Chapter 1

Readability

- overall simplicity
  - constructs
- feature multiplicity
  - accomplish something in multiple ways
- operator overloading
  - single symbol ~~for~~ multiple definitions

orthogonality:
how small components can combine to make smallthings & produce the outcome
↑ ortho, ↑ simplicity

datatypes:
adequate ability to define datatypes & structures

Syntax design:
> special words (for, if)
> form & meaning (appearance indicates purpose)
>

Writability

how easily language can be used to create programs for chosen domain
simplicity & ortho

abstraction:
define & use structures, allowing details to be ignored
> process: subprogram implements algorithm used various times (algor)

expressivity
- convenient expressions
- efficient

Reliability

> performs to specifications for certain conditions

type checking

exception handling

aliasing

1970s – procedure oriented to data-oriented methods
1980's – object oriented design

–

Cost

training
writing programs
compiling programs
optimization: collection of techniques compilers use to decrease size/increase speed of code
language implementation system:
↑ expense, ↓ use
poor reliability
program maintenance!!

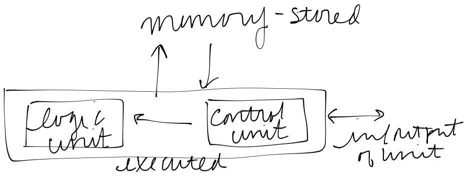
portability
ease of moving a program from implem. to another

generality:
applicable to diff. apps

well-definedness
completeness & precision of lang official doc

Language design influence

imperative languages:
designed around von Neumann architecture
> data from cpu to memory transmitted back & forth



"fetch/execute style"

Language categories

imperative
functional
logic – rule based
object-oriented
visual (ex. NET)
scripting (ex. perl, ruby)

how languages are complimented
* compiler – fast
* interpreter – error catching
* hybrids
* preprocessors (processed before compiled)

Chapter 2