Enabling AI-Augmented MBSE with MCP

Systems Engineering Capstone: Advisor Review

Andrew Dunn
GitLab, Public Sector

Greg Pappas
Department of Defense, Army DEVCOM

Dr. Stephen Rapp (Advisor)
Wayne State University, ISE

2026-02-15

Thesis & Motivation

Central Thesis

Harness design matters more than model capability for AI-augmented systems engineering.

LLMs alone cannot reliably produce system models. Tooling that provides parsing, validation, and structural feedback transforms LLM output from unusable to useful.

The Problem Isn't Going Away

- **Raw LLM capability is insufficient.** SysMBench tested 17 LLMs including frontier models; best BLEU score was **4%**, best semantic F1 was **62%** [1]
  - Enhancement strategies (few-shot, chain-of-thought) provided only marginal improvement. The failure mode is structural (modeling language syntax), not reasoning
- **Context windows are a ceiling, not a floor.** Bader et al. exceeded 16K tokens generating **~8 elements** in XMI; even 30% pre-processing reduction was insufficient [2]
  - Larger windows don't help when the format is inherently verbose
- **Corpus scarcity can't be trained away.** SysML v2 was adopted July 2025; training data barely exists. SysTemp compensates with templates because LLMs can't learn from limited examples [3]
- **Single-shot generation fails.** Ferrari et al. found correctness not significantly above baseline across 28 requirements documents; session memory pollution actively degrades output [4]

The Gap

- **0** MCP servers for SysML among 7,364+ public MCP repositories
- SE project context alone consumes **40K+ tokens**, leaving minimal budget for model content
- No standardized AI-MBSE interface; every paper implements custom integration

Why Now

- MCP protocol reached stability (Nov 2024), adopted by major AI providers
- SysML v2 textual notation explicitly recommended as more LLM-friendly than XMI [2]
- 75,000+ GitHub stars on MCP ecosystem repos; momentum is real
- Tree-sitter grammars are straightforward to produce (~25 hours to 99.6% coverage), but OMG's KEBNF spec doesn't easily yield one (335+ LR conflicts). A hand-tuned grammar is a meaningful contribution
- No tree-sitter grammar for SysML v2 existed; tree-sitter's error recovery and incremental parsing make it uniquely suited for AI tooling where ANTLR doesn't fit

Exploring the Space

- An MCP server with context-aware tools (L0/L1/L2 detail levels, cache IDs, overflow detection) is a **research instrument** for exploring reduction strategies, not just a product
- Each tool call is a measurable experiment: what context does the LLM actually need to reason about a model?
- Positions capstone as infrastructure for continued research (directed study → INCOSE 2027 benchmark)

SysML v2: From Documentation to Computation

What is MBSE?

Model-Based Systems Engineering means expressing system designs as structured, machine-readable models rather than documents. SysML is the OMG standard language for this: system architects use it to define parts, requirements, behaviors, and constraints for complex systems in defense, aerospace, automotive, and medical domains.

Why Do People Want LLMs in MBSE?

MBSE has a persistent **model-reality gap**: system architectures that can't be validated, simulated, or traced to requirements without extensive manual effort and custom tooling. Engineers spend more time maintaining models than reasoning about designs.

LLMs promise to bridge this by reading, generating, and modifying models directly. But they need the right input format, and they need structured tooling to compensate for their limitations (4% BLEU on system model generation even with frontier models [1]).

The v1 → v2 Revolution

SysML v1 was graphical-only, UML-based, with ambiguous semantics. XMI interchange failed because "every tool exports XMI differently." v2 (adopted July 2025 by OMG, 80+ organizations, 7 years of development) is a fundamentally different language built on KerML formal logic:

```
part def Vehicle {
    attribute mass :> ISQBase::mass;
    part engine : Engine {
        attribute mass :> ISQBase::mass = 200 [kg];
    }
    attribute totalMass :> ISQBase::mass =
        engine.mass + transmission.mass;
}
```

Requirements become evaluable constraints that return true/false, enabling automated verification directly from models.

Why This Changes Everything for AI

Property	SysML v2	SysML v1 / XMI
Format	Textual, human-readable	Graphical / XML serialization
Diffable	Yes (Git-native)	No (binary diagrams, verbose XML)
Semantics	KerML formal logic	UML profile, ambiguous
Constraints	Built-in expression language	OCL (software-oriented)
Token cost	Compact (~50 tokens/element)	Verbose (~500+ tokens/element)
Paradigm	Computation	Documentation

The XMI Problem

Bader et al. found XMI serialization consumed **16K+ tokens** for just **~8 model elements** [2]. Even 30% pre-processing reduction was insufficient. Compare the same element in both formats:

```
<packagedElement xmi:type="uml:Component"
  xmi:id="_abc123" name="Vehicle"
  visibility="public">
  <ownedAttribute xmi:type="uml:Property"
    xmi:id="_def456" name="mass" .../>
</packagedElement>
```

vs. `part def Vehicle { attribute mass :> ISQBase::mass; }` in SysML v2.

Why Now

SysML v2's textual notation makes LLM+MBSE tractable for the first time. Models can be stored in Git, parsed by tree-sitter, processed through CI/CD pipelines, and accessed by AI agents via MCP. This is the exact workflow this capstone demonstrates.

Literature Positioning

The Harness Matters More Than the Model

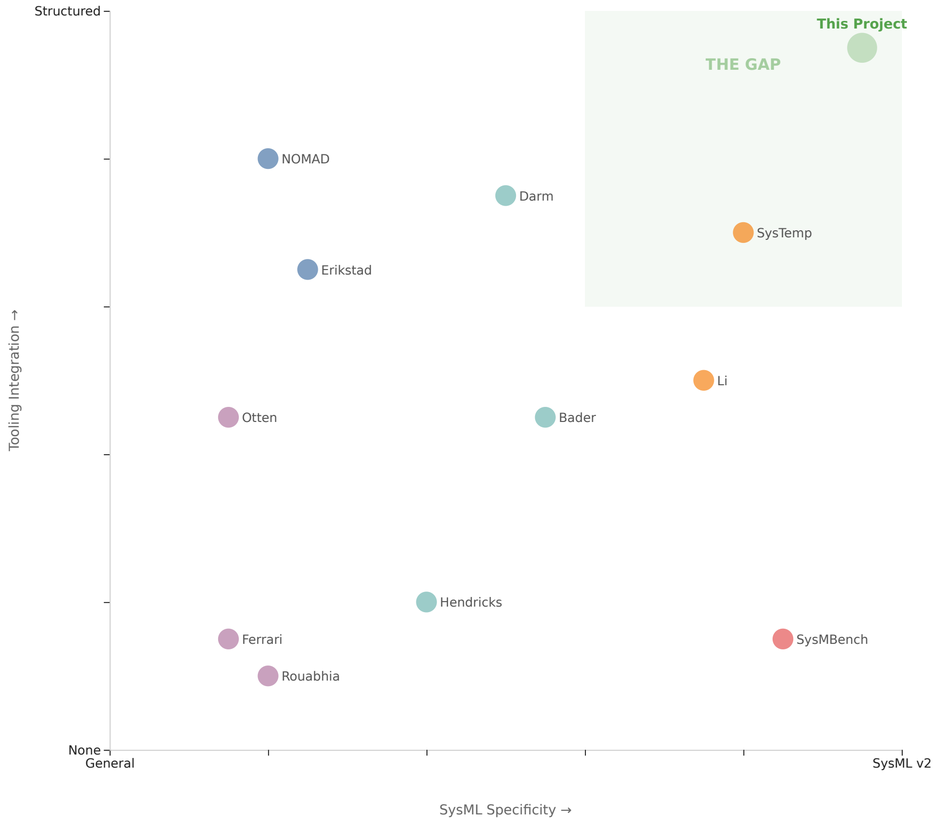
Our central thesis: **harness design matters more than model capability** for AI-augmented systems engineering. The literature provides strong quantitative evidence.

SysMBench [1] tested 17 LLMs including frontier models on system model generation. Best BLEU score: **4%**. Best semantic F1: **62%**. Enhancement strategies (few-shot, chain-of-thought) provided only marginal improvement. The failure mode is structural, not reasoning. When the best available LLMs achieve 4% on system models, external tooling is not optional but essential.

The literature reveals five context management strategies (avoidance, staged decomposition, template-mediated structuring, progressive narrowing, multi-agent partitioning) but each paper implements custom integration. No standardized interface exists.

The Sweet Spot

Exploring the intersection of **SysML v2 specificity** and **structured MCP tooling** is a meaningful research contribution. The scatter plot (right) shows 10 papers across two dimensions. No existing work occupies the top-right quadrant. SysTemp comes closest (template-mediated SysML v2) but uses custom integration, not a reusable protocol. **0** MCP servers for SysML among 7,364+ public MCP repositories.



Infrastructure with Independent Value

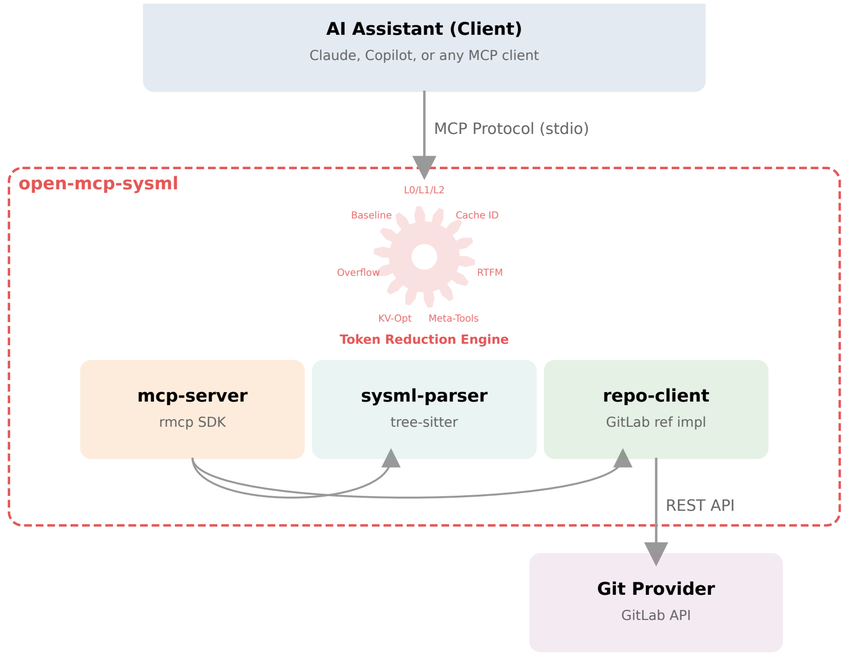
The artifacts built for this exploration have standalone utility regardless of thesis outcome:

- **tree-sitter-sysml**: First SysML v2 grammar for tree-sitter. MIT licensed, 6 language bindings. Usable for syntax highlighting, code navigation, and IDE support independent of any LLM integration
- **kebnf-to-tree-sitter**: First automated KEBNF-to-tree-sitter converter. Applicable to any OMG specification grammar, not just SysML
- **open-mcp-sysml**: Reusable MCP server pattern for any Git-hosted modeling language. Provider-agnostic design (GitLab as reference implementation)

Project Concept & Architecture

What We Built

A **Rust MCP server** that gives AI assistants structured access to SysML v2 models via the Model Context Protocol.



Technology Choices

Decision	Choice	Rationale
Language	Rust 1.85	Memory safety, single binary, rmcp SDK
Parser	tree-sitter	Error recovery, incremental, 6 bindings
Transport	stdio	Sufficient for evaluation; HTTP planned
Git provider	GitLab (ref impl)	Trait-based; provider-agnostic design

Five MCP Tools (Phase 1)

Tool	Purpose	Token Cost
`sysml_parse`	Parse SysML v2 with L0/L1/L2 detail	100–2,000
`sysml_validate`	Return parse diagnostics	~300
`sysml_list_definitions`	List all definitions in model	~200
`repo_list_files`	List .sysml files in repo	~300
`repo_get_file`	Read file from repository	Variable

Token Reduction Strategies

Literature-informed strategies, each sourced from practitioner implementations:

#	Strategy	Reduction	Source
0	Vanilla (baseline)	0%	n/a
1	L0/L1/L2 Tiered ⊠	~90-95%	OpenViking [5]
2	Cache ID + Summary	~97%	xc-mcp [6]
3	RTFM Docs	~80%	xc-mcp [6]
4	Two Meta-Tools	~95%	mcp-proxy [7]
5	KV-Cache Opt	10x cost	Manus [8]
6	Overflow Detection	Advisory	EACL 2026

Strategy 1 implemented in Phase 1. Strategies 2-6 defined in Phase 2 PRD with implementation timeline TBD; some may land within capstone scope depending on benchmark execution schedule.

Products: Near-Term & Far-Term

Near-Term (Capstone Scope, Apr 2026)

Deliverable	Status	Description
tree-sitter-sysml	✅ Complete	SysML v2 grammar: 99.6% coverage, 125 tests, 6 bindings
kebnf-to-tree-sitter	◐ In Progress	Automated KEBNF→tree-sitter converter, 640/640 rules parsed
open-mcp-sysml	✅ Phase 1	Rust MCP server: 5 tools, 22 tests, stdio transport
Capstone Book	◐ In Progress	15-chapter SE documentation (this Quarto book)
GVSETS 2026 Paper	◐ Drafted	AI-Augmented MBSE via MCP: draft Mar 23, final Jun 5

Far-Term (Post-Capstone)

Deliverable	Timeline	Description
sysml.rs	Directed Study	Semantic analysis engine on tree-sitter-sysml
Grammar Transposition	Q3-Q4 2026	KEBNF methodology for INCOSE/SysEng Journal
INCOSE 2027 Benchmark	Q3 2027	SE benchmark for evaluating LLM capability
sysml-grammar-benchmark	Q2-Q3 2026	Comparative grammar dashboard (Quarto + D3)
MCP Server Phase 2	Post-capstone	Token reduction, HTTP transport, SysML v2 API

Key Metrics

- **2,236** lines of grammar.js
- **274/275** external SysML v2 files parsed (99.6%)
- **190/190** tree-sitter queries covered
- **93%** automated KEBNF rule conversion (vs 60-70% estimated)
- **~25 hours** total development time for grammar

Directed Study Intent

We intend to continue this work as a **directed study** focused on:

1. **sysml.rs**: Rust-native semantic analysis (type checking, scope resolution, constraint evaluation for SysML v2)
2. **INCOSE 2027**: SE-specific benchmark evaluating LLM performance with and without MCP tooling
3. **Conflict resolution**: Resolving 335+ LR conflicts in the spec-driven grammar

The directed study builds directly on capstone infrastructure and targets two additional publications.

SE Process & Tailoring

INCOSE Processes Sampled

6 of 30+ processes, tailored per Handbook 5th Ed §4.3.4: “tailoring should be commensurate with project scope and risk”:

Artifact	INCOSE Process	Handbook §
SEP	Project Planning	2.3.4.1
Stakeholder Analysis	Stakeholder Needs & Req	2.3.5.2
SyRS	System Requirements Def	2.3.5.3
ADD	Architecture Definition	2.3.5.4
VVP	Verification & Validation	2.3.5.9, 2.3.5.11
RTM	Traceability	3.2.3

These 6 represent the **minimum viable SE backbone** for a software-intensive project: planning (SEP), requirements chain (Stakeholders → SyRS), design allocation (ADD), verification (VVP), and cross-cutting traceability (RTM). The 24+ processes omitted (Configuration Management, Decision Management, Integration, etc.) are either handled implicitly by Git/CI tooling or are not meaningful for a 3-person academic team over 15 weeks.

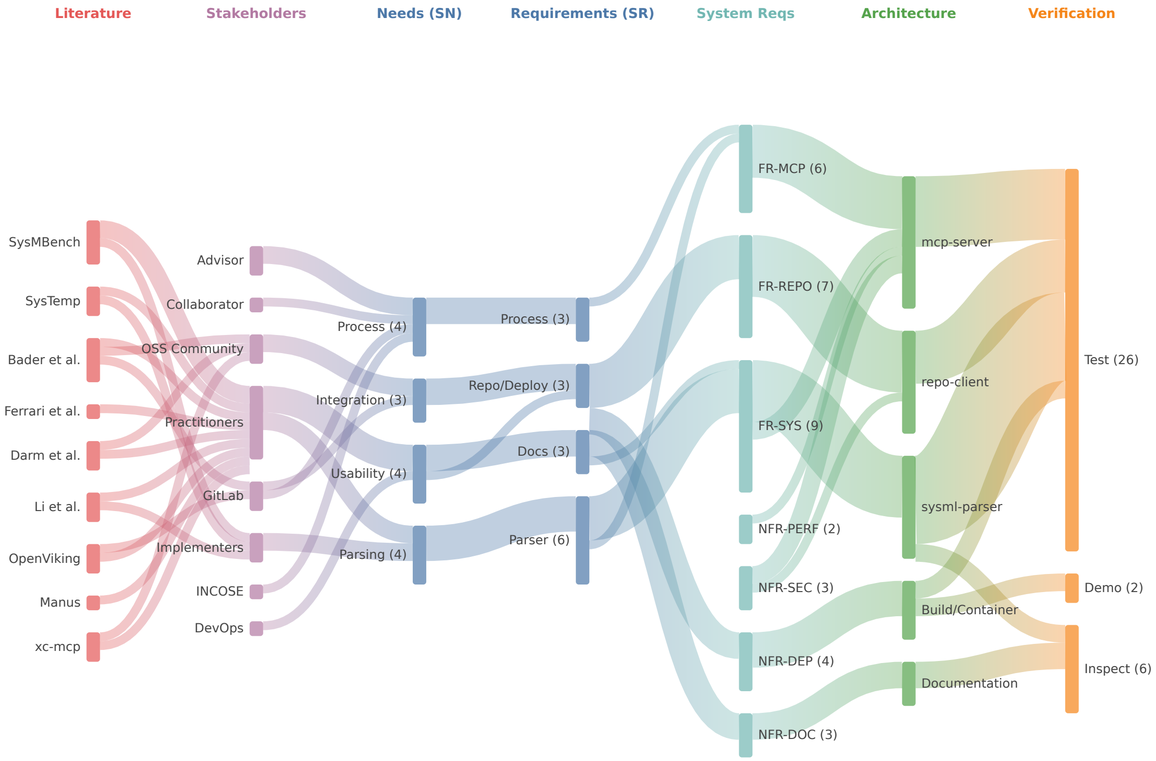
Tailoring Rationale

Constraint	Decision
15-week timeline	Combined SRR + PDR into single assessment
3-person team	Informal stakeholder validation via iterative dev
Academic scope	Interface requirements in ADD only (no IR-xxx IDs)
Implementation-first	Reviews after implementation; more meaningful

SRR and PDR conducted **after** initial implementation: requirements grounded in real constraints, architecture decisions validated by working code, review findings immediately actionable. Trade-off (scope commitment before formal review) mitigated by small team and rapid iteration.

Lifecycle: Hybrid Agile + Formal Gates

Phase	Weeks	Activities
Concept	1-2	Literature review, SEP, stakeholders
Design	3-4	SyRS, ADD, architecture selection
Implement	5-11	Grammar, MCP server, GVSETS paper
V&V + Ship	12-15	VVP execution, capstone delivery



18 Papers → 8 Stakeholders → 15 Needs → 15 Requirements → 34 System Reqs → 7 Components → 34 Verifications

Stakeholders & Requirements Flow

Why Formal Stakeholder Analysis?

We didn't just build a tool. Per INCOSE Handbook §2.3.5.2, we identified 8 stakeholders, elicited 15 needs, derived 15 stakeholder requirements, and decomposed those into 34 system requirements. This discipline ensures the system serves real users, not assumed ones.

Stakeholder Categories

Category	Stakeholders	Strategy	Key Needs
Primary	Advisor, Collaborator	Manage Closely	SE methodology, defensible deliverables
Practitioners	OSS Community, MBSE Practitioners	Keep Satisfied	Git integration, single binary, CI/CD examples
Ecosystem	GitLab, SysML v2 Implementers	Keep Informed	GVSETS demo, spec conformance
Community	INCOSE, Sensmetry	Monitor / Inform	Novel contribution, interoperability

Requirements Traceability Chain

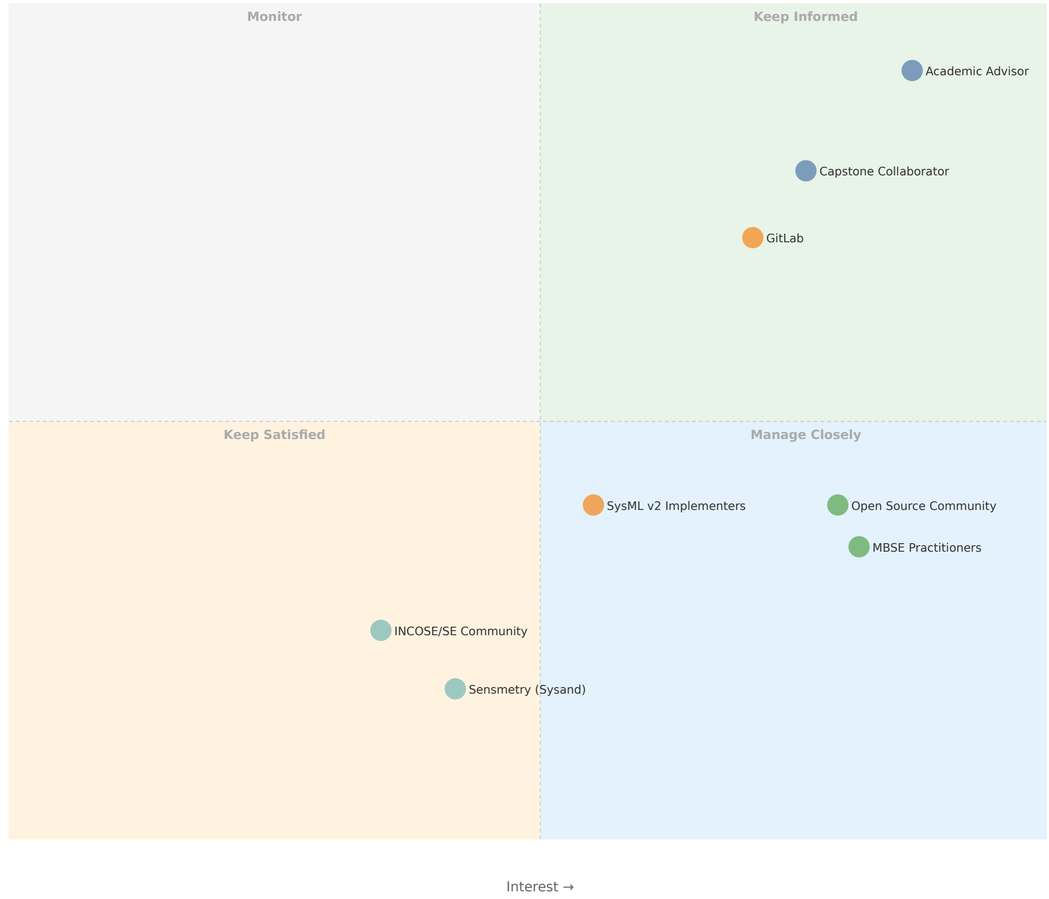
8 stakeholders → 15 needs → 15 requirements → 34 system requirements

The 15 stakeholder needs (SN-001 through SN-015) trace to 15 stakeholder requirements (SR-001 through SR-015) with some many-to-one mappings. These decompose into 34 system requirements spanning functional (FR), non-functional (NFR), and interface (IR) specifications.

Representative needs driving the design:

- **SN-001** (OSS Community): Git provider API integration for existing repositories
- **SN-007** (Practitioners): Easy installation (single binary, no dependencies)
- **SN-012** (Practitioners): Model validation against SysML v2 spec
- **SN-014** (Implementers): OMG spec conformance in parsing

Power / Interest Analysis



SE Artifacts & Review Status

Artifacts Produced

Artifact	Ch	Key Content
SEP	6	Lifecycle, risk register, gates
Stakeholders	8	8 stakeholders, 15 needs/reqs
SyRS	9	34 requirements (FR/NFR/IR)
ADD	10	3-crate arch, tool specs
VVP	11	11/34 verified, VMA matrix
RTM	16	SN→SR→SyR→Test trace

~4,500 lines across 15 chapters

Review Gates

Gate	Date	Status
SRR	Feb 14	☒ Complete
PDR	Feb 14	☒ Complete
CDR	Mar 29	☒ Pending

Both passed with caveats reflecting our implementation-first approach:

- **SRR**: Interface requirements deferred (documented in ADD); stakeholder validation conducted iteratively; requirements baselined with understanding that benchmark execution may drive updates
- **PDR**: VVP requirement IDs reconciled (FR-Umcp → FR-MCP naming); ADD tool definitions updated to match implementation
- All action items resolved except VVP test case updates (in progress for CDR)

Risk Register

Risk	Status
R1: SysML v2 API availability	Mitigated
R2: GVSETS deadline pressure	**Open**
R3: Grammar complexity	Closed (99.6%)
R4: MCP SDK maturity	Closed
R5: Container deployment	Planned (CI/Linux)
R6: Benchmark validity	Mitigated
R7: Spec-driven conflicts	Closed
R8: Team availability	Closed
R9: tree-sitter org acceptance	**Open**

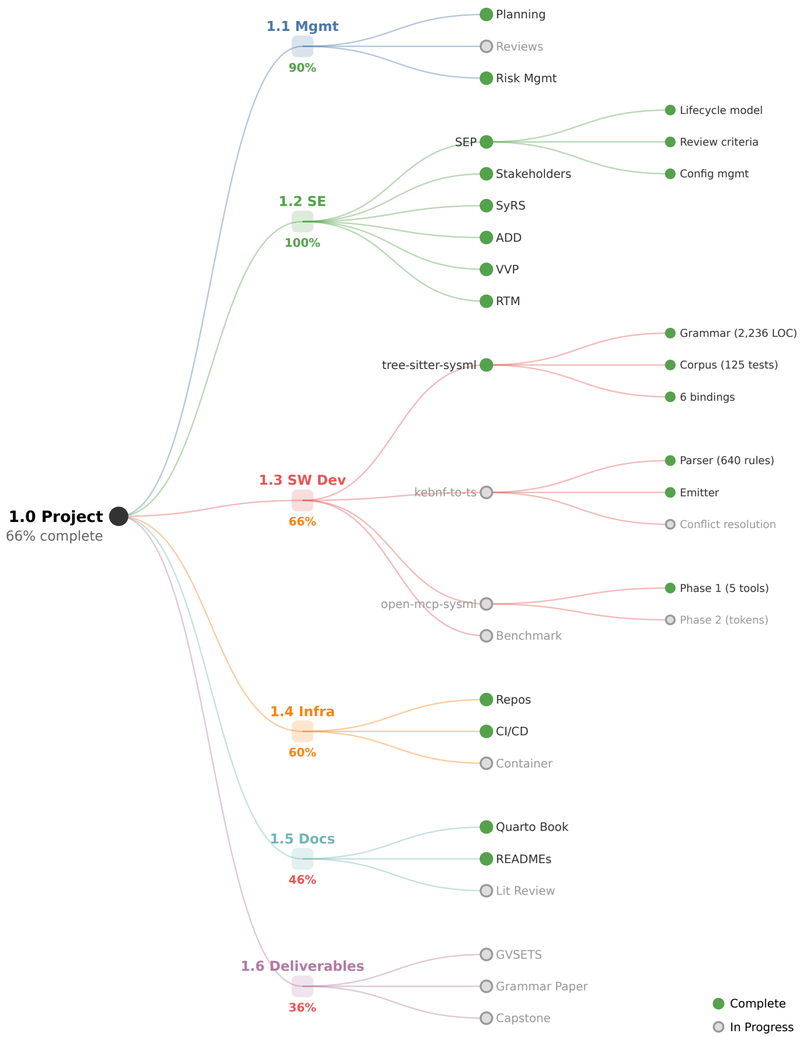
4 closed, **2 mitigated**, **1 planned**, **2 open**. No risk ≥ escalation threshold.

VMA Coverage

11/34 requirements verified, covering the highest-risk requirements verifiable through current test infrastructure. Planned methods across all 34: **26 by Test** (automated CI), **2 by Demonstration** (manual), **6 by Inspection** (review).

The 23 deferred requirements span HTTP transport, SysML v2 API integration, repository write operations, and security, all Phase 2+ features. CDR (Mar 29) is the next verification checkpoint.

WBS Structure



Implementation Results

tree-sitter-sysml (Grammar)

Metric	Value
Training file coverage	**100%** (100/100 OMG files)
External file coverage	**99.6%** (274/275 files)
Corpus tests	**125/125** passing
Query coverage	**190/190**
Grammar size	**~2,236 lines**
Development time	**~25 hours**
Language bindings	C, Rust, Go, Python, Node.js, Swift

kebnf-to-tree-sitter (Converter)

Metric	Value
KEBNF rules parsed	**640/640** (100%)
Direct conversion	**38%** of rules
Strip-and-convert	**55%** of rules
Total automation	**93%** (vs 60-70% estimated)
LR conflicts remaining	**335+** (vs 54 in brute-force)
Development time	**~12 hours** (vs 60-100h estimated)

open-mcp-sysml (MCP Server)

Metric	Value
Crates	3 (`mcp-server`, `sysml-parser`, `repo-client`)
MCP tools	**5** implemented
Tests	**22** passing
Development time	**~20 hours**

Dual-Path Grammar Strategy

Two independent paths to SysML v2 grammar, validated by cross-comparison:

	Brute-Force	Spec-Driven
Repository	tree-sitter-sysml	kebnf-to-tree-sitter
Method	Empirical: study corpus	Formal: parse OMG KEBNF
Authoring	Manual grammar.js	Automated rule generation
Conflicts	54 (resolved)	335+ (in progress)
Status	☒ Production ready	◐ Research contribution
Value	Immediate practical use	Reproducibility + INCOSE paper

Cross-validation found 12 grammar rules where brute-force diverged from spec intent; each path catches errors the other misses.

Research Infrastructure

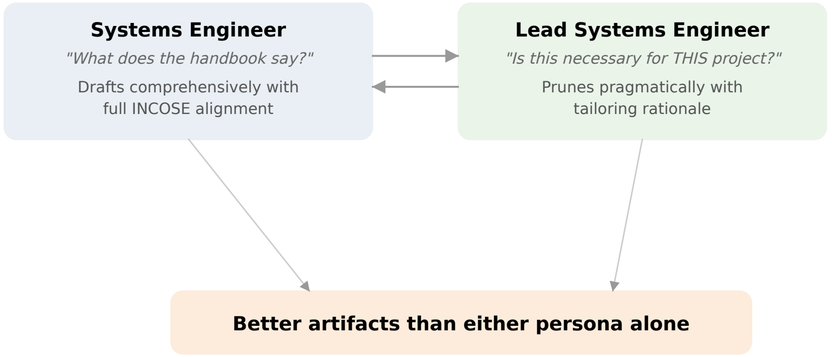
The dual-path grammar + MCP server form **infrastructure for post-masters study and continued contributions**:

- **sysml.rs** (directed study): Semantic analysis engine built on tree-sitter-sysml. Import resolution, type checking, constraint evaluation beyond syntax-only parsing
- **Grammar transposition** (INCOSE/SysEng Journal): The spec-driven converter targets a methodology paper on automated KEBNF-to-tree-sitter conversion
- **INCOSE 2027 benchmark**: Grammar + MCP server enable evaluating LLM performance on SE tasks with and without structured tooling

AI-Augmented Meta-Process

The SE / LSE Dialectic

We designed a **multi-persona AI workflow** where two personas with deliberate tension produce better artifacts than either alone:



Seven Skill Personas

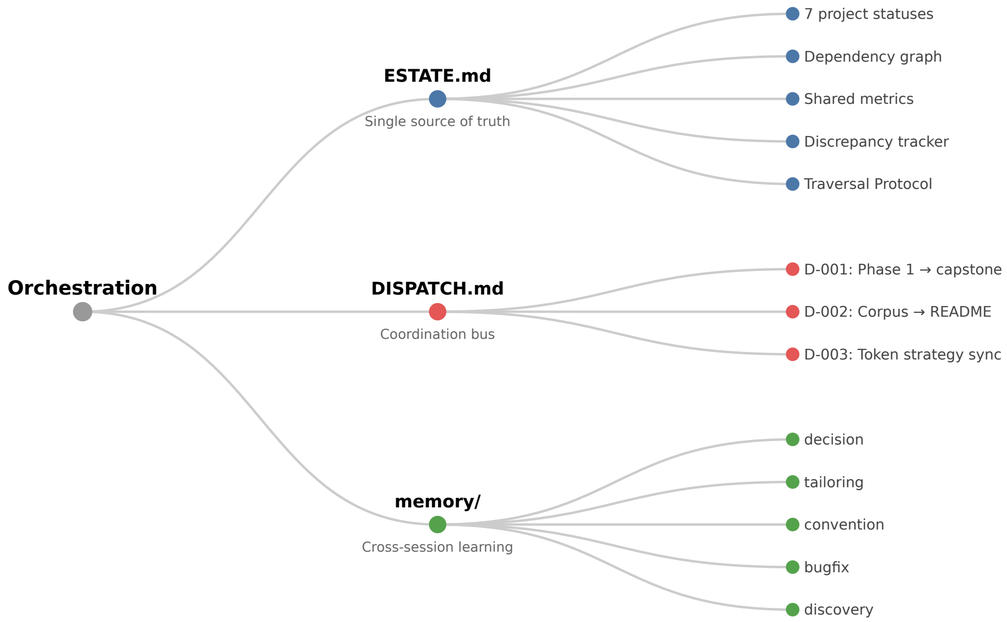
Skill	Role	Runs As
`systems-engineer`	INCOSE-aligned drafting	Subagent
`lead-systems-engineer`	Pragmatic review + tailoring	Subagent
`upstream`	MCP/SysML/INCOSE spec lookup	Subagent
`mcp-builder`	Implementation guidance	Subagent
`publisher`	Academic writing (GVSETS, INCOSE)	Main ctx
`diagrammer`	Precision ASCII/Unicode diagrams	Main ctx
`quarto`	Build validation before push	Main ctx

Three-Tier Model Allocation

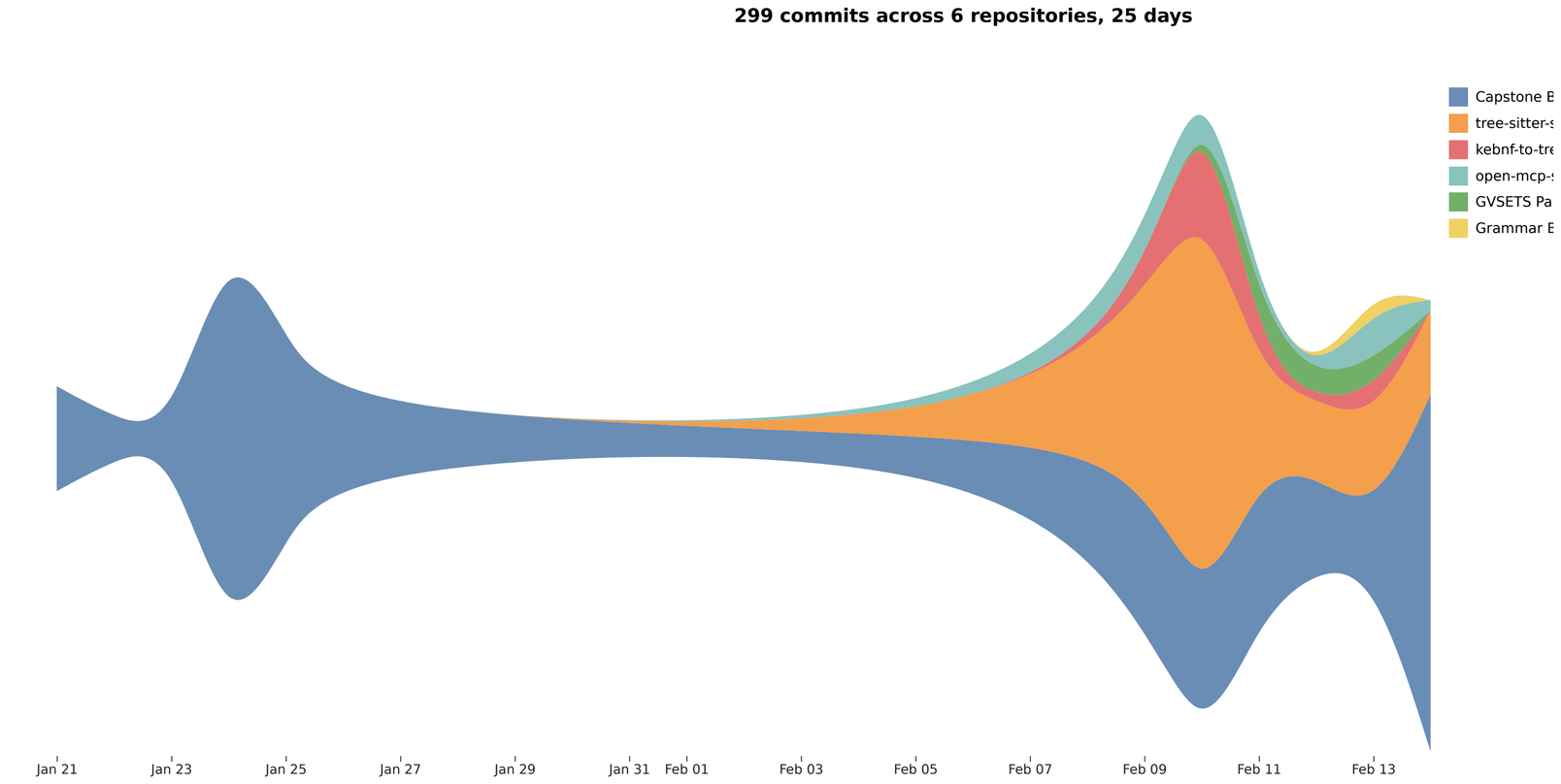
Tier	Model	Purpose	Context
Orchestrator	Opus	Cross-project decisions, consistency	Full estate
Worker	Sonnet	Single-project implementation	Project scope
Scout	Haiku/Flash	File search, status checks	Minimal

Token-cost-aware: expensive models for cross-cutting decisions, cheaper models for bounded tasks. Inspired by Yegge's Gas Town orchestration pattern and Beads git-backed memory [9]; adapted to a minimal file-based approach (ESTATE.md + DISPATCH.md + memory/) that provides the coordination value without the infrastructure complexity.

Estate-Level Orchestration



Iteration & Evolution



The streamgraph reveals a clear phase transition: capstone-only work (Jan) gave way to simultaneous multi-repo development (Feb 9+). The widening bands show how grammar, server, and paper work fed each other; cross-project commits on the same day were the norm, not the exception.

By the Numbers

Metric	Value
Total commits	**299**
Repositories	**6**
Active days	**13** of 25 calendar days
Peak day	**Feb 10** (69 commits)
Capstone lines	**~4,500** across 15 chapters
Capstone commits	**168**
Literature entries	**18** papers reviewed

Evolution Narrative

Phase 1 (Jan 21-26): Capstone foundation: SEP, literature review, stakeholder analysis, initial architecture. 78 commits in 6 days.

Quiet period (Jan 27 - Feb 8): Research and specification study. Reading SysML v2 spec, MCP protocol, tree-sitter docs.

Phase 2 (Feb 9-14): Implementation sprint: grammar development, MCP server, GVSETS paper, converter tool. All 5 repos active simultaneously. 221 commits in 6 days.

Consistency pass (Feb 14): Full traversal protocol plus presentation polish: title renames, tool reconciliation, D3 diagram scaling, narrative strengthening. 34 capstone commits in one day.

Lessons Learned

Technical Surprises

Grammar development was dramatically faster than expected. ~25 hours for 99.6% external coverage vs. an estimate of weeks. Tree-sitter's declarative grammar.js and test-driven corpus approach turned what we expected to be the critical path into a solved problem within the first sprint.

Automated KEBNF conversion exceeded estimates. 93% automation rate vs 60-70% initial estimate. The converter tool finished in ~12 hours vs 60-100h estimated. The remaining 7% (335+ LR conflicts) is the genuinely hard problem, but the tool eliminated the mechanical work entirely.

Cross-validation caught real divergences. 12 grammar rules where the brute-force grammar diverged from spec intent. Neither path alone would have found these. The dual-path strategy wasn't just academic; it validated itself by catching errors in both directions.

Process Insights

Implementation-informed reviews produced better artifacts. Conducting SRR and PDR after initial implementation meant requirements were grounded in real constraints, not theoretical. Review findings were immediately actionable because working code existed. The trade-off (scope commitment before formal review) was mitigated by small team and rapid iteration.

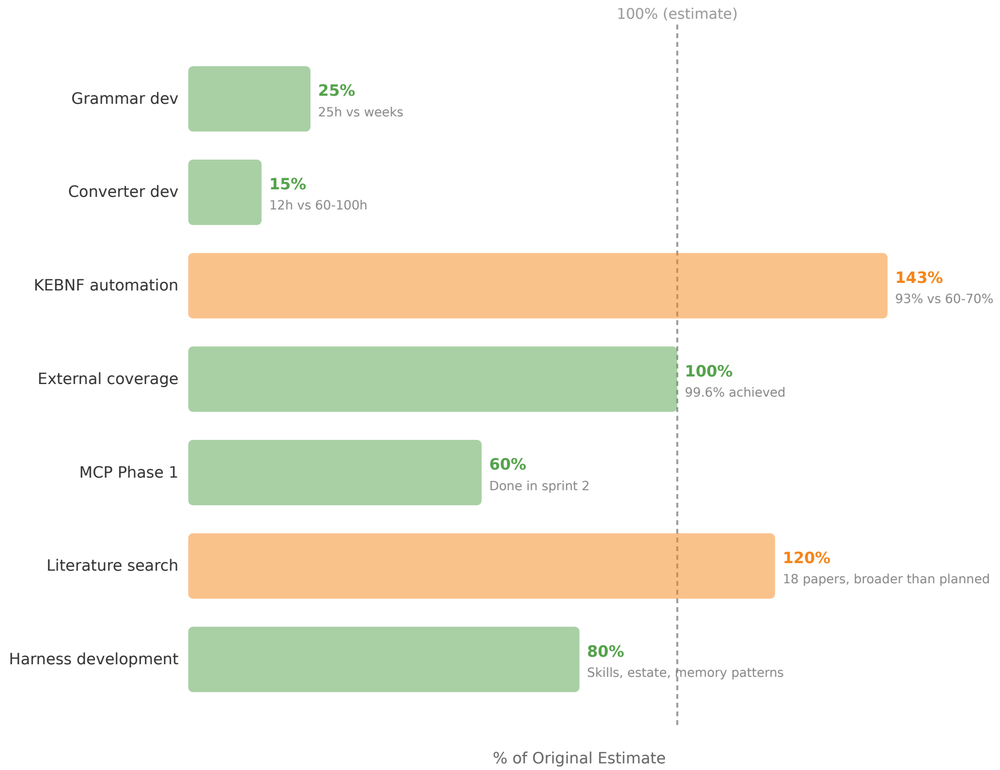
The SE/LSE dialectic was genuinely useful. The multi-persona AI workflow (Systems Engineer drafts comprehensively, Lead Systems Engineer prunes pragmatically) produced better INCOSE-aligned artifacts than either approach alone. The deliberate tension forced explicit tailoring rationale for every deviation.

Context engineering is an emerging discipline. The literature search surfaced practitioner work (Manus, OpenViking, mcp-proxy) that was as valuable as academic papers for informing token reduction strategies. The best architectural patterns came from open source implementations, not journals.

What We'd Do Differently

- Start CI container builds from day 1 (R5 risk would have been retired earlier)
- Begin stakeholder validation outreach earlier (Sensmetry, GfSE)
- Establish the estate orchestration pattern before the implementation sprint, not during it

Actual vs. Estimated Effort



Green bars = completed under estimate. Orange = exceeded estimate (a good thing for automation rate). The dashed line marks the original estimate baseline.

Demo

Content to be determined pending presentation timetable.

Next Steps & Conclusion

Remaining Milestones

Milestone	Date	Action
GVSETS draft	**Mar 23**	Execute V1/V4/V5 benchmarks, fill metrics
CDR	**Mar 29**	Critical Design Review with team
Capstone delivery	**Apr 25**	Final book + all artifacts
GVSETS notification	May 1	Accept/reject
GVSETS final	Jun 5	Camera-ready paper
GVSETS presentation	Aug 11	Conference (Novi, MI)

Deferred Scope (Post-Capstone)

Item	Rationale
HTTP transport	stdio sufficient for evaluation
SysML v2 API integration	Local parsing sufficient (R1)
Container deployment	CI/Linux builds planned
Token reduction Phase 2	6 strategies defined, implementation timeline TBD

Three Publications

Paper	Timeline	Focus
GVSETS 2026	Mar–Jun 2026	MCP architecture + 3-condition benchmark
Grammar Transposition	Q3–Q4 2026	KEBNF-to-tree-sitter methodology (INCOSE/SysEng Journal)
INCOSE 2027	Q3 2027	SE benchmark for LLMs (directed study)

The Thesis, Validated

The capstone demonstrates that **structured tooling** (grammar-aware parsing, tiered token budgets, validation feedback) transforms LLM interaction with system models from unreliable to practical.

- **7 token reduction strategies** defined (1 implemented, 6 in Phase 2 PRD), each sourced from practitioner literature and open source implementations
- **Dual-path grammar** provides both immediate utility (tree-sitter-sysml) and formal research contribution (kebnf-to-tree-sitter)
- **MCP protocol** provides the standard interface; **SE process** ensures defensible foundations
- Infrastructure positions three publications and a **directed study** (sysml.rs + INCOSE 2027 benchmark)

[1] D. Jin, Z. Jin, L. Li, Z. Fang, J. Li, and X. Chen, “A System Model Generation Benchmark from Natural Language Requirements.” 2025. Available: https://arxiv.org/abs/2508.03215

[2] E. Bader, D. Vereno, and C. Neureiter, “Facilitating User-Centric Model-Based Systems Engineering Using Generative AI,” in *Proceedings of the 12th international conference on model-based software and systems engineering (MODELSWARD 2024)*, SCITEPRESS, 2024.

[3] Y. Bouamra, B. Yun, A. Poisson, and F. Armetta, “SysTemp: A Multi-Agent System for Template-Based Generation of SysML v2.” 2025. Available: https://arxiv.org/abs/2506.21608

[4] A. Ferrari, S. Abualhaija, and C. Arora, “Model Generation with LLMs: From Requirements to UML Sequence Diagrams,” *arXiv preprint*, 2024, Available: https://arxiv.org/abs/2404.06371

[5] Volcengine, *OpenViking: Context database for AI agents*. (2025). Available: https://github.com/volcengine/OpenViking

[6] C. Luddy, *Xc-mcp: XCode CLI MCP server with progressive disclosure*. (2025). Available: https://github.com/conorluddy/xc-mcp

[7] S. Rodda, *Mcp-proxy: Aggregating MCP proxy with progressive tool disclosure*. (2025). Available: https://github.com/IAMSamuelRodda/mcp-proxy

[8] Y. Ji, “Context engineering for AI agents: Lessons from building manus.” Accessed: Feb. 12, 2026. [Online]. Available: https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

[9] S. Yegge, “Welcome to gas town.” Accessed: Feb. 12, 2026. [Online]. Available: https://steve-yegge.medium.com/welcome-to-gas-town-4f25ee16dd04