# *6*
# *Passing the Word*

## *6 Passing the Word*

- Written Specifications – The Manual
- Formal Definitions
- Direct Incorporation
- Conferences and Courts
- Multiple Implementations
- The Telephone Log
- Product Test

## *The manual*

- The architect prepares the external specification of the product as the user would see the product
- Preparation takes several cycles with feedback from users and implementers
- Describe every detail visible to user (what)
- Not describe what is invisible (how)
- Style: Accurate and complete
- Consistency dominates

## *Formal definitions*

- Present information *very precisely*
- Natural (human) language is not suitable
- Formal notations are required for precision

## *Merits and weaknesses*

- Merits
  - Very precise, accurate and complete
  - Incompleteness shows up very conspicuously
- Weaknesses
  - Lacks comprehensibility
  - Needs prose explanations
- Formalisms capture *what*; Prose explains *why*

## *Standard specifications*

- Specifications can have both a formal definition, and a prose definition
  - Either (but only) one can be the standard
  - The other must be derived
  - With each clearly labeled as such
- E.g., Algol 68 has formal definition as standard and prose definition as descriptive

## *Algol68 BNF fragment*

```
<PROGRAM> ::= PROGRAM [USE_LIST] UNIT FINISH
<USE_LIST> ::= USE USER_ID {USE_LIST}
<AMODE> ::= BUILT_IN_AMODE | ROW | REFERENCE
   | PROC_MODE |  UNION | STRUCTURE
   | USER_MODE_ID
<AMODE_LIST> ::= AMODE {,AMODE}
<BUILT_IN_AMODE> ::= VOID | BOOL | CHAR
   | INT | REAL
<ROW> ::= [ARRAY_RANGE]AMODE
<ARRAY_RANGE> ::= UNIT[:UNIT]
<REFERENCE> ::= REF AMODE
<PROC_MODE> ::= PROC [(AMODE_LIST)] AMODE
```

From http://cs1.cs.nyu.edu/~robbins/Compiler/

## *Implementations as formal definitions*

- Most formal definitions are implementations
- Syntax does not require an implementation
- Semantics need an implementation.
  - Since implementations include both internal (how) and external details (what), and
  - Specifications apply only to externals, then
  - Must distinctly identify each

## *Trade-offs*

- Unambiguous
  - i.e., always correct, by definition
  - (by someone's definition, see Sun v. Microsoft)
- Over-prescribes the definition
  - Invalid syntax would produce some results, which could become a part of definition
  - Unexpected results might be produced sometimes for boundary cases which would become a part of definition (e.g., register junk)

## *Trade-offs (cont.)*

- Confusion
  - Between the standard implementation
  - And its prose description
- The implementation cannot be modified while being used as a standard

## *Sun versus Microsoft on Java*

- *De facto* standard implementations can dominate
- January 2001 Microsoft agreed to pay Sun $20 million, to accept termination of the prior license agreement, and to a permanent injunction against use of the JAVA COMPATIBLE trademark. Sun has granted Microsoft a limited license to distribute its current version, provided that all future versions pass Sun's compatibility tests. This part lasts seven years. Beyond that, Microsoft can not distribute Java technology.

## *Direct incorporation*

- Useful for defining inter-module interfaces
  - Declare passed parameters or shared storage
  - Can be included at compile time in a macro
  - Declaration can be altered, with only recompilation being necessary
- For example
  - C header files (*.h)
  - Java interfaces
  - XML schema

## *XML Schema example*

```
<?xml version="1.0" encoding="UTF-8"?>
<book isbn="0836217462">
 <title>
  Being a Dog Is a Full-Time Job
 </title>
 <author>Charles M. Schulz</author>
 <character>
  <name>Snoopy</name>
  <friend-of>Peppermint Patty</friend-of>
  <since>1950-10-04</since>
  <qualification>
    extroverted beagle
  </qualification>
 </character>
 <character>
  <name>Peppermint Patty</name>
  <since>1966-08-22</since>
  <qualification>bold, brash and tomboyish</qualification>
 </character>
</book>
```

From http://www.xml.com/pub/a/2000/11/29/schemas/part1.html

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
        <xs:element name="character" minOccurs="0" maxOccurs="unbound
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="friend-of" type="xs:string" minOccurs
                 maxOccurs="unbounded"/>
              <xs:element name="since" type="xs:date"/>
              <xs:element name="qualification" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## *Conferences and courts*

- Apart from direct consultations, formal, larger meetings are useful
- Biweekly or weekly, depending on the size
- Architects and implementers (or their representatives) attend

## *Conference agenda*

- Problems or changes can be proposed by anyone with a prior written distribution of the proposal
- Detailed changes are carefully considered by the implementers and users prior to the meeting
- Creativity and brainstorming are welcome
- Architects enter final solutions into the manual
- If concensus fails, the chief architect decides, in extreme cases, along with the project manager

## *Advantages*

- Same group, i.e., everyone is up to date
- Everyone is deeply involved in the outcome and commited to the goals
- Attendees can search for solutions inside and outside of obvious boundaries
- Written proposals hasten decisions and avoid inconsistencies
- The chief architect's final auhority avoids compromise and delay

## *Courts*

- Many minor issues can accumulate over time
- Project-wide “supreme court” meetings resolve the accumulated issues before a major freeze
- Issues are listed on placquards around the room
- Decisions on all these minors problems are made and the manuals are updated accordingly
- Managers of marketing, engineering etc. also attend this meeting

## *Multiple implementations*

- Over time, the product and the manual drift apart
- The product defines the *de facto* standard, since it is often more difficult to change than the manual
- Multiple implementations can force consistency
  - Only variant implementations need to change
  - Brooks proposes at least two implementations initially, to force compatibility among the products
- Recall example of Sun versus Microsoft Java

## *The telephone log*

- Numerous questions arise during implementation
- Implementers should consult the architect directly
- The architect should maintain records of these *ad hoc* questions and answers
- The architect's logs are concatenated and distributed to users and implementers
- Modern counterparts
  - IRC Chat logs, newsgroups, forums, FAQ's

## *Product test*

- Product tests capture specification information
- An independent test group acts as a surrogate customer, comparing the product to specification
- A defect-tracking system is crucial in communicating among testers, implementers and architects
  - To ensure that all discrepancies are resolved
  - To capture the rationale for the resolution
  - To gauge progress towards release