1. Compiler architecture evolution

Monolithic compiler vs Modern compiler

Reason for IR

Semantic IR introduction

Wider term of Compilation (all based on semantic analysis):

- Code generation

- Human understandable view (visualization)

- Static analysis

- Program interpretation

- XML as SR format

1. Using XML as an IR for Compiler Development

- XML as example of open IR format

1. C++ program database

- Relational DBMS as program model storage for easy and effective interoperation and data sharing between analysis modules

- Mapping C++ to ERD

- Incremental parsing optimization: parse only changed part

1. Software Knowledge System  
   - Alternative approach to the one chosen by Meyer would be an integration of SKB into an editor: the exact thing that LSP is good for. It was not possible back than without rewriting half of OS but it’s possible now.

His ideas can be implemented as an LS analysis module for a known language model.

1. Semantic APIs for PLs

- Lack of instruments for C++, industry don’t understand that programs are not just text objects: program is a set of types, functions, expressions.

- Problems of classic compilers’ IR for semantic analysis. It’s fragmented and useful only for compiler. (VC++), there is no guarantee that compiler’s IR will not change in the next release

- IR as a logical component, product of a compiler frontend.

- Problems of IR being compiler tables: rather low-level with very limited use cases.

- SR maps to constructions of a language

- SR interface in SemantiC++ core functionality: generation, validation, semantic search

- Distributed (recursive) validation

- Semantic search

1. Traditional compilers’ IR problems
2. Modern compilers and Semantic Representation
3. LSP and distributed approach to building devtools