12
Sharp Tools

Sharp Tools

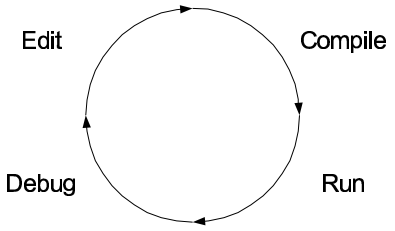
- Target Machines
- Vehicle Machines and Data Services
- High-Level Language and Interactive Programming

Tool support

- The toolmaker role supports common tools
  - Aim for global optimization through sharing
- Individual programers create specialized tools
  - Aim for local optimization through expedience
- Essential tools include
  - Computer facility, with scheduling policy
  - Programming language, with usage policy
  - Utilities; Debuggers; Test case generators
  - Text-processing system

Programming cycle

- Tools accelerate one or more stages in the programming cycle



Target machines

- The target machine executes the program
- Vehicle machines contribute to preparation
- Most contemporary application development uses the same machine for vehicle as for target
- Common counter-examples include
  - Embedded controllers, mobile devices and embedded (software) interpreters
  - For example, anti-lock brakes, cell phones and web plugins (e.g., Macromedia Flash)

Target facilities

- Target machine operators and/or hardware specialists, especially for new designs
- Excessive machine capabilities, to achieve function before optimizing for efficiency
- Instrumented hardware and/or software, to allow for controlled execution (e.g., single-step) and state analysis (e.g., stack and memory dump)
- For example, see commercial in-circuit emulators at http://www.isystem.com/

Scheduling

- Target machine time is scarce for new hardware
- Brooks tried two scheduling strategies
  - 4 runs per day @ 2½ hour turnaround each
  - 5-hour blocks, each dedicated to a single team
- Dedicated blocks of time were more productive
  - 10 runs in a 6-hour block are better than
  - 10 runs spaced 3 hours apart
- How does this relate to the programming cycle?

Simulators

- Simulators substitute for target machines
- Dependability is more valuable than accuracy
- Lab-built, preproduction and early hardware
  - Often does not work reliably
  - Often changes from day to day
- A dependable simulator is useful, even after the production hardware becomes available
- How does this relate to the programming cycle?

Compiler and assembler vehicles

- Cross-compilers
  - Execute on one machine architecture
  - Output object code for a different architecture
- For example, the GNU C compiler (gcc) compiles
  - From C, C++, Objective C, Fortran, Java, Ada
  - To about 30 architectures (including S/390)
  - On about 10 major architectures
- See http://gcc.gnu.org/

Program libraries and accounting

- Brooks description covers both
  - Source code version control
  - Integrated release control
- Individuals use separate working areas
- Project uses a common central repository
- Changes progress from individuals to repository
- Integrated results progress through approval from *development* through *test* to *production* release

Documentation system

- Electronic documentation allows for
  - Version synchronization with source code
  - Indexes, cross-references and searching
- Design and user documentation can be stored using the same repository tools as the source code
- Analysis tools can extract commentary text from source code and generate indexed documents
- For example, Knuth's literate programming unifies the document with the program

High-level languages

- Modern compilers overcome original objections
  - Execution efficiency
  - Code size
  - Expressiveness
- Assembly programming still finds applications on
  - Resource-constrained platforms such as embedded controllers or mobile processors
  - Operating systems; Compilers

Interactive programming

- Interactivity and concurrent multiuser access were novel innovations in the 1960's
- The modern programmer still chooses between compiled and interpreted languages
- E.g., C++ and Java versus Python and PHP
- How does this decision relate to the programming cycle?