Where Does Reusable Generative Knowledge Come From?

ICSR7/GP2002

James M. Neighbors
Bayfront Technologies, Inc.
1280 Bison B9-231
Newport Beach, CA 92626 USA
+1 714 436 0322x4
James.Neighbors@BayfrontTechnologies.com

How has knowledge been gained and evolved into reusable generative forms in the past?

"The fundamental objective of GP is to consolidate and automate knowledge about construction, composition and configuration of reusable assets, thus making this knowledge itself reusable."
– GP2002 Call for Participation

"Those who cannot remember the past are condemned to repeat it."
– George Santayana (1863-1952) and probably many others

How did we get where we are? A ridiculously brief and biased history.

1940s

- machine language
- scientific computing - batch processing
- Turing and Post – conceptual symbol manipulation
- von Neumann – stored program machine, programs as data
- machines come with subroutine packages

1950s

- assembly language
- business computing - batch processing, punched tape or cards.
- unit record equipment (URE) part of system process
- Wilkes – subroutine libraries
- Chomsky – people possess innate language ability
- Backus, FORTRAN – scientific subroutine packages tied together under language
- Electronic Data Processing (EDP) – a business domain (FlowMatic)
- subroutine packages a big deal – part of user groups like SHARE
- UNCOL – write once, run anywhere, general-purpose language

1960s

- ALGOL – write once, run anywhere, general-purpose language
- business computing – first batch processing later timesharing online
- Hopper, COBOL – EDP subroutine packages under language
- McCarthy, LISP – programs as truly runtime data
- Sutherland, Sketchpad – graphics starts up - theory of interaction
- Amdahl and Mealey, System 360/OS 360 – rise of architectures
- Schorre, Meta II – meta parsing system
- Strachey, General Purpose Macrogenerator (GPM) – meta macro system
- Sammet, Programming Languages, Tower of Babel cover - languages
- Englebart, Stanford ARC – windows, mouse, tablet, chordboard - theory of interaction
- classical systems analysis – especially automation of manual systems
- IBM JCL automates unit record equipment processes
- CODASYL databases – databases startup
- EDP COBOL generators - report generators, RPG, app generators, Mark IV
- NATO Software Conferences – something has gone wrong
- McIlroy, software components – reuse theory begins
- Dijkstra, partitioning of software – architecture theory begins
- Knuth, Vol. I Fundamental Algorithms – analysis of algorithms starts up

1970s

- C – write once, run anywhere, general-purpose language
- business computing - mainframes start to decline, minicomputers and later "workstations" rise
- personal computing – people get their own weak machines, still timesharing - now with modems
- Unix escapes from AT&T
- Unix shells, pipes, and filters automate unit record equipment processes

Database 1970s

- Codd, Relational Algebra – theory for describing and manipulating data
- Zloof & de Jong – IBM System for Business Automation generator
- SQL – interpretive language for database

Interface 1970s

- Atari, Pong game (head-to-head) also Core Wars, Space Wars, and Adventure (role-playing)
- CAD, CAM, and CAE – graphics used to actually build things
- Smalltalk – graphic user interactions using objects
- Warnock, PostScript – a graphic interpreter language

AI, knowledge-based programming 1970s

- Balzer, Domain-Independent Automatic Programming
- Hewitt, Actors and planning – theory of process and AI planning techniques
- Green, Psi Program Synthesis System – has architecture
- Rich, Schrobe, and Waters – MIT Programmers Apprentice
- Prywes, generation and completeness of business apps
- Johnson, YACC/LEX – compiler generator, meta compiler
- Neighbors, Draco – a theory of reuse with domain analysis
- InterLISP – LISP maxes out, extensible language parable

Networking 1970s

- Cerf, TCP/IP - inter networking starts up
- CS at universities have email and newsgroups by end of decade

Software Engineering 1970s

- structured programming - rise of software engineering
- Boehm, software vs. hardware cost crossover curves – 1967 was crossover
- Brooks, Mythical Man-Month – rise of management methodologies
- software development methodology wars: SP,SSD,SSA,SADT,JSD

Architecture 1970s

- Bell, Threaded Code – meta theory of programs as data
- Dijkstra – theory of architecture – theory both horizontal and vertical decomposition
- Knuth – theory levels of abstraction
- Wirth, Stepwise Decomposition – theory vertical decomposition
- Parnas, On the Criteria… – theory maximum information hiding
- Coupling and Cohesion – theory of architecture formation
- deRemer, Programming in the Large vs. Programming in the Small
- Tichy and Cooprider, Module Interconnection Languages – language
- Parnas, Program Families – theory of reuse, versions

Formal Theory 1970s

- formal semantics and proofs of correctness as software method
- program transformation systems
- Bauer, formal transformation systems – five golden transformations
- Burstall and Darlington, formal synthesis of sort
- Scott-Strachey, Manna and Waldinger, formal semantics
- Backus, FP – formal language w/o side-effects

1980s

- ADA – write once, run anywhere, general-purpose language, contains packages
- business computing – Unix workstations or PCs over LANs some access to mainframes
- personal computing – individuals have their own usable machines, some on networks, some on modems
- Biggerstaff and Perlis, ITT Reusability conference
- Clocksin, Prolog – a formal theory language for logic programming
- Venture Capital startup fever
- innovation shift in universities to Unix Open Source
- Apple Macintosh – personal graphic computer
- Visicorp and Microsoft, windows on PCs
- Goguen, construction languages based on formal theory
- Lanier, virtual reality – theory of interaction
- Prieto-Diaz, Software repositories – organized libraries of source code
- Object-Oriented everything – analysis, design, implementation, databases, GUIs
- Hierarchical database models – enabled by networking
- Baxter, maintenance of knowledge based derived systems

- Scacchi, business process models including software process models leading to reuse processes and software factory model
- Formal organizational processes attempt to supplant unit record equipment process concepts
- OpenGL – a general language for graphic presentation
- US Patent Office starts to patent software

1990s

- JAVA – write once, run anywhere, general-purpose language
- business computing – cheap PCs or Unix/MS Windows/Apple workstations on cheap LANs with internet access
- personal computing – individuals have strong machines, most on wired networks, with color graphics
- Berners-Lee, HTML – language for scripting FTP over Internet (i.e., web begins) – an interaction model
- CMU FODA – theory of reuse using feature relations
- Smith, KIDS, algebraic specifications – theory of reuse using formal theory
- Srinivas, algebraic specifications – theory of reuse using formal theory
- Scheer, ARIS, business process engineering w/architecture (SAP)
- Simonyi, Intentional Programming, theory of reuse metaprogramming
- Microsoft, Windows 3.1 – first non-poisonous networked version
- Some form of windows replaces shell for most users
- Id Games, DOOM – first person shooter, cooperative game play
- web browser wars
- huge competing abstractions – APIs for OS, nets, graphics, DB, apps
- chat rooms – theory of interaction
- Microsoft DirectX – graphics and media APIs 5000 pages of documentation

- digital everything – pictures, cameras, movies, music
- Intellectual Property owners become concerned – governments and lawyers show up - Napster
- Internet business madness
- Massive Online Games – Ultima Online, Everquest, Asherons Call each with 100,000 users online – theory of interaction
- Software Reuse becomes a requirement

2000s

- Your Name Here – write once, run anywhere, general-purpose language
- business computing - individual supercomputers with high speed links to corporate blades and grids
- personal computing – individuals have multiple processor supercomputers on wireless and wired networks with trillion FLOP 3D graphic processors – gamers push the limits
- wireless joins wired networking – making more venues for computing
- mainframes return in the form of blade architectures – reconfigureable modules (processor, mem, storage) with switch fabric backplanes
- grid computing – solving problems using machines on LANs and WANs.
- even larger competing abstractions – OS, nets, graphics, DB, Apps
- Open Source continues to confuse economists
- platforms continue to proliferate: embedded, PCs, PDAs, game boxes, cell phones, wearables, blades, grids
- some surprising technology will show up – crystal store? 10TB removable? 2500 DVD movies on a $5 plastic cube – 10 second transmission time.

Evolutionary Process of Ideas

- Evolutionary process phases:
  1. “new interface”
  2. example code copying
  3. protocol and/or theory
  4. finally language
- “New interface” identified as:
  1. API documented as “man pages” – one function per page in alphabetical order (i.e., relationships unclear).
  2. Set a bunch of bits/fields in structures
  3. Call through API passing structures
  4. Get yes/no response (or system crash)
- Examples of this process: DOS, graphics, databases, networks, report writers
- Historical “new interface” examples over 40 years:
  1. Mainframe Data Control Blocks (DCBs) in 1960s
  2. X Windows and MS Windows in 1980s
  3. MS Direct X programming in 1990s
- Books for a “new interface” (e.g., Petzold) always successful
- Where does generator jump in? If/When/How wrap of DirectX?

Historic Lessons for Generators

- A write once, run anywhere, general-purpose languages is a technically good idea that never seems to survive the effect of market forces.
- The past is represented in present, perhaps not always for the best. URE-JCL
- Man-machine interaction continually becomes more intimate.
- Moore's Law shows no sign of abating.
- Software is probably maintaining its 3%/year productivity gain.
- Currently something like 66% of software development is wasted.
- Global consequences if software construction remains labor intensive.
- **PROVEN** Time to Market was key to selling reuse not NRE costs or quality
- It isn't reusable if…(works both ways, pick up good work)
  1. …it doesn't compile, link, and execute as it used to.
  2. …it can't explain what it does.
  3. …it doesn't address the user's problem.
  4. …it can't interface to existing systems.
- Don't ask what new implementation structure you can use to be trendy but instead look to see who is still around and why, especially in a generator.
- It not about the implementation target, it's about the problem being solved.
- What was the problem again?

Here's a Good Problem from 1956 and Thoughts

- This problem is really hard. Why not just run away screaming?

"For over a century Man has been able to use, for his own advantage, physical powers that far transcend those produced by his own muscles. Is it impossible that he should develop machines with "synthetic" intellectual powers that will equally surpass those of his own brain? I hope to show that recent developments have made such machines possible—possible in the sense that their building can start today. Let us then consider the question of building a mechanistic system for the solution of problems that are beyond the human intellect. I hope to show that such a construction is by no means impossible, even though the constructors are themselves quite averagely human."

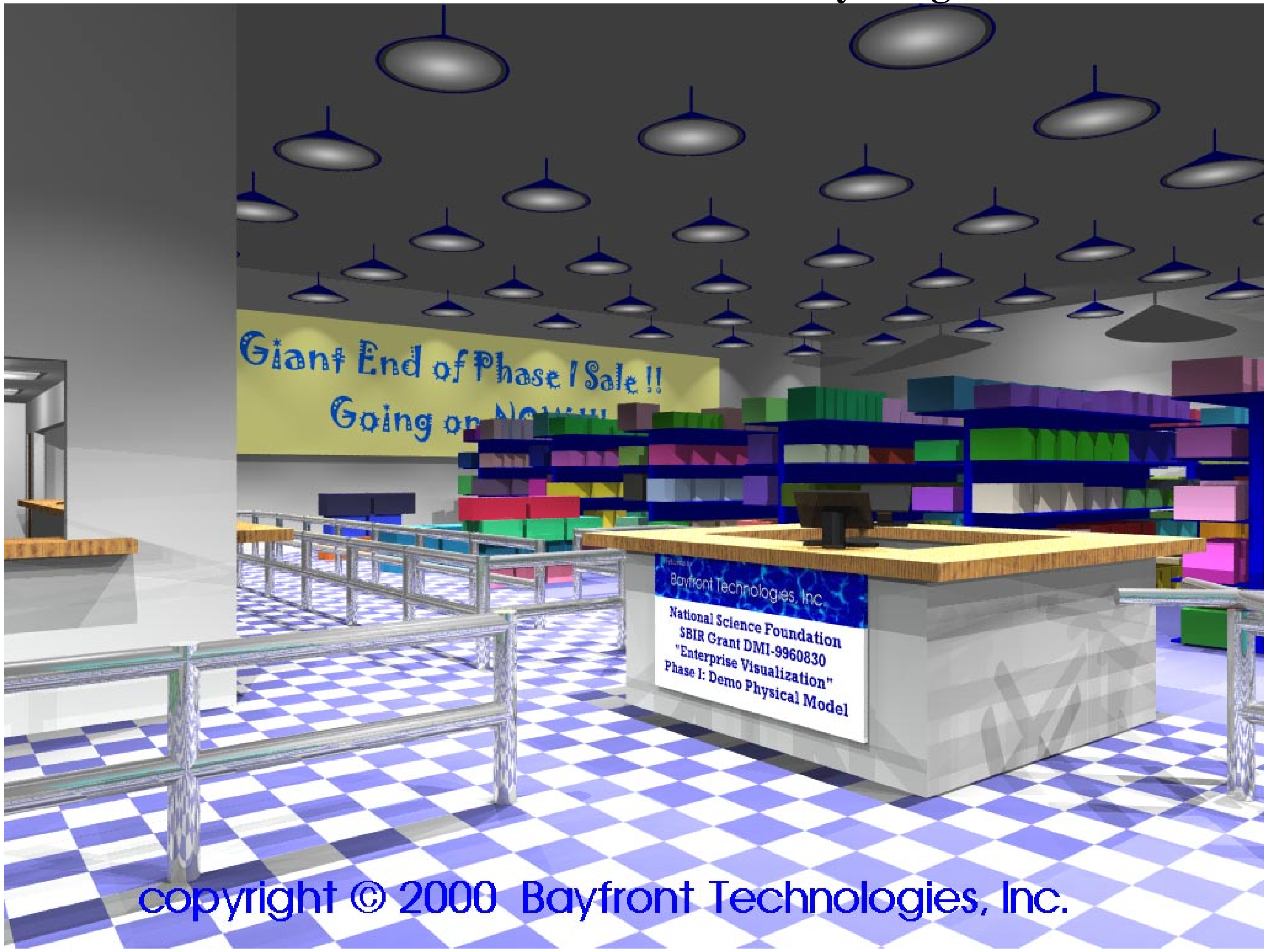
Ashby, Ross, *Design for an Intelligence-Amplifier*, ***Automata Studies***, edited by C. E. Shannon and J. McCarthy, pp. 215-234, Princeton University Press, 1956.

"According to Ashby an intelligence amplifier typically involves (a) two separate systems, one that generates descriptions of alternative courses of action and another that selects among them those appropriate, to a degree better than chance, using an externally specified criterion, (b) a circular flow of information between the two which (c) stops when the criterion is satisfied. The intelligence thereby amplified enters a system through the exogenous choices of a criterion and through the construction of the machine. The designer may not and does not need to know the full range of alternatives among which an intelligence amplifier makes appropriate choices."

Krippendorff, Klaus, ***A Dictionary of Cybernetics***, The Annenberg School of Communications University of Pennsylvania, 1986.

- The slow evolutionary process we've seen is the above cycle. We and generation are a major part of the cycle. We speed it up.
- No one gets to win or the "answer" or be "right" – only less wrong.

Generated Information is not Necessarily Program Code


Giant End of Phase I Sale !!
Going on NOW!!!
Bayfront Technologies, Inc.
National Science Foundation
SBIR Grant DMI-9960830
"Enterprise Visualization"
Phase I: Demo Physical Model
copyright © 2000 Bayfront Technologies, Inc.