Software Simplicity, and Hence Safety -- Thwarted for Decades

Edward S. Lowry
Bedford Mass. USA
eslowry@alum.mit.edu
users.rcn.com/eslowry October 2004

Abstract – **Leading providers of software could best improve their productivity and the safety of their products by large scale simplification. However, incentives to complicate other people's lives have been strong and leaders have avoided noticing opportunities for serious simplification for decades. In failing to pursue simplification (or knowledge of basic structures), the software community has shown an aversion toward fundamental issues and the public safety that seems unprecedented in other technologies. Half the labor force works to arrange pieces of information having almost no idea what is a reasonable structure for pieces of information. In effect, the software community has been keeping human minds debilitated on an increasingly large scale in order to keep them in a state of dependency. Leading organizations responsible for software safety should be regarded as a menace to public safety until their competence in software simplicity is demonstrated.**

Keywords: software; simplicity; safety

SIMPLIFICATION, LONG DELAYED

Software safety is a problem of managing complexity effectively. An obvious approach is to eliminate unhelpful complexity. Technology for doing so has fallen decades behind the leading edge.

The office of US Senator John Kerry helped to check on the delay. The main question was whether anyone working in the US Federal Government has experience working with technology which allows for simplicity in expressing software as advanced as what IBM designed by 1974 and published in 1977[1]. Since December 2000 he has presented the question to 9 government agencies. So far, none has reported finding anyone that advanced and all of them avoided answering the question.

The agencies queried were:

- Office of Technology Policy (Dept of Commerce)
- National Science Foundation
- United States Air Force
- General Accounting Office
- National Aeronautics and Space Administration
- National Institute for Standards and Technology
- Dept of Education
- Dept of Homeland Security
- Dept of Defense

As of this writing only NSF, GAO, NASA, NIST, and DHS have responded on paper. My interpretation of those responses and non-responses is that they could not readily find such competence, they are not interested in developing such competence, and they were willing to conceal its absence, even from a US Senator. To my knowledge there are only a few dozen people anywhere with such experience. They worked at Digital Equipment Corporation during the 1980s.

For decades I have observed that software leaders have shown little tolerance for serious efforts at simplification and little tolerance for the truth about simplification. That lack of tolerance can now be fairly well documented. The ACM Computing Surveys V28 No1, March 1996 [2] contains "69 short articles that span the discipline of computer science". It almost totally ignores simplicity issues -- the dominant problem with software for millions of users. I found only three sentences which were on topic, and they only acknowledged simplicity as a valid goal. Current computer languages have poor simplicity of expression compared with what IBM designed 30 years ago[1]. The correspondence with Sen. Kerry's office shows that a small but somewhat independently selected sample of software leaders are consistent in resisting the truth about simplicity and willing to violate a public trust to do so. The risks in some cases seem very large.

Results of decades of deliberate opposition to simplification include:

- Deficient math and science education due to failure to express precise information precisely.
- A flood of needless and burdensome complexity going into schools as educational technology.
- Users entangled in proprietary complexity for decades.
- Massive degradation of the quality of technical information: accessibility, usability, clarity, interoperability, ductility.
- A probable contributing factor to crash of KA801, August 1997, 226 dead.
- A possible contributing factor to 20% of US casualties in Gulf War I.
- A probable contributing factor to 3 friendly fire deaths in Gulf War II.
- A contributing factor to the August 2003 northeast power outage, 50 million affected.
- A contributing factor in many cyber-attacks.
- Failure of the FAA to upgrade its air traffic safety systems.

Based on a talk at the 2004 International Symposium on Technology and Society (ISTAS'04).

- Large part of the $60B annual cost to the US of software errors (as estimated by NIST).
- Political leaders dependent on technical advisors who are over 30 years behind the leading edge on simplicity issues.
- Burdensome technological instability.
- Large learning loads for skills of ephemeral value.
- Computer science research blunted.
- Monopolization in the software industry.
- Multi $B information system fiascos. Eg IRS.

Simplifying Technical Education and Communication

The damage caused by the aversion to simplification goes well beyond software. Simplification of software and simplification of the presentation of most kinds of technical subject matter are closely related problems. Educators massively teach people how to arrange pieces of information but they have almost no idea what is a reasonable structure for pieces of information. As discussed in Appendix A, extreme simplification constrains the data structures used to represent information including technical knowledge. It appears to lead to an enduring optimum structure for “atomic” pieces of information. That optimum can be used to provide simple, readable, integrated presentations of technical subject matter [3]. Simplifications of subject matter in the past have been localized, but this applies to almost any technical subject from Accounting to Wave mechanics.

One result of understanding optimum data object structure makes it practical to fix a fundamental (but generally unnoticed) problem with technical education: the pervasive failure to express precise information precisely enough. Because of inadequacies in traditional mathematical notations, teachers have expressed only part of their technical subject matter using precise mathematical expressions. Other information which could have been expressed precisely has necessarily been diffused into natural language text, diagrams, metaphors, examples and other informal expression. This has left the student with a large non-optional burden of distilling out the precise information and integrating it. Better understanding of the basic structure of information shows that much of that burden can be avoided. Multiple presentations can be helpful, of course, but the availability of a single, complete, integrated presentation can avoid much mystification.

Simplified representation of technical knowledge may broadly enhance communication and creativity. Use of current technical languages forces people to be obscure.

Some Prestigious Perversities

The following observations are based on personal experiences. It would be helpful if others would confirm the specifics and the overall pattern they suggest.

- In the 1970s IBM spent roughly $1B on a project whose stated goal was making development of computer applications easier but the goal was an elaborately presented sham.
- The National Science Foundation used misleading evasion in 1996 to avoid exposing its weaknesses in simplicity and again in 2002.
- In 2001 the World Wide Web Consortium volunteered that the languages they were introducing were basically 1970s technology.
- In 1992 Digital Equipment Corporation moved the leading edge of its software technology backwards by 15 years from a simplicity point of view. In 1988 they squelched discussion of data object optimality.
- The technical journals of the leading computer professional societies, ACM and IEEE, have published nothing on simplicity of expression in computer language or on the fine structure of information as advanced as what IBM [1] and DEC [4] published in 1977 and 1978.
- The IEEE misleads the public and violates its own code of ethics by granting "Certified Software Development Professional" credentials without disclosing its 30 year skill gap in simplicity.
- Hewlett-Packard presented its largest product rollout in 2003 using a marketing campaign centered on simplicity, while its patent portfolio [5] helps to keep software in a state of complexity that is "square wheel" unreasonable.
- "Simplicity is a jihad" for Microsoft, said Bill Gates in 1998 [6], as Microsoft prepared to introduce a major new programming language C#, whose simplicity of expression is decades behind the leading edge.
- About 200 participants in the High Dependability Computing Consortium had little response to questions about their abilities in software simplicity.
- MIT has over 400 computer science projects using software technology 30 years behind the leading edge on simplicity.

Interpretations

I am an engineer with limited understanding of the social processes which have produced the above failings but experience with the issues makes it appropriate to comment. Summarizing broadly, my experience has been that higher prestige has correlated well with lower tolerance for large scale simplification and lower tolerance for the truth about simplification.

Incentives to complicate other people's lives have a long history associated with a lot of social tension. Replacement of Roman numerals took centuries. Social upheaval accompanied simplification of the French legal system (Code Napoleon) and the introduction of the metric system. Simplicity has been a recurring theme in political and religious tensions.

In the computer industry complexity burdens form a large barrier discouraging customers from switching vendors. Efforts to keep those barriers high has been a major factor obstructing simplification. Anxieties about being simplified

out of jobs and billion dollar revenue streams are very real.

Large scale simplification is readily seen as threatening to cash flow streams, both large and small. Even if individuals are sympathetic to simplification, they are sensitive to its effect on colleagues. The result is classic bureaucratic resistance to simplification in larger organizations. Among software experts, simplification can obsolete much of that part of their working knowledge which consists of detailed operation of idiosyncratic systems.

Estrangement from Engineering

Software technology has not internalized traditional engineering values very effectively. The basic engineering value of getting more for less requires some quantification of benefits and costs, or at least dependable comparisons.

Simplification fits that paradigm better than most computer science. Making data objects more uniform, and up to a point simpler, produces improvements in:

- simplicity of applications,
- simplicity of language, for a given level of capability,
- generality of language

all at once. It expands the engineering envelope on three leading edges. In doing so it presses the limits for those leading edges and clarifies what is possible for each of them.

An example of inadequate use of engineering criteria occurred in the design of the Ada language. It was designed to meet requirements specified in terms of features rather than capability levels which could be compared. As a result Ada has many features but poor simplicity of expression compared with what was known several years before it was designed. Other languages such as C++, Java, and C# have been introduced with capabilities far behind known leading edges of the engineering envelope. The term "software engineering" has been widely inverted so that its practitioners are more focused on adapting people to technology rather than vice versa. The state of software engineering is on display at many bookstores where hundreds of feet of bookshelf space tell people how to tell their computers what they want done.

What may be the most influential paper in software in recent decades, "No Silver Bullet..." by F.P. Brooks Jr. [7] broadly discouraged traditional engineering approaches to software problems.

In many engineering fields, prizes for winning competitions and breaking records have helped to focus attention on how to advance the leading edges and to create institutional knowledge of best practices. In contrast, the ACM International Collegiate Programming Contest has impeded progress by specifying what technology the participants may use.

Corrective Responses

There are probably large organizations that would really welcome major simplifications. Telling them about the long term obstruction and potential for improvement could encourage them to initiate improvements or create pressures on others to do so. Individuals could also take action. The current lack of effort suggests that even small energy levels could have an impact.

Some could help by using various forums or creating forums to expose the backwardness of software providers and whatever irresponsible attitudes they have. Taxpayers and tuition payers can use their standing to extract the truth about knowledge of simplification from those with incentives to be oblivious. The coding examples in Appendix B can be used to make comparisons of simplicity of expression to test how up-to-date a person's programming language skills are. "Toward Perfect Information Microstructures"[8] can be used as an initial competence criterion for understanding of the structure of pieces of information.

High standards of competence and disclosure of competence may reasonably be demanded when public safety is at issue. Initial focus on providers in public safety areas could speed the improvement of standards in other areas.

As the potential for simplifying technical subject matter becomes clarified, analogous questioning of technical educators may become increasingly appropriate. Senior academics may be asked: "How can anyone who does not understand the structure of pieces of information be qualified to teach people how to arrange pieces of information?"

Use of colorful language in exposing backwardness is problematic but may be appropriate with people who avoid facing facts but are sensitive to public relations. The "square wheel" unreasonableness of current representations used for software and technical knowledge may be suggested or queried. Software workers and technical educators might be compared to architects and builders who have never seen a reasonably shaped brick. Journalists could search for computer scientists who can give an honorable account of themselves on simplicity issues and report their findings. The search could be framed as being in the aftermath of a hypothetical failure to intercept a nuclear armed missile.

The larger scientific and engineering communities might be persuaded to encourage more discipline among computer scientists as a way of preserving their own claim to public confidence. Scholarly study of the anomalous behavior is also appropriate. There may be unique lessons to be learned about the dependability of expert communities in general. Careful checking for sincerity of efforts to simplify seems like a long term need which grows as technology proliferates.

Acknowledgments

I wish to thank Alex Brown, Leslie Lowry, and Muriel Adcock for encouragement and assistance.

References

[1] E. S. Lowry, "PROSE Specification", IBM Poughkeepsie Laboratory Technical Report TR 00.2902, Nov 1977.

[2] "Perspectives in Computer Science". ACM Computing Surveys, Vol 28 No 1, March 1996.

[3] E. S. Lowry, "Formal Language as a Medium for Technical Education", Proceedings of ED-MEDIA 96, p407, AACE, June 1996. See also users.rcn.com/eslowry.

[4] E.S. Lowry, "Accurate description of system structure - A new standard of language quality", Proceedings of the Digital Equipment Computer Users Society, Vol 5, No2, 1978, p833.

[5] E.S. Lowry, "Data Processing System Having a Data Structure with a Single Simple Primitive", US Patent No. 5,664,177, Sept. 1997.

[6] M.J. Miller. "NT 5.0 and Beyond", PC Magazine Nov 17, 1998, pg 4.

[7] F.P. Brooks Jr, "No Silver Bullet: Essence and Accidents in Software Engineering", IEEE Computer, Vol 20 No 4, April 1987.

[8] E. S. Lowry, "Toward Perfect Information Microstructures", Oct. 2003, Unpublished, See users.rcn.com/eslowry.

[9] P. Zave, "Feature Interactions and Formal Specifications in Telecommunications", IEEE Computer, Aug 1993, pg28.

[10] K. B. Bruce, "Progress in Programming Languages", ACM Computing Surveys, Vol 28 No 1, March 1996, pg247.

[11] D.D. Chamberlain et al, "SEQUEL2: A Unified Approach to Data Definition, Manipulation, and Control", IBM J. Res. Development, Nov. 1976, pg 560.

APPENDIX A: SIMPLIFICATION OPTIMIZES DATA OBJECTS

Failure to take straightforward steps in simplification has left significant issues obscured. One such issue is the way that the discipline of simplification imposes constraints on the fine structure of the data. Those constraints create more opportunities to simplify.

When needless complexity is thoroughly removed from precise technical description, there is a kind of phase change (analogous to freezing) which forces uniformity on the structure of the component data objects used to represent the technical subject matter. The simplification leads toward a unique optimum structure for data objects for non-trivial subject matter. The optimum is expected to be hierarchically interconnected pointers [8] but that requires verification.

It appears to be one of a few dozen engineering optimizations which converge on a stable structure similar to convergence on: round wheels, vertical pillars, flat mirrors etc. In each case a deficiency can be reduced to zero and no further (vibration in wheels, shear forces in pillars, distortion in personal mirrors etc). The abrupt barrier to further improvement makes the optima permanent with well defined structures. In most cases, the deficiency can be eliminated without significant tradeoffs arising. In some cases however, the straightforward optimum needs to be compromised. For example: spherical submarines or straight line roads.

The effect of extreme simplification can be described in terms of 6 kinds of constraint imposed on the structure of data objects. The first 3 constraints are very simple and similar to the constraint that wheels be round. They require that the data objects be: purely connective, asymmetrical, and all have the same structure.

Making data objects purely connective with no internal state eliminates complexity resulting from the need to encode the internal state. Eliminating pairs of connections that are symmetrical avoids complexity which is no help since symmetrical connections would defeat the need for deterministic operation. Needs for non-deterministic operation are better provided in other ways. Using a single composite object with all the kinds of connection needed simplifies the language and does no harm since unused connections can be ignored.

There is also a need to provide for both simply expressed operations on aggregates of data (functional expressiveness) AND data structures which can accurately represent many kinds of information structure (structural expressiveness). Providing both the uniformity of data objects needed for functional expressiveness and the flexibility of data objects needed for structural expressiveness at the same time imposes additional constraints.

Empirically, there are data object structures which practicably satisfy these constraints, but only a few. Choosing among them has little effect on the simplicity achievable or the way applications would be expressed. They have no dependency on the technical subject matter being represented as long as the subject matter is not structurally trivial. This makes it plausible that there is a single permanent optimum structure for almost all data objects! With the prospect that all data objects (pretty much everywhere and always) will be the same, it makes sense to think very carefully about choosing their structure. The analysis[8] completely eliminates specific kinds of harmful complexity with little need for compromise so differing ways of measuring complexity have little effect on the results.

The questions presented by Senator Kerry also sought information about what the government agencies knew about the fine structure of information. The correspondence confirms other observations that the software community has given almost no thought to its most basic structures. From Airfoils to Zippers, technologists have given meticulous attention to their basic structures. Here too, the software community has shown an aversion toward fundamental issues that has little if any precedent.

One result of finding a unique optimum structure for data objects (in non-trivial data structures) is that the functions which operate on them can easily be merged into a single language, regardless of the technical subject matter. Formal language can then have an enduring core set of functions. These make it ***technically*** practical to develop a "universal language supporting technical literacy". Formal language can have a durable core and flexible open-endedness analogous to natural language. Commentary by software leaders on the possibility of achieving full language generality has tended toward unsupported claims that it cannot or should not be done [9,10].

APPENDIX B: SHANNON EXAMPLES

To assist in validating a 30 year delay in improving simplicity of expression, raising quality standards in future efforts, and

testing people's competence, the following list of example expressions are provided. They indicate what degree of simplicity and clarity is achievable in a multi-purpose language and roughly what was known to be achievable in 1974 as recorded in: "PROSE Specification" [1] and implemented at DEC about 1983.

These examples are translated from the first 10 examples given[11] for Sequel 2 (now SQL) in the IBM Journal of R&D, Nov 1976. For the first 10 expressions Sequel 2 (a specialized data base language) uses 130 tokens. Shannon (a multi-purpose language) uses 99 tokens. The original Sequel 2 code is omitted as irrelevant. Better comparisons would be with C++, Java, Ada, Cobol, etc. Permanent elimination of categories of needless complexity is the more significant issue.

Expression 1.

English: Names of employees in Dept. 50

Shannon: name of employee of dept(50)

Expression 2.

Eng: All the different department numbers in the Employee table.

Shan: dept_no of employee condense

Expression 3.

Eng: Names of employees in Depts. 25, 47 and 53.

Shan: name of employee of every dept where 25 or 47 or 53

Expression 4.

Eng: Names of employees who work for departments in Evanston.

Shan: name of employee of dept of Evanston

Expression 5.

Eng: List the employee number, name and salary of employees in Dept. 50, in order of employee number.

Shan: for employee of dept(50) minfirst empno
show(empno, name, salary)

Expression 6.

Eng: Average salary of clerks.

Shan: average (salary of clerk)

Expression 7.

Eng: Number of different jobs held by employees in Dept.50

Shan: count job of employee of dept(50) condense

Expression 8.

Eng: List all the departments and the average salary of each.

Shan: for dept show(it, average(salary of its employee))

Expression 9.

Eng: Those departments in which the average employee salary is less than 10,000.

Shan: dept where average(salary of its employee) < 10000

Expression 10.

Eng: The departments that employ more than ten clerks.

Shan: dept where count(its clerk) > 10