Software Renovation

and the

Millennium Meltdown

Arie van Deursen
CWI

Renovation

- Software Renovation:
  The fight against *rotting bits*.

- Old age, new user demands, deceased suppliers, flawed design, ancient programming methods, ...

- Legacy system:
  Software system which resists change.

- "Process of repairing and improving large software systems to get them into a good condition."

(Why not throw it all away?)

- Enormous investments;
- Modifications cheaper than restarting from scratch;
- Software *works*: just difficult to modify;
- Where do we live during the reconstruction?
- People like the software they know;
- (Reuse is the best form of reuse).

Example Renovations

- COBOL-74 to COBOL-85;
- COBOL to OO-COBOL-97 (98);
- IMS to relational DBMS;
- Extension to Euro single currency;
- C(++) to Java.
- Year 2000 Corrections;

A Double Digit Problem with a Triple Comma Solution

- Gartner Group estimates::
  - $1.10 per executable LOC;
    10 staff years per MLOC
  - Data reference per LOC ratio of 1:50.
- MITRE estimate:
  - 1.0 – 5.0% of the code affected
  - $0.75 – $8.52 per LOC
    8 to 71 staff years per MLOC

Portfolio Inventory

- Consider a Fortune 500 company:
  $O(10^7)$ lines of code,
  $O(10^4)$ programs, $O(10^4)$ libraries.

- for 10% of executables, the source code cannot be identified.

- 10% of the maintained sources are not used.

- 15% of copybooks not used.

- 10% compiled 10 years, 0.5% 20 years ago.

- Library/compiler versions and options?

The Renovation Prelude

Realize run-time environment that can be reconstructed. Avoid:

- analysis of old modules, yielding erroneous choices, plans, ...
- multiple or no analysis;
- conflict between "normal maintenance" and automatic renovation changes.

Consistent development, testing, and production environment.

2-Digit Year 2000 Exposures

- Arithmetic: addition/subtraction;
- Relational: comparison/sorting
- Fixed century constants (1900)
- 00 or 99 as exception code
- illegal merge of 2-digit and 4-digit years.
- 2000 is a leap year

Code Extracts

- ```
  01 DATE
      02 DAY      PIC   99.
      02 MONTH    PIC   99.
      02 YEAR     PIC   99.
  ```

- ```
  IF (YR-1 > 65 AND YR-2 > 65)
       OR (YR-1 < 66 AND YR-2 < 66)
     IF YR-1 < YR-2
       MOVE YR-1 TO OLDEST-YEAR
     ELSE
       MOVE YR-2 TO OLDEST-YEAR
   ELSE
     IF YR-1 < YR-2
       MOVE YR-2 TO OLDEST-YEAR
     ELSE
       MOVE YR-1 TO OLDEST-YEAR
  ```

Leap Years

- ```
  fun leap(y) =  divisible(y,4) &
    ( !divisible(y,100) | divisible(y,400) )
  ```

- ```
  DIVIDE YEAR BY 4 GIVING Q REMAINDER R.
  IF R=0
    DISPLAY "Leap year!".
  END-IF
  ```

- ```
  IF R=0 and YEAR NOT = 2000
    MOVE 29 TO FEB-DAYS
  END-IF
  ```

- ```
  IF YEAR = "92" OR "96"
    MOVE 366 to BN79QRPP
  ELSE
    MOVE 365 to BN79QRPP
  END-IF
  ```

Impact Analysis

- Detect "potentially affected" fragments.
- Incorrect results:
  - False negative: analysis missed an infection area;
  - False positive: analysis marked safe fragment as infected.
- Analysis spectrum:
  Rough and fast (no false negatives) $\rightarrow$
  clever and slow (reduce false positives).

Impact Analysis (II)

- Search seeds:
  - Variable names, constants
  - Time system calls, JCL scripts
  - Data dictionaries, data files.
- Seed propagation:
  - Moves, calls, I/O, ...
- Attach confidence level to seeds / propagation rules.

Year 2000 Corrections

- Widen the data to 4 digits.

- 100 year range: (sliding) window

  20th century: $B < Y < 99$
  21st century: $0 < Y < B$

- Hexadecimal byte encodings:

  `PIC 9(2)`: 99 = 1999, 9A = 2000, ..., 9F = 2005, A0 = 2006, ...

  `PIC X(2)`: '99' = 1999, '9a' = 2000, ..., '9z' = 2025, ...

- Switch from YYMMDD to CYYDDD.

Refine/2000

- Reasoning Systems, Palo Alto, Stanford spin-off (1985), staff of ± 35;
- Build a *system model*:
  modules, databases, jobs, interfaces, and their data flow.
- Annotate data elements with *types*:
  - years, quarters, months, days;
  - Absolute vs. interval;
  - Range
  - Related elements (records, aliases);
- Use slicing to reduce *false positive*.

Slicing COBOL

```
01 ...
  05 YEAR             PIC 99.
  05 ANV              PIC 99.
01 ...
  05 PARTS-PER-ORDER PIC 99.
  05 PARTS-COUNT     PIC 99.
01 ...
01 TEMP               PIC 99.
P1.
  MOVE YEAR TO TEMP.
  ...
  MOVE TEMP TO ANV.
P2.
  MOVE PARTS-PER-ORDER TO TEMP.
  ...
  MOVE TEMP TO PART-COUNT.
```

Following `MOVES`, is `PART-COUNT` a date variable?

Automatic Code Modification

- Use the widen-the-date approach
- Correct PICtures (redefinitions)
- Correct statements
  (comparison '95', addition '1900');
- Warn about screen maps and JCL scripts;
- Expand stored databases
  - Batch vs on-the-fly conversion.

AutoEnhancer/2000

- Peritus Software, Massachusetts, started 1991, staff of $\pm$200;

- Types in analysis:
  - Ordered sequence of {`C`, `Y`, `M`, `D`, `Z` };
  - Byte position offset (aliases);
  - Manual initial seeds; automatic propagation (L/G equiv. relation).

AutoEnhancer/2000 (Cont.)

- Correction rules for *isolation*, data division, and procedure division;
- Each rule:
  - Informal explanation, examples;
  - Formalization (weakest preconditions);
  - Correctness proof.
- Automatic generation of test case / data base transactions.

COMUDAS

- IBM Netherlands:
  COMmon Uithoorn DAte Services

- Common date routine developed to replace all existing date routines.

- Runs on CICS, MVS, COBOL, PL/1;

- Support for country-dependent dates: weekend definitions, (fiscal) closing dates, shop dates, holidays

- ISO, USA, JIS, SYSDATE, ... formats

The Research Perspective

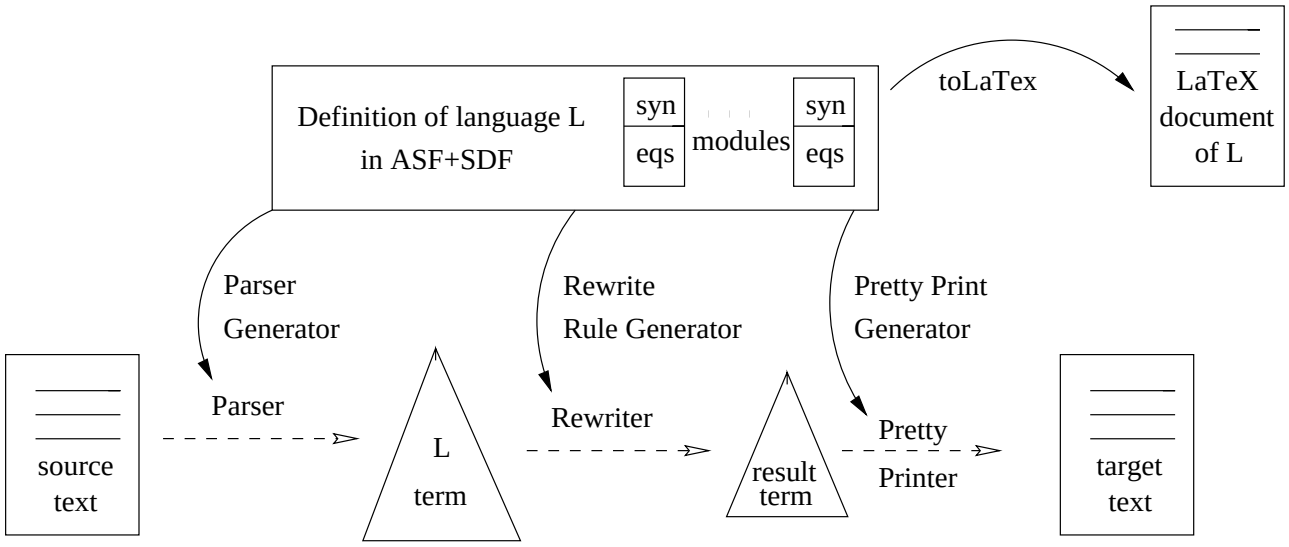
- Lexical analysis, type checking, data flow analysis, program slicing, program understanding.

- **language-independent** solutions:

  Useful commercial tools available for common languages: specialized tools required for less popular languages

- Technology beyond Y2K: apply to Euro, OO extraction, re-modularization, GUI migration, ...

Research background: ASF+SDF



- Parse source text;
- Rewrite source term to target term;
- Unparse / present resulting structure.

AMAST Series in Computing

LANGUAGE PROTOTYPING:

An Algebraic Specification Approach

by

A van Deursen
(CWI, Amsterdam)

J Heering
(CWI, Amsterdam)

P Klint
(CWI, Amsterdam & Univ. Amsterdam)

Language prototyping provides a means to generate language implementations automatically from high-level language definitions. This volume presents an algebraic specification approach to language prototyping, and is centered around the ASF+SDF formalism and meta-environment. The volume is an integrated collection of articles covering a number of case studies, and includes several chapters proposing new techniques for deriving advanced language implementations. The accompanying software is available in the public domain.

Readership: Software practitioners, graduate students and researchers in computer science.

350pp (approx.) **Pub. date: Summer 1996**
981-02-2732-9 **£48**

World Scientific
An International Publisher

World- Wide Web: http://www.wspc.co.uk or http://www.singnet.com.sg/~wspclib

Type Checking

- Year 2000 analysis = type checking.
- Find types of expressions, statically detect errors.
- Interpretation of program at the level of types:

  date − date = duration

- Interval analysis:

  $[x\text{-}y] + 1 = [(x+1)\text{-}(y+1)]$

Origin Tracking

- Specify analysis in *functional style*;
- **Automatically** maintain links from created symbols back to original term;
- *Syntax-directed* origin tracking;
- Reverse origins to get an *attributed* tree;
- Use attributes for display/further querying.

Cliches

- Build library of cliches:

  Stereotyped code patterns of common programming strategies, data-structures, and algorithms.

- Reduce *false positive* cases.
  List correct year manipulation cliches.

- Useful for year 2000 compliance *validation*.

- Difficult in general; suitable for specific (Y2K, Euro) applications?

Further Research Issues

- Language-independent syntactic querying;
- Realistic case studies;
  (modular COBOL grammar);
- Program slicing
  (aliases, goto's, inter-program)
- Clustering techniques for modularization;
- Data flow analysis: map languages to Dhal
  "Data-High-Abstraction Language"

Cooperation with Industry

- Presentations must be **problem-oriented** — *not* technology oriented.

- Knowledge must serve a purpose – enabling new products, services, etc.

- Bridge gap between industry and academia:

  publication versus commercial opportunity.

- Management needs plans/progress info before it can decide for start/continuation.

  (Brooks: wrong figures better than none).

Tacit vs Explicit Knowledge

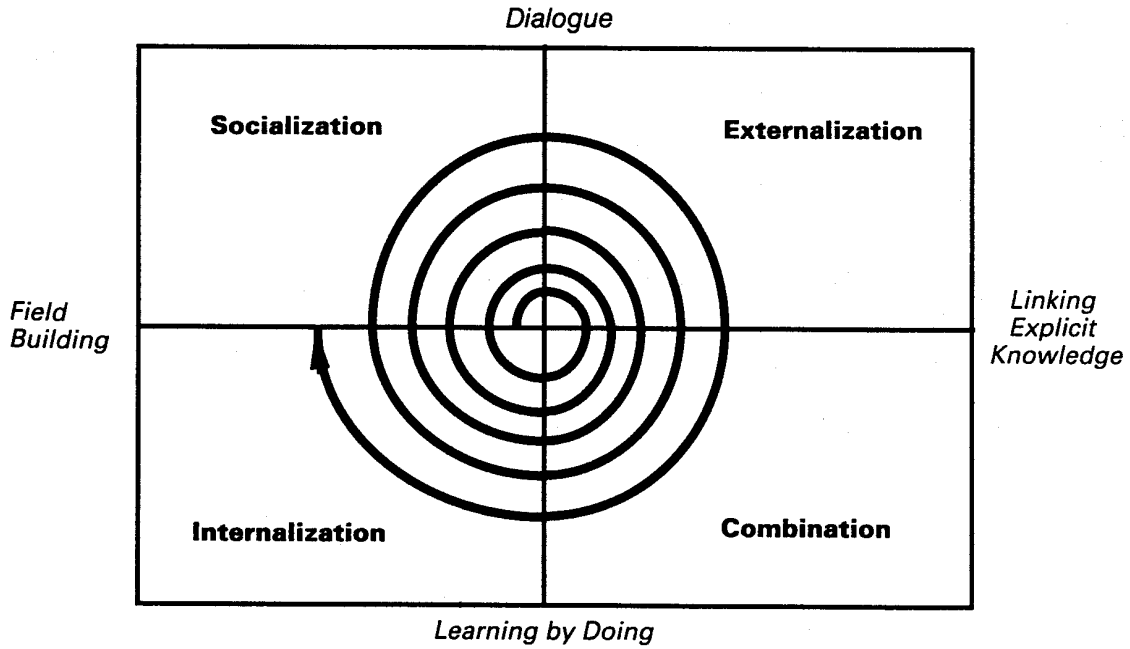


Figure 3-3. Knowledge spiral.

From I. Nonaka and H. Takeuchi, *The Knowledge Creating Company*

Knowledge Creation

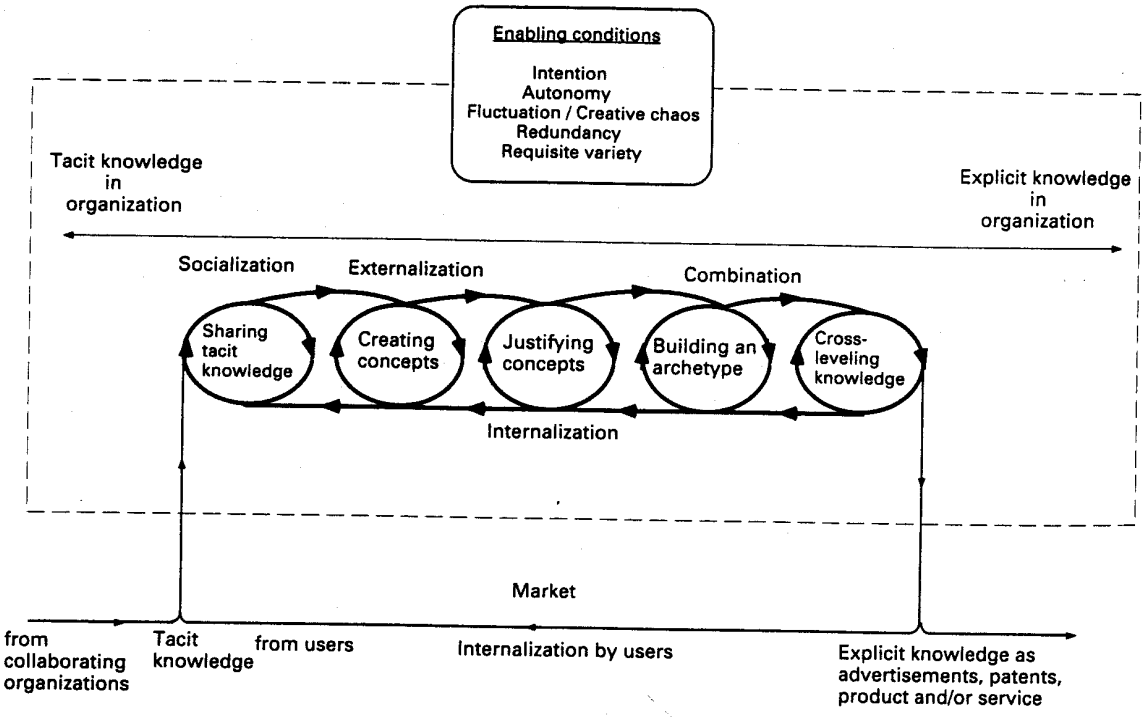


Figure 3-9. Five-phase model of the organizational knowledge-creation process.

From I. Nonaka and H. Takeuchi, *The Knowledge Creating Company*

Summary

- Software Renovation
  - Inventory;
  - Code Analysis (type checking);
  - Code Correction;
  - Research questions.
- The Year 2000 Problem.

  Code extracts, tools, costs.
- Industrial knowledge creation.