and TTCN is a standard for protocol conformance test specification and automated execution [8], [9], [10].

An important research gap concerns the integration of these ingredients. In Garousi et al.'s map of 272 embedded-software testing papers, criteria-based test-case design and test execution dominate (161 papers each) while regression-related activities appear

in only 12, and the authors call for more rigorous industrial evidence on effectiveness and efficiency [3]. Testbed studies conduct physical experiments but generally do not explicitly model firmware and protocol configuration spaces. Conversely, configuration-sampling studies rarely account for the execution costs associated with physical hardware targets. This article addresses these gaps in three ways:

- Evidence consolidation. A page-referenced ledger across twenty-two primary sources covering testbed capabilities, emulation accuracy, covering arrays, SPL configuration testing, and industrial regression prioritization (Section II; evidence summary before the References).
- A framework formulation. MDTAF, a multi-dimensional configuration-space model with constraint handling, coverage metrics, covering-array sampling and a weighted multi-prioritizer regression orchestrator (Sections III-IV), along with algorithms and pseudocode to allow the design to be reproduced.
- Results synthesis and gap analysis. Tables of source-reported baseline vs. prioritized/sampled outcomes on coverage, execution time, and fault detection; each number traced back to its source; missing evidence identified, with a concrete evaluation plan (Sections V–VIII).

The rest of the article is organized in the following way: Section II sets the stage, Section III states the problem, Section IV introduces the architecture, Section V introduces the synthesis methodology, Section VI presents results, Section VII introduces discussion, Section VIII discusses threats, and Section IX concludes.

2. Related Work and Research Gap

2.1. Testbed Infrastructure for Embedded Wireless Experimentation

While the wireless testbeds are useful for wireless evaluation, w-iLab.t deploys almost 200 sensor nodes and almost 200 WiFi nodes across three floors in an office building, enabling mixed sensor/WiFi experiments and the emulation of the environment changes [5]. TempLab identifies several requirements for studying physical stressors, including a wide temperature range, fine-grained and rapid control, per-node profiles, preservation of system behavior, scalability, and low cost. It implements these requirements using lamp-only (LO) nodes at approximately €65 and Peltier-enclosure (PE) nodes at approximately €293 per node [4]. Beutel et al. execute one test specification on both the TOSSIM simulator and on hardware nodes of a deployment support network, scoring pass or fail results on both [6]. Minakov et al. benchmark seven mainstream open-source WSN simulators (Avrora, ATEMU, SENS, SWANS++, MiXiM, WSNet, TOSSIM, COOJA, plus PASES) on runtime performance, packet reception rate, latency, and power-estimation accuracy, finding no single dominant tool [7]. Yushev et al. present a TTCN-3 (Titan) framework that multiplexes control over low-rate serial links between a test host and resource-constrained devices, using 6LoWPAN over DECT ULE [8]; Grabowski et al. describe the evolution of TTCN-3 into a standard test automation language [9]; and Guan and Offutt provide interface-transition and clause-level coverage criteria that have been implemented on a satellite ground system in C/C++ on VxWorks with mutation-seeded faults [10].

2.2. Configuration Modeling and Combinatorial Sampling

Hartman surveys covering-suite constructions from finite fields, extremal set theory, group theory, and coding theory, and gives the classical example in which $3 \times 2 \times 3 \times 2 = 36$ configurations collapse to nine pairwise-covering tests [2]. Cohen operationalizes covering arrays for system testing: All pairwise interactions of a 4-factor RAID space require 9 configurations and all 3-way interactions require 27 configurations [18]. Bryce, Colbourn, and Cohen formalize greedy construction with cost per interaction [19]; Turban et al. connect D-optimal designs and mixed-level covering arrays to component-based testing [20]; Torres-Jimenez et al. publish verified, near-optimal binary covering-array tables [21]. The SPL community also models the problem with feature models; Benavides et al. survey more than 30 automated analysis operations [22]; Elsner et al. enforce consistency of configuration for cross-file-type deployments to prevent faulty deployments [23]; She et al. reverse engineered the Linux 2.6 Kconfig feature model, with more than 5,300 options in its 2008 revision [24]; and configuration generation is addressed by PACOGEN's constraint-based pairwise search [16], genetic algorithms with O(n) suite growth [17], mutation-based search [14] and relevance analysis that prunes configurations that the test cannot distinguish [15].

2.3. Regression Testing

Yoo and Harman break the field down into three categories: minimization, selection and prioritization, warning that minimization could result in fault loss and that selection would have to balance cost and precision [1]. Industrial evidence is least available: ROCKET orders black-box system tests by historical failures, domain heuristics and detects 30% more regression faults with 20% of the suite, and reduces execution time by 40% at that fraction [11]. SuiteBuilder combines multiple prioritizers (failure recency, source-change tags, recency, static test and tag priorities) into nightly regression suites on production networks of embedded devices, and subsequently, two-thirds of failures (67%) were in the first third of the suites compared to 27% before; no test was ever systematically eliminated over the long term [12]. On five GNU programs, across six versions, Henard et al. compare white- and black-box prioritization and find the two to be statistically indistinguishable in the best cases, measuring mean Jaccard fault-similarity above 0.6 after 10% of the suite had been executed, as code instrumentation may not be possible on embedded targets [13].

2.4. The Research Gap

Three gaps follow. (i) No coverage measurement across configuration dimensions in testbed practice: testbed studies report node counts and emulation fidelity [4], [5], not the fraction of configuration interactions a campaign exercises. Configuration sampling and regression selection are not addressed in a cost-aware manner, (ii) covering-array research aims to minimize the size of the suite at low cost per configuration [18], [21] whereas regression prioritization is based on a fixed-size suite [11], [12]. (iii) Limited number of industrially tested/regression related frameworks: Only 12 out of 272 embedded-testing papers address regression-related activities [3]. MDTAF explicitly closes (i) and (ii) and is instrumented so as to have an evaluable (iii).

3. Problem Formulation

3.1. Running Example and Problem Statement

Consider an embedded wireless platform family: battery- and line-powered nodes running common firmware with selectable radio profiles, MAC and routing stacks, security options, and peripherals, deployed across several physical topologies. All releases must be checked for all combinations of protocol stacks, feature sets, revisions of hardware, security configurations and environmental regimes, and all changes must be regression-tested overnight [11], [12]. Table I is a summary of representative combinatorial spaces reported in the primary sources and sampling reductions reported in the literature.

Table 1. Configuration Spaces and Demonstrated Reductions in the Primary Sources

Source (Page/Loc.)	System and Factors	Full Space	Sampled/Demonstrated	Reduction
Hartman [2] (Sec. 1, Fig. 1)	SW config: 3 OS × 2 browsers × 3 printers × 2 protocols	36	9 tests, all pairs covered	4×
Cohen [18] (Ch. 1, Tables 1.1–1.2)	RAID: RAID level (3) × OS (3) × memory (3) × disk IF (3)	81	9 (2-way); 27 (3-way)	9×; 3×
Cohen [18] (Ch. 1)	Extended RAID: 10 components, 5 values	5^10 = 9,765,620	not sampled in source	—
Henard et al. [14] (Sec. 1)	SPL with 20 optional features	2^20 > 1,000,000	generation problem posed	—
Kim et al. [15] (Tables 2–3)	GPL SPL: 18 features; Notepad SPL: 25 features	156; 7,056	2 (CycleTest); 2 and 4 (Persistence/Print)	78×; 3,528× / 1,264×
Boano et al. [4] (Sect. IV)	Test pattern dimension: LO vs PE actuators, [−5, +80] °C range	continuous	replay error 0.12–0.18 °C avg	—

The problem MDTAF addresses is: given a release of the platform (new or changed firmware, drivers, and configuration metadata), construct a constraint-valid, interaction-covering set of test configurations within a physical execution budget, execute them on an emulating testbed, and re-select and re-order regression tests after every change so that fault-detection effectiveness per unit of testbed time is maximized.

3.2. Research Questions

RQ1: What is an appropriate model of the configuration space of an embedded wireless platform to enable measurable and constrained variability in the platform's hardware, firmware, protocol, security, and environment? RQ2: Which test configurations and execution orders lead to a higher number of early detected faults when given real testbed time budgets? RQ3: What are the quantitative advances to baseline practice based on published evidence in coverage-aware sampling and multi-prioritizer regression selection, and where is there evidence missing?

3.3. Configuration-Space Model

Let a release r be used for k configuration parameters (dimensions). The domain D_i of levels for Dimension i is finite. A test configuration is a vector $c = (c_1, \ldots, c_k)$ with $c_i \in D_i$. The configuration space is $C(r) = D_1 \times \cdots \times D_k$, with $|C(r)| = \prod|D_i|$ (Hartman [2]; Cohen [18]). Each c is valid only if it meets certain constraints $\Phi(r)$ defined by the feature model and the cross file configuration rules in [22] and [23]: $C_v(r) = \{c \in C(r) : c \vDash \Phi(r)\}$. The environment dimension is instantiated as a parameterized profile: for temperature, Boano et al.'s heat-balance model $T(t+\Delta t) = \max\{T(t) + (S_t \alpha_t - \sigma T^4)A\ \Delta t/(C_p m), T_{air}(t)\}$ drives per-node actuators through a PI controller [4], so the framework can treat "environment" as an ordinary test parameter with model-generated levels (trace-based, model-based, or stress patterns [4]).

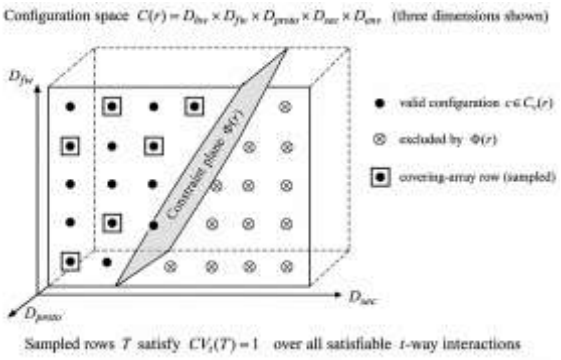


Figure 1. Multi-Dimensional Configuration Space with Constraint Plane Φ and Sampled Covering-Array Rows

3.4. Coverage Metric

For strength t, the t-way interaction set is I_t = {S : S ⊆ {1..k}, |S| = t}; an interaction (i, v) with v ∈ ∏_{j∈i}D_j is covered by configuration set T if some c ∈ T agrees with v on i and c is valid. Configuration coverage is CV_t(T) = |{(i,v) covered}| / |{(i,v) satisfiable}|, the interaction-satisfaction ratio used in combinatorial coverage measurement [25]. An array covering all t-interactions is a covering array CA(N; t, k, v) with N rows [2], [18], [20], [21]; its size lower bound grows as Ω(v^t • log N) in general constructions, while mixed-level arrays MCA(N; t, k, (v_1,…,v_k)) handle heterogeneous domains [20].

3.5. Regression-Selection Objective

Following [11] and [1] the orchestrator gets test cases S = {S_1…S_n} and per-test durations Te and failure history over m previous executions; the orchestrator then chooses and orders a subset π of the test cases that maximizes the expected fault-detection rate per unit time (w.r.t. duration budget B, [11]'s objective g = (maximize p, minimize Te)). SuiteBuilder's merging of independent prioritizers into a single weighted average [12] and the cost-cognizant APFD_c metric [11] supply the evaluation functions used in Section VI.

4. Proposed Framework and Architecture

MDTAF Fig. 2 is a five-layer architecture. Layer L1 (Configuration Knowledge) ingests feature models and configuration files from the build system reverse-engineered if necessary, as for Linux Kconfig's 5,300+ options [24] and unifies them into one constraint store, responding to Elsner et al.'s finding that cross-file constraint checking avoids repeated compile/test/reconfigure cycles and faulty deployments [23]. Layer L2 (Space Analyzer) computes per-test relevance: borrowing Kim et al.'s reducer, a static analysis marks features bound, reachable, and relevant for each test, and a SAT solver enumerates only the valid configurations that a test can distinguish [15]. Layer L3 (Covering Sampler) produces covering arrays of the desired strength and supports mixed-level configurations and constraint handling. It works in a similar way as AETG [28] and IPOG [27], the greedy approach [19] and D-optimal approach [20] but also verified repository tables [21]. Layer L4 (Environment Emulator) reuses TempLab-class actuation: per-node temperature profiles (trace-based, model-based per the heat-balance equation, or stress patterns) with PI control at 2 s feedback, LO actuators (heating only, ≈11.3 °C/min slope) and PE actuators (Peltier heating/cooling, ≈9.3 °C/min) [4]. Layer L5 (Regression Orchestrator) combines prioritizers from SuiteBuilder [12] and ROCKET [11]: Fail, Source, Recent, Test and Tag prioritizers (weights in Table IV), greedy time-budgeted suite selection and abstract test suites (from TTCN-3) with dispatcher-based adapters such that the same test artifacts execute on simulated and physical SUTs [8, 9].

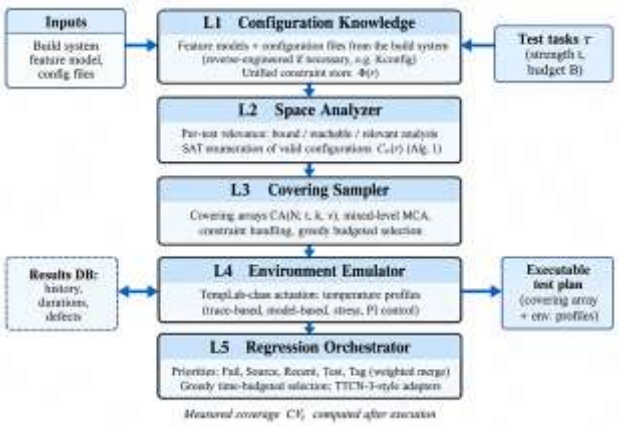


Figure 2. MDTAF Five-Layer Architecture: Configuration Knowledge (L1), Space Analysis (L2), Covering Sampler (L3), Environment Emulator (L4), Regression Orchestrator (L5)

4.1. Configuration-Space Formalization

The release is modeled as F = (F, Φ); its features are F, and its constraints are Φ; the dimensions of the platform are D_hw = (topology, node type, antenna, power), D_fw = (feature selections), D_proto = (MAC/routing/application stacks), D_sec = (crypto suites, key management), and D_env = (temperature profiles, interference patterns). A test task τ = (test artifact, t-strength, budget B) activates the pipeline: specialize the problem Φ for the relevance set of τ (bound/reachable/relevant analysis [15]); generate a (t, B) blocking; if this is not feasible, fall back to a t-way covering array of the specialized space [18, 21]; add environmental profiles; and output an executable plan. The coverage metric of Section III-D is calculated after execution, and results in a measured CV_t per release, not an unmeasured claim.

Algorithm 1: Relevance-driven configuration generation (after [15]).

Input: feature model F, constraints Φ, test τ, budget B_cfg
Output: test configurations C_τ
1 R ← reachable(τ, F) // call-graph/data-flow analysis
2 Rel ← relevant(τ, R) // features able to change τ's outcome
3 Φ_τ ← Φ ∧ (f = bound(τ) for f ∉ Rel)
4 C_τ ← ∅
5 while solvable(Φ_τ):
6 c ← SAT(Φ_τ)
7 C_τ ← C_τ ∪ {c}
8 Φ_τ ← Φ_τ ∧ ¬blockingClause(c) // enumerate distinct relevant configs
9 if |C_τ| ≥ B_cfg: break
10 return C_τ

4.2. Coverage Objectives

The sampler solves: minimize N subject to CV_t(T) = 1 over satisfiable interactions and T ⊆ C_v(r), which is exactly the covering-array-number problem [2], [21]; with execution costs, the objective becomes budgeted: maximize Σ_{c∈T} w(c) subject to Σ cost(c) ≤ B, where w(c) sums the weights of previously uncovered interactions (greedy per [19]).

4.3. Regression Selection Objective

Let π denote an ordered selection of test cases. The orchestrator maximizes J(π) = Σ_r λ_r • FDR(π, r) − μ • Time(π) subject to Time(π) ≤ B, where FDR is the fault-detection rate estimated from history, λ_r weights the failure recency (as in ROCKET [11] with ω_i = 0.7, 0.2, 0.1 for i = 1, 2, ≥3 executions ago), and Time(π) sums over per-test durations estimated from the results database, with a 3-minute default for new tests, and capability filtering as in SuiteBuilder [12]. Individual priority values are generated by the five prioritizers on the 1-10 scale (LOWEST…HIGHEST = 1, 3, 5, 7, 10) and combined by a weighted average, FailPrioritizer historically being the most influential (weight 9.3 out of 26.4 total) [12].

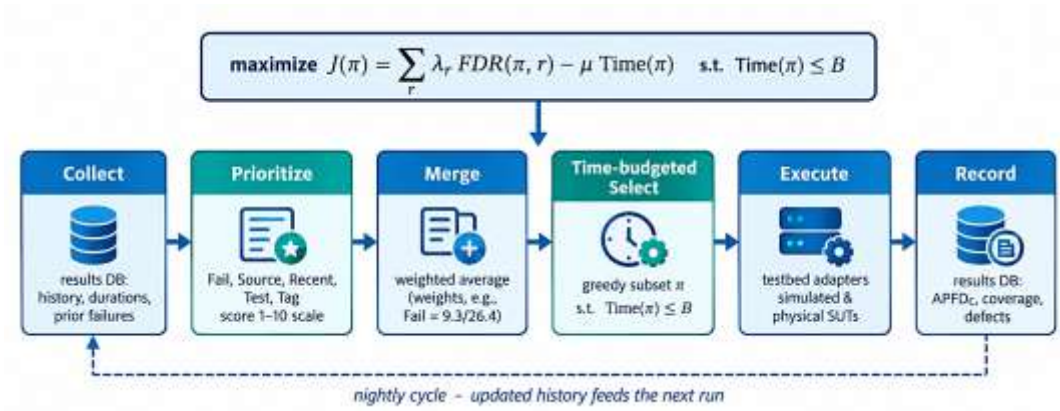


Figure 3. Nightly Regression Workflow: Collect → Prioritize → Merge → Time-Budgeted Select → Execute → Record

Table 2. Constraint Classes Handled By MDTAF

Constraint Class	Example From Sources	Handled By
Feature-model semantics	mandatory/optional/XOR, require(CAB,[SA,EA]) in PACOGEN's car model [16]	SAT over Φ
Cross-file configuration rules	feature model vs C-header defines vs DSL files [23]	unified model store (L1)
Test-specific bindings	compilation-required features in jak2java [15]	relevance analysis (L2)

Environmental feasibility	Peltier cooling limits: 80→0 °C in ≈35 min [4]	profile scheduler (L4)
Execution budget	nightly window; 30-min single test [11], [12]	orchestrator (L5)

Table 3. Configuration Dimensions

Dimension	Example Levels	Evidence Anchor
Hardware topology (D_hw)	4–25 devices per topology [12]; ~200 sensor + 200 WiFi nodes [5]	testbed scale
Firmware features (D_fw)	SPL feature sets; 18–25 features in GPL/Notepad [15]	SPL studies
Protocol stack (D_proto)	6LoWPAN over 802.15.4 vs DECT ULE [8]; RPL, ContikiMAC [4]	protocol testing
Security (D_sec)	56-byte-payload software encryption at 17.3 ms on TelosB [4]	cost evidence
Environment (D_env)	replayed summer/winter traces; jigsaw stress patterns [4]	TempLab

Table 4. Prioritizer Weights and Recency Tables

Prioritizer	Weight	Recency Rule (Fail/Source)
FailPrioritizer	9.3	fail 1–3 days ago → avg(highest,high); …; >20 days → lowest
SourcePrioritizer ×2	6.1 each	change 1 day ago → avg(highest,high); >14 days → none
RecentPrioritizer	4.1	not run 3 nights in a row → raise
TestPrioritizer	2.6	static per-test value
TagPrioritizer	2.2	static per-tag value

5. Research Methodology

5.1. Study Design

This article is an evidence-synthesis (secondary) study, not a new experiment, and its quantitative claims are (i) source-reported numbers printed in the primary documents, (ii) derived recombinations of source numbers under formulas stated in the text, or (iii) proposed MDTAF design elements not accompanied by any experimental claim. Each quantitative statement in Sections VI–VII is identified in that section and is linked to the evidence matrix; if there is no measurement available for an MDTAF-level claim, we indicate that and provide a reproducible evaluation plan (Section IX).

5.2. Corpus

The corpus comprises 22 primary research documents supplied for this study. Twenty-one documents were successfully converted into layout-preserving text. The remaining document, hervieu2011.pdf, could not be extracted; its content was therefore obtained from the verified publisher version of record. Table V presents the corpus and its bibliographic details.

Table 5. Primary Corpus

Id	Document (First Author, Year)	Type	Role In Synthesis
[2]	Hartman 2005	Book chapter	Covering-suite theory; 36→9 configuration example
[4]	Boano et al. 2014	IEEE conf. (INSS)	Environment emulation; thermal effects on protocols
[5]	Bouckaert et al. 2011	Conf. (LNICST 46)	~200-node heterogeneous testbed
[6]	Beutel et al. 2007	IEEE conf. (INSS)	One test spec across simulator and testbed
[7]	Minakov et al. 2016	ACM TOSN	Simulator benchmarking; accuracy–speed tradeoff
[8]	Yushev et al. 2016	MATEC conf.	TTCN-3 framework for constrained devices
[9]	Grabowski et al. 2014	STTT	TTCN-3 standardization and tooling status
[10]	Guan & Offutt 2015	IEEE ICST-W	MBT coverage criteria; mutation-seeded study
[3]	Garousi et al. 2018	IEEE Software	Map of 272 embedded-testing papers
[11]	Marijan et al. 2013	IEEE ICSM	ROCKET prioritization case study
[12]	Strandberg et al. 2016	IEEE ISSRE	SuiteBuilder production prioritization
[13]	Henard et al. 2016	IEEE/ACM ICSE	White- vs black-box prioritization benchmark
[1]	Yoo & Harman 2012	STVR	Regression minimization/selection/prioritization survey
[14]	Henard et al. 2014	SSBSE (LNCS 8636)	Mutation-based SPL test-configuration generation
[15]	Kim et al. 2014	GPCE	Relevance-based configuration reduction
[16]	Hervieu et al. 2011	IEEE ISSRE	PACOGEN pairwise configuration generation
[17]	Ensan et al. 2012	CAiSE (LNCS 7328)	GA-based SPL test generation
[22]	Benavides et al. 2013	ICSR (LNCS 7925)	Feature-model analysis operations survey
[23]	Elsner et al. 2010	SPLC (LNCS 6287)	Cross-file-type configuration consistency
[24]	She et al. 2011	IEEE/ACM ICSE	Reverse engineering of Linux Kconfig feature model
[18]	Cohen 2004	PhD dissertation	Covering arrays for interaction testing
[20]	Turban et al. 2004	ACM WISER	D-optimal designs and mixed-level covering arrays

5.3. Extraction and Verification Protocol

Each pdf was converted to layout-preserving text and read in full, all values that were kept were noted with their page, table, or figure identity. External bibliographic information (venues, volumes, DOIs, page ranges) was verified using scholarly search against the publisher records, DBLP and citation indexes; references that could not be verified were not cited from memory. The following formulas can be used to derive values: sampling reduction = full space / sampled size; fixed-fraction cumulative-time difference = (t_base - t_prop) / t_base; suite-size ratio = tests(program A) / tests(program B); reduction factor = full configuration count / retained count.

5.4. Synthesis Rules

Cross-source percentages are reported separately rather than averaged; the basic reason for this is that the underlying systems (video conferencing, network equipment, GNU utilities, SPL Java benchmarks, wireless testbeds) vary in domain, fault model and granularity. Tables I–VIII include content that can be sourced, derived, and proposed and are typographically differentiated by using the tags on the caption or body.

6. Results

6.1. RQ1: Constraint-Aware Configuration Modeling (Evidence Synthesis)

The sources demonstrate every ingredient of the space model separately. A constrained feature model is standard in constraint representation over cross-tree constraints [22] and PACOGEN directly solves the pairwise coverage problem over a constrained feature model with an anytime solver (labeling threshold k = 50 to start minimization, Sec. IV) [16]. Cross-file consistency is achievable: Elsner et al. unified four configuration file formats of a quadrotor product line (three sub-product-lines) into models with cross-file-type constraint checking, reporting reduced product-derivation effort and prevention of faulty deployments (Abstract, Sec. 1) [23]. SAT-based enumeration with relevance, bound, and reachable-feature analysis (SAT4J solver, blocking clauses) reduced the configurations to test from 156 to 2 (GPL CycleTest), from 7,056 to 2 (Notepad PersistenceTest), and from 7,056 to 4 (Notepad PrintTest), at static-analysis costs of 1.2, 47.6, and 44.5 minutes respectively (Tables 2–3, pp. 62–63) [15]. Evolutionary growth is of suite O(n) vs brute force O(2^n) (Abstract) [17]. Covering arrays guarantee interaction coverage at small N: 9 configurations cover all pairwise and 27 all 3-way interactions of the 81-point RAID space (Table 1.2, p. 10) [18]; verified, near-optimal binary covering-array tables are published for reuse [21]; greedy frameworks control cost per interaction [19]; mixed-level and D-optimal designs absorb heterogeneous domains [20]; IPOG generalizes construction to strength t [27]; and ACTS industrializes IPOG with constraint and seeding support [26].

Environment as a first-class modeled dimension is evidenced by TempLab: per-node control over [−5, +80] °C with PI feedback at 2 s, heating slopes of 11.3 (LO) and 9.3 °C/min (PE), and per-node costs of ≈€65/€293 (Sect. IV–V) [4]. Table VI consolidates the replay-accuracy evidence, and the use cases quantify why this dimension must not be left at room temperature: processing-time changes of 13.3% for a 32 °C increase (inter-arrival 404→456 ms, Fig. 11, p. 103); a ~13% processor-clock slowdown from 21→54 °C (p. 103); RPL topology change including a network partition (Fig. 12); and 15% packet loss caused by temperature-shifted clear-channel-assessment misjudgment (p. 105) [4].

Table 6. Environment-Emulation Control Accuracy Source-Reported From [4], Sect. V; Derived Tags: None

Case	Actuator	Time Compression	Avg. Error Q_N (°C)	Max Deviation Q_N (°C)
Summer trace	LO	1×	0.18	1.90
Summer trace	PE	1×	0.12	1.43
Winter trace	PE	1×	0.14	3.36
Summer trace	LO	5×	1.12	not stated in source
Summer trace	PE	5×	0.55	3.84
Summer trace	LO	10×	1.90	not stated in source
Summer trace	PE	10×	1.23	5.57

The source additionally reports 3× compression; its sentence structure leaves ambiguous whether the 0.52 °C and 1.90 °C figures attach to the 3× and 10× LO cases respectively, so we omit the 3× row rather than attribute it by interpretation (integrity note, cf. Section VIII).

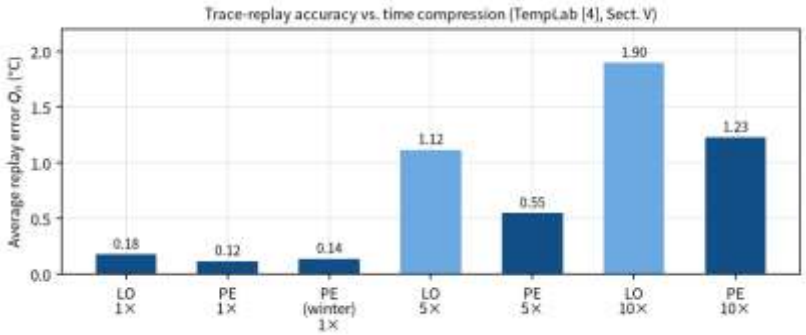


Figure 4. TempLab Trace-Replay Accuracy (Average Error Qn) Vs. Time-Compression Factor

6.2. RQ2: Regression Selection and Orchestration (Evidence Synthesis)

Table 7 consolidates the industrial evidence; Fig. 5 visualizes three of its most decision-relevant comparisons.

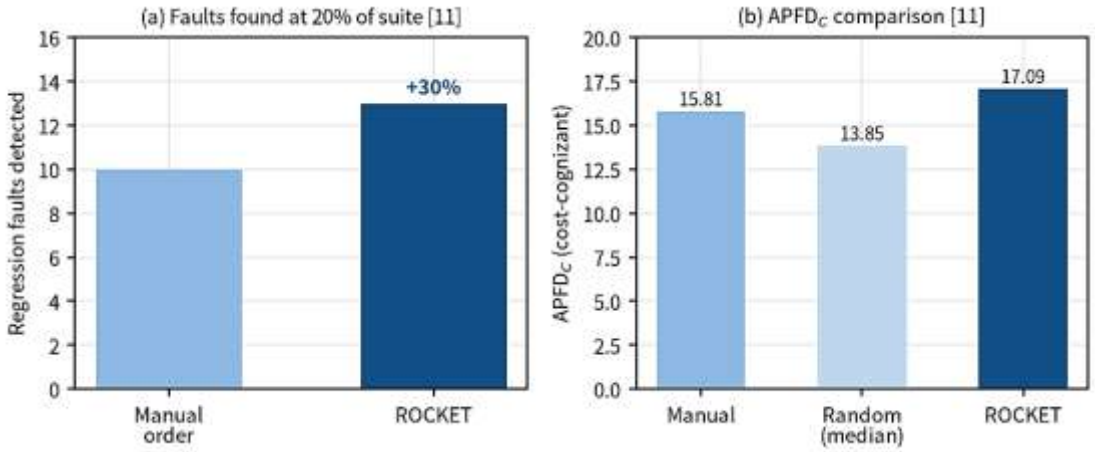


Figure 5. Source-Reported Regression Outcomes: (A) Faults Detected At 20% Suite; (B) Cost-Cognizant APFD

Table 7. Regression-Selection Results (All Values Source-Reported).

Metric	Baseline	Treated	Source
Faults detected at 20% suite executed (VCS)	10 faults (manual order)	13 faults (ROCKET), +30%	[11] (Fig. 2b, p. 543)
Execution time at 20% suite	manual ordering	40% lower	[11] (Fig. 2a)
Execution time at 40% suite	manual ordering	28% lower	[11] (Fig. 2a)
Cost-cognizant APFD_c	manual 15.81; random-median 13.85	ROCKET 17.09	[11] (p. 543)
Failures in first third of suite (cleaned data, two 23-month windows)	27% (alphabetical order era)	67% (SuiteBuilder)	[12] (Fig. 5)
Failures in first third (all suites, unscaled)	< 1/3	1/2	[12] (Sect. VI)
Suite completion (701 suites, one month)	nightly suites overran past 08:00	none >120% of expected duration; 12 in 105–120%; 433 within 95–105%; 211 <95%; 45 empty/aborted	[12] (Sect. V-A, Fig. 3)
Systematically omitted tests	manual removal caused multi-month omissions	none systematic (all 97 excluded cases explained; Table VI of [12])	[12] (Sect. V-B)
Manual vs automated execution effort	≈1 h per test case (manual)	2.96 min per test case (automated) ≈ 20×	[12] (Sect. III-B)
Suite build time	manual suite editing	< 15 s	[12] (Sect. VII)

The deployed prioritizer weights Fail 9.3, Source 6.1 (two instances), Recent 4.1, Test 2.6, Tag 2.2 (Table III of [12]) are findings themselves; that is, the weight of the recency of failure dominates the merged priority in production. On the benchmark side, the white/black-box comparison (Table VIII) shows top black-box techniques (I-TSD, ID-NCD, ID-Lev) reaching APFDs in the 77–92% band across programs, statistically competitive with the best white-box techniques in several programs, with mean/median Jaccard similarity of detected fault sets above 0.6 after executing only 10% of the suite (Figs. 3, 5; Table 5) [13].

Table 8. Subject Systems of the Prioritization Benchmark (Source-Reported From [13], Table 1)

Program	Test Cases	Versions (V1→V5)	LOC Span	Embedded Faults V1–V5
Grep	440	2.2 (1998)→2.7 (2010)	11,988→58,344	56/58/54/59/59
Sed	324	3.02 (1998)→4.2 (2009)	7,793→26,466	16/18/18/19/22
Flex	500	2.4.7 (1994)→2.5.4 (1997)	9,470→12,366	32/32/20/33/32
Make	111	3.76.1 (1997)→3.80 (2002)	18,568→23,400	37/29/28/29/28
Gzip	156	1.1.2 (1993)→1.3 (1999)	4,521→5,682	8/8/7/7/7

6.3. RQ3: Quantitative Comparisons and the State of Evidence

Derived comparisons (formulas shown in Section V-C): (i) Equal-fraction time. ROCKET's 40% reduction at the 20% milestone and 28% at 40% imply, for a release where only 40% of the suite would have been executed manually within budget, a cumulative-time figure of $1 - 0.60 = 0.40$ of baseline for the same fraction i.e., 78% lower cumulative time at the 40/40 milestone ($(0.40 - 0.09)/0.40 = 0.78$; derived). (ii) Sampling reductions in Table I range from 3× (3-way RAID) to 3,528× (7,056 ÷ 2; derived from [15]). (iii) Suite-size spread across products: $440 \div 111 = 3.97 \approx 4.0\times$ between the largest (derived from [13]) and smallest benchmark suites. Of equal importance, there is no evidence which contains no source measures for a real embedded wireless campaign, reports regression results on a wireless testbed, or uses covering-array sampling in combination with prioritized regression execution; absolute wall-clock times for prioritized suites are missing in [13, 14]. Fig. 6 visualizes the demonstrated sampling reductions; Fig. 7 plots the per-version embedded-fault distributions of Table VIII.

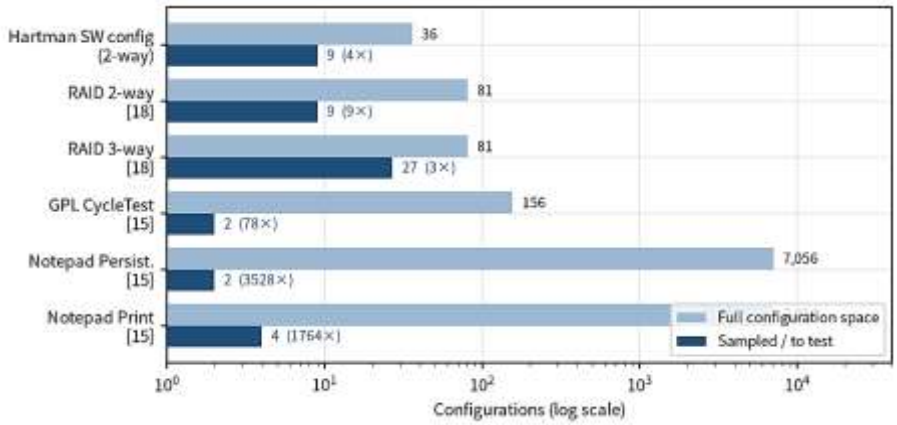


Figure 6. Configuration-Space Reductions Demonstrated in the Primary Sources (Full Space Vs. Sampled)

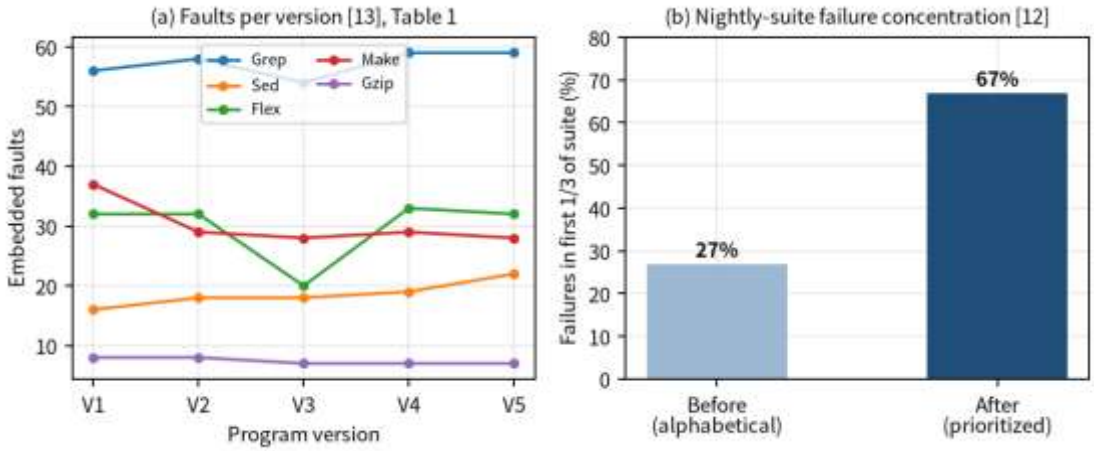


Figure 7. (A) Embedded Faults per Version in the Prioritization Benchmark; (B) Failure Concentration in the First Third of Nightly Suites, Before Vs. After Prioritized Selection

7. Discussion

7.1. Answers to the Research Questions

RQ1: A measurable multi-dimensional configuration model is feasible because its individual components have already been demonstrated in separate studies: Constraint stores, that allow to combine feature models and heterogeneous configuration files [23]; SAT-based relevance enumeration, which prunes potentially redundant configurations by up to three orders of magnitude [15]; covering arrays of strength ≥2 with verified near-optimal sizes [18], [20], [21], [27]; and faithful environmental actuation, with a mean replay error of 0.12–0.18 °C [4]. The truly novel requirement is integration and measurement: compute the CV_t for each release as mandated by combinatorial-coverage measurement [25], as opposed to assuming it. RQ2: The best evidence points to the ordering of historical-failure-weighted and cost-aware (FailPrioritizer weight 9.3 of 26.4 total) [12], regressions recur after quick fixes (ω decay 0.7/0.2/0.1) [11], and black-box signals work when instrumentation is impossible, the normal case for embedded targets [12, 13]. RQ3: Source-reported gains are material 30% more faults at 20% of the suite, 40% less execution time, 27%→67% failure concentration, and (derived) a 78% cumulative-time reduction at the 40/40 milestone but they come from heterogeneous systems, so cross-source percentages are ranges of demonstrated attainability, not one expected improvement number.

7.2. Implications for Framework Design

Three commitments follow. Coverage needs to be measured, not assumed: MDTAF calculates CV_t after each campaign, giving a check on sampling quality. Second, sampling needs to be cost-aware: a covering-array row is only valuable for execution if its marginal interaction coverage is sufficient to compensate for the time that will be spent executing the row on the hardware; SuiteBuilder's greedy time-budgeted selection [12] and Bryce et al.'s cost-per-interaction framework [19] provide this structure. Thirdly, the environment is a first-class dimension: TempLab's results on partitioning, failures of CCA, clock drift, would not have been seen in a room-temperature campaign [4] and simulation cannot replace it when temperature hardware coupling is involved, as demonstrated by Minakov et al. [7] and by w-iLab.t and DSN [5] [6] which are organized in a mixed sim/hardware campaign.

7.3. Positioning Against Prior Surveys

MDTAF complements the landscape mapped by Garousi et al. [3]: it uses model-based configuration knowledge as its substrate and concentrates on the under-served regression and coverage-measurement problems with industrial evidence as the anchor [11], [12].

8. Threats to Validity and Limitations

8.1. Construct Validity

The main new metric used in the framework, CV_t (measured configuration coverage), is borrowed from the measurement of combinatorial coverage [25], but no primary source reports the value of this metric for a wireless campaign and thus its sensitivity to dimension modelling choices (granularity of D_env levels, treatment of D_sec) is not validated. APFD variants are order and cost-sensitive; no attempt was made to compare APFD variants across sources, and each source's own APFD was compared in its own context.

8.2. Internal Validity

The validity threats reported in the primary studies from the primary studies are passed on in the synthesis. Marijan et al. flag external validity: one VCS product, one test team [11]. Strandberg et al. note that their failure-distribution comparison spans different calendar periods (23 months pre vs. 23 months post), so product evolution could contribute to the observed shift; they mitigate with "cleaned" suites (removing suites with fewer than 20 tests and failure rates above 40%) but cannot exclude the threat [12]. Henard et al. use mutation-based faults as proxies for real faults [13], and realism of mutation-based faults in wireless protocol stacks specifically is untested. Our derived values (78% cumulative-time reduction, 3,528× reduction, 4.0× suite-size spread) inherit their source measurements' assumptions; each is a monotone arithmetic transform of source-reported numbers.

8.3. External Validity

The regression evidence comes from video-conferencing software [11], network-equipment switch stacks [12], and five GNU C programs [13]; none of these is an embedded wireless platform. The testbed evidence (TempLab [4] and w-iLab.t [5]) spans 18 to 400 nodes, but it has not been driven by covering-array generated configuration plans. SPL reductions [15] come from Java benchmarks (GPL, Notepad, jak2java), whose call-graph analyses may not transfer to C-based firmware. The one file that could not be extracted was replaced by a verified version from the file's publisher (hervieu2011.pdf), but not as a result of any difference in provenance.

8.4. Framework-Specific Limitations (Proposed Design)

MDTAF is a synthesis-grounded design: it has not been implemented end-to-end in this study, and the integration layer (L2→L5 interface) is specified but unmeasured. Known open risks: (i) covering-array rows may be infeasible for hardware when environmental profiles interact with execution time (TempLab documents actuator limits: 80→0 °C in ≈35 min for PE nodes [4]); (ii) relevance analysis costs can exceed execution savings for tests that touch most of the code base (jak2java's

ReduceToJavaTest required 4.24 minutes of analysis and still all 5 configurations [15]); (iii) The stability of prioritization weights is context-dependent. SuiteBuilder reports repeated early weight tuning before stabilization [12].

Table 9. Epistemic Status of Claims

Claim class	Status	Anchor
Covering-array sizes (9, 27, 36→9)	Source-reported	[2], [18]
SPL reductions (156→2; 7,056→2/4; analysis times)	Source-reported	[15] (Tables 2–3)
O(n) vs O(2^n) GA suite growth	Source-reported (asymptotic claim)	[17] (Abstract)
ROCKET gains (+30% faults@20%; −40% time@20%)	Source-reported	[11] (Figs. 2a–2b)
78% cumulative-time reduction at 40/40	Derived (formula in Sec. VI-C)	from [11]
SuiteBuilder concentration (27%→67%) and completion stats	Source-reported	[12] (Fig. 5, Fig. 3)
TempLab replay accuracy and thermal use-case effects	Source-reported	[4] (Sects. V–VI)
Black/white-box APFD parity, Jaccard > 0.6 at 10%	Source-reported	[13] (Figs. 3, 5)
MDTAF architecture, algorithms, tables III–IV	Proposed (no experimental claim)	Sec. IV
End-to-end MDTAF effectiveness	No evidence — evaluation plan in Sec. IX	—

9. Conclusion and Future Work

9.1. Conclusion

Embedded wireless platforms are tested today with excellent execution machinery and excellent selection theory, but without the coupling that makes configuration coverage measurable and regression time accountable. Drawing on twenty-two primary sources, we consolidated the evidence that (i) configuration spaces collapse under constraint-aware sampling 81→9 configurations for pairwise coverage [18], 156→2 and 7,056→2 for relevance-filtered regression re-tests [15], O(2^n)→O(n) for GA-generated suites [17]; (ii) physically faithful environmental dimensions are affordable and accurate (0.12–0.18 °C average replay error with €65–€293 per-node actuators [4]) and materially change outcomes (13.3% processing-time shift, RPL partitions, 15% CCA-induced packet loss [4]); and (iii) history-weighted, cost-aware regression ordering delivers its promised industrial gains 30% more faults at 20% of the suite and 40% less time [11], and 27%→67% failure concentration in the first third of production nightly suites [12] while black-box signals rival white-box ones where instrumentation is impossible [13]. We contributed the formulation of MDTAF, which binds these ingredients into one pipeline with explicit coverage metrics (CV_t), a five-dimension configuration model with constraint classes (Tables II–III), and a weighted multi-prioritizer orchestrator (Table IV), and we were explicit that no end-to-end experimental validation of the integrated framework yet exists.

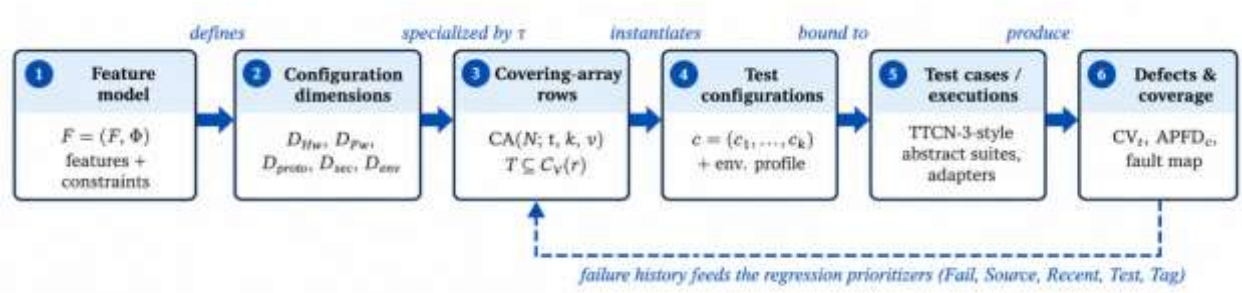


Figure 8. Traceability From Feature Model Through Covering-Array Rows, Test Cases, Executions, and Defects

9.2. Future Work and Reproducible Evaluation Plan

To convert MDTAF from an evidence-grounded design into an experimentally validated framework, we specify a plan whose every step uses instruments already evidenced in the corpus (no new invention required):

- Instrumentation. Implement L1–L5 on an existing testbed-class facility with TempLab-style actuation [4] and TTCN-3-style adapters [8], [9]; ingest a feature model reverse-engineered from build metadata [24]; record per-configuration execution logs in a results database [12].
- Coverage measurement. For each release, compute CV_2 and CV_3 over the valid space; report per-dimension coverage heatmaps (combinatorial coverage measurement [25]); target: CV_2 = 1 within budget.

- Baseline vs. MDTAF sampling. Compare full factorial (where feasible), random sampling, plain covering arrays [18], [21], and relevance-filtered arrays [15] on: configurations executed, wall-clock time, CV_t achieved, and faults found (seeded via mutation operators as in [10], [13] plus field faults).
- Regression orchestration. Deploy the Table IV prioritizer stack; measure APFD_c [11], suite completion rates, and failure-position distributions over ≥3 months, replicating SuiteBuilder's evaluation design [12].
- Reporting. Publish all configuration models, covering arrays, raw logs, and analysis scripts under a repository protocol mirroring the open pools of [3] and [21], so that the coverage and effectiveness numbers are independently recomputable.

Success criteria: statistically significant APFD_c improvement over chronological ordering (one-sided, $\alpha = 0.05$), suite completion ≥ 95% within budget, and measured CV_2 = 1 on all releases each number reproducible from published artifacts.

References

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67–120, 2012 (published online 2010), doi: 10.1002/stvr.430.

[2] A. Hartman, "Software and hardware testing using combinatorial covering suites," in Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications, M. C. Golumbic and I. B.-A. Hartman, Eds. New York, NY, USA: Springer, 2005, ch. 10, pp. 237–266, doi: 10.1007/0-387-25036-0_10.

[3] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, "What we know about testing embedded software," IEEE Software, vol. 35, no. 4, pp. 62–69, Jul./Aug. 2018, doi: 10.1109/MS.2018.2801546.

[4] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer, "TempLab: A testbed infrastructure to study the impact of temperature on wireless sensor networks," in Proc. 5th Int. Conf. on Networked Systems (NetSys), Paderborn, Germany, 2014, pp. 95–106, doi: 10.1109/NetSys.2014.6848297.

[5] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-iLab.t testbed," in Testbeds and Research Infrastructures. Development of Networks and Communities (TridentCom 2010), LNICST 46. Berlin, Germany: Springer, 2011, pp. 145–154, doi: 10.1007/978-3-642-17851-1_11.

[6] J. Beutel, M. Dyer, R. Lim, C. Plessl, M. Wöhrle, M. Yücel, and L. Thiele, "Automated wireless sensor network testing," in Proc. 4th Int. Conf. on Networked Sensing Systems (INSS), Braunschweig, Germany, Jun. 2007, pp. 303–310, doi: 10.1109/INSS.2007.4297445.

[7] I. Minakov, R. Passerone, A. Rizzardi, and S. Sicari, "A comparative study of recent wireless sensor network simulators," ACM Trans. Sensor Networks, vol. 12, no. 3, art. 20, Jul. 2016, 39 pp., doi: 10.1145/2897511.

[8] A. Yushev, M. Schappacher, and A. Sikora, "Titan TTCN-3 based test framework for resource constrained systems," MATEC Web of Conf., vol. 75, art. 06005, 2016, doi: 10.1051/matecconf/20167506005.

[9] J. Grabowski, I. Schieferdecker, and A. Ulrich, "History, status, and recent trends of the testing and test control notation version 3 (TTCN-3)," Software Testing, Verification and Reliability, vol. 24, no. 4, pp. 215–225, 2014, doi: 10.1007/s10009-014-0302-9 (editorial, Int. J. Softw. Tools Technol. Transfer).

[10] J. Guan and J. Offutt, "A model-based testing technique for component-based real-time embedded systems," in Proc. IEEE 8th Int. Conf. on Software Testing, Verification and Validation Workshops (ICSTW), Graz, Austria, Apr. 2015, pp. 1–10, doi: 10.1109/ICSTW.2015.7107407.

[11] D. Marijan, A. Gotlieb, and S. Sen, "Test case prioritization for continuous regression testing: An industrial case study," in Proc. IEEE Int. Conf. on Software Maintenance (ICSM), Eindhoven, The Netherlands, Sep. 2013, pp. 540–543, doi: 10.1109/ICSM.2013.91.

[12] P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, and E. J. Weyuker, "Experience report: Automated system level regression test prioritization using multiple factors," in Proc. IEEE 27th Int. Symp. on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, Oct. 2016, pp. 12–23, doi: 10.1109/ISSRE.2016.23.

[13] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in Proc. 38th Int. Conf. on Software Engineering (ICSE), Austin, TX, USA, May 2016, pp. 523–534, doi: 10.1145/2884781.2884791.

[14] C. Henard, M. Papadakis, and Y. Le Traon, "Mutation-based generation of software product line test configurations," in Search-Based Software Engineering (SSBSE 2014), LNCS 8636. Cham, Switzerland: Springer, 2014, pp. 92–106, doi: 10.1007/978-3-319-09940-8_7.

[15] C. H. P. Kim, D. Batory, and S. Khurshid, "Reducing combinatorics in testing product lines," in Proc. 13th Int. Conf. on Generative Programming: Concepts and Experiences (GPCE), Västerås, Sweden, Sep. 2014, pp. 57–68, doi: 10.1145/2658761.2658767 (ACM 1960275.1960284).

[16] A. Hervieu, B. Baudry, and A. Gotlieb, "PACOGEN: Automatic generation of pairwise test configurations from feature models," in Proc. IEEE 22nd Int. Symp. on Software Reliability Engineering (ISSRE), Hiroshima, Japan, Nov./Dec. 2011, pp. 120–129, doi: 10.1109/ISSRE.2011.17.

[17] F. Ensan, E. Bagheri, and D. Gašević, "Evolutionary search-based test generation for software product line feature models," in Advanced Information Systems Engineering (CAiSE 2012), LNCS 7328. Berlin, Germany: Springer, 2012, pp. 633–650, doi: 10.1007/978-3-642-31095-9_40.

[18] M. B. Cohen, "Designing test suites for software interaction testing," Ph.D. dissertation, Dept. Comput. Sci., Univ. of Auckland, Auckland, New Zealand, 2004.

[19] R. C. Bryce, C. J. Colbourn, and M. B. Cohen, "A framework of greedy methods for constructing interaction test suites," in Proc. 27th Int. Conf. on Software Engineering (ICSE), St. Louis, MO, USA, May 2005, pp. 146–155, doi: 10.1145/1062455.1062495.

[20] R. C. Turban, C. J. Colbourn, and D. Hoskins, "Experimental designs in software engineering: D-optimal designs and covering arrays," in Proc. 2004 ACM Workshop on Interdisciplinary Software Engineering Research (WISER, FSE-12), Newport Beach, CA, USA, 2004, pp. 55–66, doi: 10.1145/1029997.1030009.

[21] J. Torres-Jimenez, H. Avila-George, and E. F. Parra-Gonzalez, "Binary test-suites using covering arrays," Int. J. Software Engineering and Knowledge Engineering, vol. 28, no. 9, pp. 1321–1337, 2018, doi: 10.1142/S0218194018500377.

[22] D. Benavides, A. Felfernig, J. A. Galindo, and F. Reinfrank, "Automated analysis in feature modelling and product configuration," in Software Reuse for Competitive Advantage (ICSR 2013), LNCS 7925. Berlin, Germany: Springer, 2013, pp. 160–175, doi: 10.1007/978-3-642-38977-1_11.

[23] C. Elsner, P. Ulbrich, D. Lohmann, and W. Schröder-Preikschat, "Consistent product line configuration across file type and product line boundaries," in Software Product Lines: Going Beyond (SPLC 2010), LNCS 6287. Berlin, Germany: Springer, 2010, pp. 151–165, doi: 10.1007/978-3-642-15579-6_13.

[24] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Reverse engineering feature models," in Proc. 33rd Int. Conf. on Software Engineering (ICSE), Honolulu, HI, USA, May 2011, pp. 461–470, doi: 10.1145/1985793.1985856.

[25] D. R. Kuhn, R. N. Kacker, and Y. Lei, Introduction to Combinatorial Testing. Boca Raton, FL, USA: CRC Press, 2013, ch. 12 ("Combinatorial coverage as an aspect of test quality" per the authors' related presentations), doi: 10.1201/b14957.

[26] L. Yu, Y. Lei, R. N. Kacker, and D. R. Kuhn, "ACTS: A combinatorial test generation tool," in Proc. IEEE 6th Int. Conf. on Software Testing, Verification and Validation (ICST), Luxembourg, Mar. 2013, pp. 370–375, doi: 10.1109/ICST.2013.57.

[27] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A general strategy for t-way software testing," in Proc. 14th IEEE Int. Conf. and Workshops on the Engineering of Computer-Based Systems (ECBS), Tucson, AZ, USA, Mar. 2007, pp. 549–556, doi: 10.1109/ECBS.2007.47.

[28] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," IEEE Trans. Software Eng., vol. 23, no. 7, pp. 437–444, Jul. 1997, doi: 10.1109/32.595861.