Incremental Roadmap: v0.8.5 → v0.9.0

Current Version: v0.8.5
Target Version: v0.9.0 ("Everything-but-Haskell Working")
Timeline: 8 weeks
Status: Planning
Updated: February 1, 2026

Overview

This roadmap breaks down the path from v0.8.5 to v0.9.0 into 5 incremental releases, each building on the previous version. The goal is to achieve production-ready status with all 3 pipelines (Python, Rust, SPARK) having complete feature parity.

Release Schedule

```
v0.8.5 (Current)
    ↓
v0.8.6 - Test Infrastructure (2 weeks)
    ↓
v0.8.7 - Exception Handling (2 weeks)
    ↓
v0.8.8 - Advanced Data Structures (1 week)
    ↓
v0.8.9 - Generics & Optimization (2 weeks)
    ↓
v0.9.0-rc1 - Release Candidate (1 week)
    ↓
v0.9.0 - Production Release (End of Week 8)
```

---

v0.8.6 - Test Infrastructure (Week 1-2)

🎯 Goal

Establish comprehensive automated testing for Rust and SPARK pipelines

📋 Deliverables

1. Rust Test Suite (GAP-001)

- [ ] **spec_to_ir tests** (25 tests minimum)
- Basic spec parsing tests
- Multi-file spec tests
- Control flow spec tests
- Error handling tests

- Edge case tests
- [ ] **ir_to_code tests** (25 tests minimum)
- Basic code generation tests
- Type inference tests
- Control flow code gen tests
- Optimization tests (if applicable)
- Target-specific tests
- [ ] **Integration tests** (10 tests minimum)
- Full pipeline tests (spec → IR → code)
- Cross-compilation tests
- Determinism validation tests
- Performance baseline tests

2. SPARK Test Suite (GAP-002)

- [ ] **Automated SPARK tests** (50 tests minimum)
- GNATprove integration tests
- Proof obligation validation
- Bounded type tests
- Memory safety tests
- DO-178C compliance tests
- [ ] **Regression test suite**
- v0.8.5 feature regression tests
- Control flow regression tests
- Code generation regression tests

3. Integration Testing Framework (GAP-004)

- [ ] **Cross-pipeline validation**
- IR hash comparison tests
- Determinism validation
- Output equivalence tests
- Performance comparison
- [ ] **CI/CD Integration**
- Automated test runs on commit
- Multi-platform test execution
- Test result reporting
- Coverage tracking

✅ Success Criteria

- Rust test coverage >30%
- SPARK test coverage >30%
- All tests passing in CI/CD
- Integration tests validate cross-pipeline determinism

- Test infrastructure documented

📊 Metrics

- **Target Test Count**: 100+ tests
- **Target Coverage**: >30% for Rust/SPARK
- **Effort**: 20 person-days (2 developers × 2 weeks)

---

v0.8.7 - Exception Handling (Week 3-4)

🎯 Goal

Implement complete exception handling (try/catch/finally) across all pipelines

📋 Deliverables

1. IR Schema Extension (GAP-003)

- [ ] **Define exception operations**

```
json
  {
    "op": "try",
    "try_block": [...],
    "catch_blocks": [
      {
        "exception_type": "IOException",
        "variable": "e",
        "handler_block": [...]
      }
    ],
    "finally_block": [...]
  }
```

- [ ] **Update IR validator**

- Validate exception operation structure
- Validate exception type references
- Validate variable scoping in catch blocks

- [ ] **Schema documentation**

- Update `stunir_ir_v1` schema
- Add exception handling examples
- Document exception semantics

2. Python Implementation

- [ ] **spec_to_ir** exception parsing
- Parse try/catch/finally from spec
- Generate exception IR operations
- Validate exception structure

- [ ] **ir_to_code** exception generation

- Generate try/catch/finally C code
- Generate Rust Result patterns
- Generate Python try/except code
- Handle exception propagation
- [ ] **Tests** (20+ tests)
- Basic try/catch tests
- Multiple catch blocks
- Finally block tests
- Nested exception tests
- Exception propagation tests

3. Rust Implementation

- [ ] **spec_to_ir** exception parsing
- Follow Python implementation
- Rust-specific Result mapping
- [ ] **ir_to_code** exception generation
- Generate idiomatic Rust error handling
- Support both exceptions and Result types
- Proper error propagation with `?` operator
- [ ] **Tests** (20+ tests)
- Match Python test coverage
- Rust-specific patterns

4. SPARK Implementation

- [ ] **spec_to_ir** exception parsing
- Bounded exception structures
- SPARK-safe exception handling
- [ ] **ir_to_code** exception generation
- Generate C code with error codes
- SPARK-verifiable exception patterns
- Memory-safe exception handling
- [ ] **Tests** (20+ tests)
- Match Python/Rust test coverage
- Formal verification tests

✅ Success Criteria

- Exception handling working in all 3 pipelines
- 60+ exception-related tests passing
- Generated code compiles without errors
- Documentation complete
- Examples demonstrating exception patterns

📊 Metrics

- **Target Test Count**: 60+ exception tests
- **Target Coverage**: Exception handling >80%
- **Effort**: 30 person-days (2 developers × 3 weeks, compressed to 2)

---

v0.8.8 - Advanced Data Structures (Week 5)

🎯 Goal

Add support for advanced data structures (arrays, maps, sets)

📋 Deliverables

1. IR Schema Extension (GAP-005)

- [ ] **Array operations**
- Array initialization: `{"op": "array_init", "type": "int[]", "size": 10}`
- Array access: `{"op": "array_get", "array": "arr", "index": 5}`
- Array set: `{"op": "array_set", "array": "arr", "index": 5, "value": 42}`
- [ ] **Map operations**
- Map initialization: `{"op": "map_init", "key_type": "string", "value_type": "int"}`
- Map get: `{"op": "map_get", "map": "m", "key": "foo"}`
- Map set: `{"op": "map_set", "map": "m", "key": "foo", "value": 42}`
- Map operations: contains_key, remove, clear
- [ ] **Set operations**
- Set initialization: `{"op": "set_init", "element_type": "int"}`
- Set add: `{"op": "set_add", "set": "s", "value": 42}`
- Set operations: contains, remove, clear

2. Python Implementation

- [ ] Array support (dynamic vectors)
- [ ] Map support (dictionaries/HashMap)
- [ ] Set support (HashSet)
- [ ] Collection iteration
- [ ] Nested structure support
- [ ] Tests (15+ tests)

3. Rust Implementation

- [ ] Array support (Vec)
- [ ] Map support (HashMap)
- [ ] Set support (HashSet)
- [ ] Generic collection types
- [ ] Ownership-aware code generation
- [ ] Tests (15+ tests)

4. SPARK Implementation

- [ ] Bounded array support (fixed-size arrays)
- [ ] Bounded map support (limited capacity)
- [ ] Bounded set support
- [ ] SPARK-verifiable collection operations
- [ ] Memory safety proofs
- [ ] Tests (15+ tests)

✅ Success Criteria

- Advanced data structures working in all 3 pipelines
- 45+ data structure tests passing
- Generated code efficient and correct
- Nested structures supported
- Documentation with examples

📊 Metrics

- **Target Test Count**: 45+ data structure tests
- **Target Coverage**: Data structures >70%
- **Effort**: 15 person-days (3 developers × 1 week)

---

v0.8.9 - Generics & Optimization (Week 6-7)

🎯 Goal

Add generic/template support and optimization framework

📋 Deliverables

1. Generic/Template Support (GAP-006)

- [ ] **IR Schema Extension**
- Generic function definitions
- Type parameters
- Type constraints
- Template instantiation
- [ ] **Python Implementation**
- Generic function parsing
- Template code generation
- Type substitution
- [ ] **Rust Implementation**
- Rust-style generics
- Trait bounds
- Monomorphization
- [ ] **SPARK Implementation**

- Ada generic packages
- SPARK-verifiable generics
- Bounded type parameters
- [ ] **Tests** (20+ tests per pipeline)

2. Optimization Pass Framework (GAP-007)

- [ ] **IR Optimization Passes**
- Constant folding
- Dead code elimination
- Common subexpression elimination
- Copy propagation
- [ ] **Code Generation Optimizations**
- Loop unrolling
- Function inlining
- Register hints
- Tail call optimization
- [ ] **Optimization Tests** (25+ tests)
- Validate optimizations correct
- Verify performance improvements
- Ensure determinism preserved

3. Debug Information Generation (GAP-008)

- [ ] **DWARF Debug Info**
- Line number tables
- Variable location info
- Type information
- [ ] **Debug Tests** (10+ tests)
- Verify debug info correctness
- Test with gdb/lldb
- Validate source mapping

✅ Success Criteria

- Generic functions working in all 3 pipelines
- Optimization framework operational
- Debug info generation working
- 80+ new tests passing
- Performance improvements measurable

📊 Metrics

- **Target Test Count**: 80+ tests
- **Performance**: 10-20% speed improvement from optimizations
- **Effort**: 30 person-days (3 developers × 2 weeks)

v0.9.0-rc1 - Release Candidate (Week 8)

🎯 Goal

Final polish, bug fixes, and comprehensive validation

📋 Deliverables

1. Bug Fixes

- [ ] Fix BUG-001: Python variable redeclaration
- [ ] Fix BUG-002: SPARK stack overflow (increase limits)
- [ ] Fix BUG-003: Improve error messages in SPARK
- [ ] Fix BUG-004: Type inference for complex expressions

2. Documentation (GAP-009)

- [ ] **Complete User Guide**
- Getting started tutorial
- 10+ practical examples
- Common patterns guide
- Troubleshooting guide
- FAQ section
- [ ] **API Reference Updates**
- Document new exception handling APIs
- Document data structure APIs
- Document generic function APIs
- Document optimization flags
- [ ] **Migration Guide**
- v0.8.5 → v0.9.0 migration steps
- Breaking changes (if any)
- Deprecation notices

3. Comprehensive Testing

- [ ] **Full Test Suite Run**
- All unit tests passing
- All integration tests passing
- Cross-pipeline validation
- Performance benchmarking
- [ ] **Quality Gates**
- Test coverage >60% (all pipelines)
- No P0 or P1 bugs
- All success criteria met
- Performance targets met

4. Release Preparation

- [ ] Version number updates
- [ ] Changelog updates
- [ ] Release notes draft
- [ ] Build release binaries
- [ ] Security audit
- [ ] License verification

✅ Success Criteria

- All tests passing (300+ tests)
- Test coverage >60%
- All P0/P1 gaps closed
- No critical bugs
- Documentation complete
- Release artifacts ready

📊 Metrics

- **Total Test Count**: 300+ tests
- **Test Coverage**: >60% (Python >70%, Rust >50%, SPARK >50%)
- **Effort**: 7 person-days (release focus)

---

v0.9.0 - Production Release (End of Week 8)

🎯 Goal

Official production release of STUNIR v0.9.0

📋 Deliverables

1. Release Artifacts

- [ ] Source code release
- [ ] Precompiled binaries (Linux x86_64, Linux arm64, macOS)
- [ ] Docker images
- [ ] Release notes
- [ ] Changelog
- [ ] Security advisory (if needed)

2. Documentation Publishing

- [ ] Updated website
- [ ] API documentation
- [ ] User guides
- [ ] Examples repository
- [ ] Video tutorials (optional)

3. Announcement

- [ ] Release announcement
- [ ] Blog post

- [ ] Social media
- [ ] Email to stakeholders

✅ Success Criteria

- All v0.9.0 success criteria met (see gap analysis report)
- Release artifacts available
- Documentation published
- Announcement sent

Summary: What Changes in Each Version

v0.8.6: Test Infrastructure

New: 100+ automated tests, CI/CD integration
Improved: Reliability, quality assurance

v0.8.7: Exception Handling

New: try/catch/finally support
Improved: Error handling capabilities

v0.8.8: Advanced Data Structures

New: Arrays, maps, sets support
Improved: Data modeling capabilities

v0.8.9: Generics & Optimization

New: Generic functions, optimization passes, debug info
Improved: Code quality, performance, debuggability

v0.9.0: Production Release

New: Complete “everything-but-Haskell” feature set
Improved: Production-ready quality

Risk Management

High-Risk Items

1. **SPARK formal verification** - May take longer than estimated
   - **Mitigation**: Start SPARK work early, leverage existing patterns

2. **Exception handling complexity** - Cross-language semantics challenging
   - **Mitigation**: Prototype in Python first, adapt to Rust/SPARK

3. **Resource constraints** - May need to defer some features
   - **Mitigation**: Strict prioritization, focus on P0/P1 gaps only

Medium-Risk Items

1. **Test infrastructure setup** - Initial overhead
   - **Mitigation**: Reuse existing test patterns

2. **Optimization correctness** - Must preserve semantics
   - **Mitigation**: Extensive testing, formal verification in SPARK

Resource Allocation

Team Structure (Recommended)

- **2-3 developers** working full-time
- **1 technical writer** (part-time) for documentation
- **1 QA engineer** (part-time) for testing

Weekly Breakdown

- **Week 1-2**: Test infrastructure (all developers)
- **Week 3-4**: Exception handling (2 dev Python/Rust, 1 dev SPARK)
- **Week 5**: Data structures (all developers)
- **Week 6-7**: Generics + optimization (2 dev implementation, 1 dev testing)
- **Week 8**: Release prep (all hands on deck)

Monitoring and Reporting

Weekly Checkpoints

- **Monday**: Sprint planning, task assignment
- **Wednesday**: Mid-week status check
- **Friday**: Sprint review, demos, retrospective

Metrics Tracking

- Test count (target: 300+ by v0.9.0)
- Test coverage (target: >60%)
- Bug count (target: 0 P0/P1 bugs)
- Performance benchmarks
- Documentation completion

Decision Points

- **After v0.8.6**: Evaluate test coverage progress, adjust timeline if needed
- **After v0.8.7**: Evaluate exception handling completeness, decide on data structure scope
- **After v0.8.8**: Evaluate optimization scope, decide on RC readiness
- **After v0.9.0-rc1**: Final go/no-go decision for v0.9.0 release

---

Contingency Plans

If Behind Schedule

1. **Defer P2 gaps** to v0.9.x releases
2. **Reduce test count** targets (minimum >50%)

3. **Simplify optimization** passes (focus on constant folding only)
4. **Extend timeline** by 1-2 weeks if absolutely necessary

If Ahead of Schedule

1. **Add more tests** (target >80% coverage)
2. **Implement P2 gaps** (async/await, function pointers)
3. **Add more documentation** (video tutorials, advanced guides)
4. **Early release** of v0.9.0

Post-v0.9.0 Planning

v0.9.x Patch Releases

- Bug fixes
- Performance improvements
- Minor feature additions
- Documentation updates

v1.0.0 Planning

- Haskell pipeline integration
- IDE plugins
- Package manager integration
- Advanced optimization passes
- Production hardening

Roadmap Version: 1.0
Last Updated: February 1, 2026
Next Review: After v0.8.6 completion
Contact: STUNIR Core Team