How to Create a Programming Language

L8: MORE ON LANGUAGE DESIGN

1

OVERVIEW

- NEW LANGUAGE DEVELOPMENT – WHT IS INVOLVED
- PHASES OF DESIGN
  - FUNDAMENTAL QUESTIONS TO ASK
  - REFINEMENT OF DESIGN
- SOME MISTAKES IN LANGUAGE DESIGN
- SOME OVERARCHING PRINCIPLES

2

NEW LANGUAGE DEVELOPMENT

- DESIGNING THE LANGUAGE – SPECIFYING FUNDAMENTAL PARADIGMS THAT COMPRISE THE NEW LANGUAGE
- WRITE A COMPILER
- DEVELOP A STANDARD 'LIBRARY'
- DESIGN/DEVELOP SUPPORTING TOOLS [EDITOR, BUILD UTILS, PROFILING, ETC]

3

PHASES OF DESIGN

- TYPICALLY, THERE ARE TWO PHASES
  - FUNDAMENTAL QUESTIONS ABOUT LANGUAGE CHARACTERISTICS
    - HIGH LEVEL/BROAD STROKES
    - SHAPE THE STRENGTHS AND GOALS OF THE LANGUAGE
  - REFINEMENT OF DESIGN
    - TWEAKING FEATURES AND STRUCTURES TO ALLOW LANGUAGE TO BE 'IMPLEMENTABLE'

4

HIGH LEVEL DESIGN

- QUESTIONS INCLUDE
  - WHAT IS THE 'EXECUTION PARADIGM'?
    - IMPERITIVE
    - FUNCTIONAL
    - STATE MACHINE
    - BUSINESS RULES?
    - GOAL DIRECTED?
  - WHAT VARIABLE TYPING IS USED
    - STATIC
    - DYNAMIC

5

HIGH LEVEL DESIGN

- WHAT IS THE TARGET USAGE OF THE LANGUAGE?
  - SMALL SCRIPTS?
  - LARGE SYSTEMS?
- WHAT 'QUALITIES' ARE MOST IMPORTANT
  - EXECUTION SPEED/EFFICIENCY?
  - READABILITY?
  - EASY DEVELOPMENT (WRITABILITY)

6

HIGH LEVEL DESIGN

- WILL IT BE SIMILAR OR PATTERNED AFTER A SPECIFIC EXISTING LANGUAGE?
  - C
  - PYTHON
  - LISP/ML

7

HIGH LEVEL DESIGN

- ARE WE TARGETING A SPECIFIC PLATFORM?
  - JVM
  - .NET
- WILL IT SUPPORT ANY METAPROGRAMMING CAPABILITIES?
  - MACROS
  - TEMPLATES
  - REFLECTION
  - ETC

8

MISTAKES IN LANGUAGE DESIGN

- NULL POINTERS
- PARSER UNFRIENDLY SYNTAX
- UNCLEAR SEMANTICS
- BAD UNICODE SUPPORT
- PREPROCESSOR

9

NULL POINTERS

- PROBLEMS WITH NULL POINTERS
  - C/C++ - CRASH WHEN DEREFERNCING NULL POINTERS
  - JAVA, PYTHON, ETC – THROW EXCEPTIONS
- ALTERNATIVES
  - CREATE STATIC TYPE SYSTEM THAT FORCES REFERENCES TO BE NON-NULL
  - MAKE IT IMPOSSIBLE TO CREATE A NULL POINTER
    - MEANS CANNOT USE POINTER ARITHMETIC

10

PARSER UNFRIENDLY SYNTAX

- LANGUAGES SHOULD BE LALR OR LL(1)
- EASES DEVELOPMENT OF TOOLS
  - PROBABLY NEED TO WRITE SEVERAL PARSERS
    - COMPILER
    - LANGUAGE SENSITIVE EDITOR
    - ETC

11

UNCLEAR SEMANTICS

- AVOID:
  - UNDEFINED BEHAVIOR
  - IMPLEMENTATION SPECIFIC BEHAVIOR
- EXAMPLES:
  - STANDARDML – HAS A FULLY FORMALIZED SET OF SEMANTICS! [GOOD!]
  - C – HAS MANY IMPLEMENTATION SPECIFIC AND UNDEFINED BEHAVIORS
    - LEADS TO DIFFICULTIES WITH PORTABILITY OF PROGRAMS

12

BAD UNICODE SUPPORT

- MOSTLY CONCERNS STRING HANDLING IN THE LANGUAGE
- BUT: PROGRAMMERS SOMETIMES USE OTHER LANGUAGES TO NAME VARIABLES SO SOURCE SHOULD SUPPORT UNICODE!

13

PREPROCESSOR

- PREPROCESSORS ARE TYPICALLY USED
  - INCLUDE OTHER FILES
  - DEFINE MACROS
  - ALLOW CONDITIONAL COMPILATION
- SIMPLE MACRO EXPANSION DOES NOT KNOW THE SEMANTICS OF THE LANGUAGE CREATING ODD DIFFICULT TO TROUBLESHOOT BUGS
- INCORPORATE SUCH FEATURES INTO THE LANGUAGE DEFINITION

14

SOME OVERARCHING PRINCIPLES

- NON-TRUTHS
  - TRIED AND TRUE
  - OVER DESIGN
  - READABILITY
- TRUTHS
  - SUITABILITY – DESIGN FOR A NICHE
  - SIMPLICITY – DESIGN A SIMPLE SYNTAX WITH LITTLE/NO AMBIGUITY
  - ITERATIVE DEVELOPMENT – START WITH A MINIMALLY VIABLE LANGUAGE

15

TRIED AND TRUE [AVOID]

- DON'T TRY TO MIMIC WELL KNOWN SYNTAX TO MAKE LANGUAGE EASIER INITIALLY
  - NICHE LANGUAGES SHOULD HAVE GRAMMARS THAT CATER TO THE NICHE
  - MINIMIZES EFFORT TO SOLVE PROBLEMS
  - EXAMPLES
    - MAKE A DECLARATIVE LOGIC LANGUAGE LOOKG SIMILAR TO PROLOG
    - MAKE A 'STACK BASED' LANGUAGE LOOK LIKE FORTH

16

OVER-DESIGN [AVOID]

- AVOID BUILDING AN OVERLY COMPLEX FULL FEATURED LANGUAGE
- START WITH MINIMAL, VIABLE (USEFUL) LANGUAGE
- ADD FEATURES AS THEY BECOME 'APPARENT'

17

READABILITY [AVOID]

- DON'T WORRY ABOUT INITIAL LEARNING CURVE (READING COMPUTER LANGUAGES IS A LEARNED SKILL)
  - => RATHER, MAKE USER IT IS EASY TO PARSE
- IF LANGUAGE IS DESIGNED FOR A SPECIFIC NICHE/DOMAIN, DEVELOPERS WILL PICK IT UP

18

SUITABILITY [GOOD!]

- DESIGN LANGUAGE THAT STRONGLY TARGETS THE DESIRED PURPOSE
- DON'T TRY TO PACK LANGUAGE WITH MANY OTHER FEATURES TO MAKE IT MORE 'GENERAL PURPOSE'

19

SIMPLICITY [GOOD!]

- CREATE SIMPLE FOUNDATIONAL SYNTAX WITH MINIMAL AMBIGUITY
- MAKES PARSING EASIER
- MAKE SURE FEATURES ADDED LATER ARE COMPATIBLE WITH THE FOUNDATIONAL SYNTAX

20

ITERATIVE DEVELOPMENT [GOOD!]

- AGAIN, CREATE A MINIMALLY VIABLE LANGUAGE
- ADD FEATURES AS THE NEED IS DISCOVERED
- EASIER THAN TRYING TO FIGURE OUT EXAUSTIVELY ALL NEEDED FEATURES

21

QUESTIONS?

22