Project Post-Mortem: Borrow-Checking LISP Language

Date: December 18, 2024 ## Team Reflections ### Zachary Whitaker - Project Manager & System Architect As the project manager and lead system architect, my primary responsibility was ensuring that our vision of a memory-safe LISP implementation remained cohesive throughout development. The integration of various system components proved to be one of our biggest challenges. - Key Contributions:

```
- Developed the overall system architecture that allowed seamless interaction between the pa
- Managed project timeline and resource allocation across different development phases
- Coordinated between team members to ensure consistent implementation of design patterns
- Established development milestones and technical requirements
```

- Challenges Faced:
  - Initial timeline estimates proved optimistic given the complexity of integrating borrow checking
  - Coordinating parallel development of the parser and static analyzer required multiple architecture revisions
  - Balancing features with maintaining project scope was an ongoing challenge

Gregory Shiner - Language Guru & System Architect

My focus as language guru and system architect was on designing the core language features while ensuring they could be effectively implemented within our architectural constraints. - Key Achievements:

```
- Designed the syntax for expressing ownership and borrowing within LISP's s-expression fram
- Developed the specification for the type system and borrow checker integration
- Created the architectural framework for static analysis implementation
- Wrote most of the parser and interpreter
```

- Areas for Improvement:
  - Some language features required complex architectural solutions that impacted performance
  - Initial language specification needed several iterations to fully support all desired features
  - Documentation of language internals could have been more comprehensive
  - Interpreter should have been developed earlier on

Kevin Skinner - Language Guru & Validation Lead

As both a language guru and validation lead, I focused on ensuring our implementation met its design goals while maintaining correctness and performance.

- Key Contributions:

```
- Developed comprehensive test suite for language features and memory safety guarantees
- Designed and implemented validation frameworks for the parser
- Contributed to core language design decisions and implementation
```

  - Technical Insights:
    - Automating test creating with Rust macros significant improved test development time
    - Verification framework helped identify several critical issues early in development

Project Outcomes

Successful Aspects

- Built turing complete LISP language
- Fully implemented arithmetic operators, comparison operators, and control flow structures
- Fully implemented functional programming aspects such as lambda expressions, closures, function application, and static scoping environments
- Fully implemented type checking for arithmentic, comparison, and control flow structures for early warnings of type errors
- Partially implemented borrow checking and memory management features
- Built a rigorous testing and verification framework that enables rapid iteration and expansion

Areas for Improvement

- Integration testing could have started earlier in development
- Documentation processes needed more structure
- Better tooling support for development workflow

Unexpected Discoveries

- The architecture proved more adaptable to new features than anticipated
- Integration challenges led to innovative solutions in parsing and sugaring
- Validation process revealed optimization opportunities
- System performance exceeded expectations

Future Directions

- Enhance development tooling
- Improve error reporting system
- Expand test coverage
- Explore parallel computing capabilities
- Develop IDE integration

- Create advanced debugging tools

Conclusion

The project successfully demonstrated the viability of implementing Rust-style memory safety in a LISP environment. The team's diverse expertise in project management, language design, and validation enabled us to overcome significant technical challenges while maintaining our core design principles. Though we encountered various obstacles, particularly in integration and timeline management, the resulting system achieved its primary goals of memory safety, performance, and functional programming flexibility. Moving forward, the lessons learned from this project provide valuable insights for future language development efforts, particularly in the areas of safety-critical systems and performance-oriented functional programming.