Blind Men and a Room Full of Elephants

Vadim Zaytsev
Raincode Labs
Brussels, Belgium
vadim@grammarware.net

The world of legacy software exists [1–8]. Despite numerous prognoses about certain languages going out of fashion and disappearing, many languages resist this natural process: COBOL (1959), RPG (1959), VISION:BUILDER (1960 as MARK IV), PL/I (1964), RAMIS (1965), CLIST (1970), FOCUS (1970), PACBASE (1973), BIS (1975 as MAPPER), REXX (1979), TELON (1981), CA GEN (1987 as IEF), jBASIC (1991), etc. They continue to be used extensively across billions of lines of code, literally decades after going out of fashion [9–11]. The world of legacy software evolves and grows, absorbing new languages as years go by. Only a handful of legacy systems are suitable for complete redesign and rewrite, while the bulk of software systems in finance, banking, insurance, logistics, booking, as well as in a number of other domains, are too large, too complex and too expensive to be replaced in simplistic ways. Hence, we need to develop methods and tools to analyse such software to aid in this never-ending battle by providing compilers, analysers, transformers, refactorers, testers, linters, validators.

During this keynote, participants were confronted with a carefully selected subset of features encountered by industrial compiler engineers in legacy code. Condensed by language design magic into one deliberately small language called **BabyCOBOL** [12], this collection is meant to represent the challenge of processing legacy code to the software evolution community, such as:

- parsing indentation-driven notations,
- dealing with line continuations,
- identifying keywords which are not reserved,
- resolving symbols with case insensitivity,
- syntax highlighting with insignificant whitespace,
- implementing the picture clause data type,
- preprocessing lexical imports,
- unpuzzling sufficiently qualified identifiers,
- unfolding contracted conditions,
- relying on name-based deep traversing assignments,
- using context-dependent figurative constants,
- altering unconditional branching addresses at runtime,
- branching to sentence-based code locations,
- executing paragraphs in a serial way,
- having detachable clauses in structured loops,
- combining exception handling with other features.

The participants have witnessed the first public exposure of this meeting point between the industrial community that is creating tools for handling real legacy languages (each demanding several human-years of professional investment and therefore completely unsuitable for research prototyping) and the academic community which consists of many comfort bubbles of idealised toy languages (with consistent design that rewards emergence of glorified techniques that optimistically assume overly rational language features).

We hope that by designing university courses on compiler construction, software language engineering and software evolution around BabyCOBOL instead of MiniJava or While, will help preparing students for the future, arming them with the skill to build tools to deal with the software systems of the past. At the same

In: D. Di Nucci, C. De Roover (eds.): Proceedings of the 18th Belgium-Netherlands Software Evolution Workshop, Brussels, Belgium, 28-11-2019, published at http://ceur-ws.org

time, trying out any new technique on BabyCOBOL seems feasible to quickly prototype and fit within an academic paper, and ensures the technique's applicability to a range of legacy languages.

References

[1] S. Matthiesen and P. Bjørn, "Why Replacing Legacy Systems is So Hard in Global Software Development: An Information Infrastructure Perspective," in *CSCW*. ACM, 2015, pp. 876–890. [Online]. Available: https://doi.org/10.1145/2675133.2675232

[2] J. Q. Ning, A. Engberts, and W. Kozaczynski, "Recovering Reusable Components from Legacy Systems," in *Proceedings of Working Conference on Reverse Engineering*. IEEE CS, 1993, pp. 64–72.

[3] M. Feathers, *Working Effectively with Legacy Code*. Prentice-Hall, 2004.

[4] A. A. Terekhov and C. Verhoef, "The Realities of Language Conversions," *IEEE Software*, vol. 17, no. 6, pp. 111–124, Nov./Dec. 2000. [Online]. Available: https://doi.org/10.1109/52.895180

[5] V. Zaytsev, "Open Challenges in Incremental Coverage of Legacy Software Languages," in *Post-proceedings of the Third Edition of the Programming Experience Workshop (PX/17.2)*, L. Church, R. P. Gabriel, R. Hirschfeld, and H. Masuhara, Eds., 2017, pp. 1–6.

[6] R. Khadka, P. Shrestha, B. Klein, A. Saeidi, J. Hage, S. Jansen, E. van Dis, and M. Bruntink, "Does Software Modernization Deliver What It Aimed for? A Post Modernization Analysis of Five Software Modernization Case Studies," in *ICSM*. IEEE, 2015, pp. 477–486. [Online]. Available: https://doi.org/10.1109/ICSM.2015.7332499

[7] E. Aeschlimann, M. Lungu, O. Nierstrasz, and C. F. Worms, "Analyzing PL/I Legacy Ecosystems: An Experience Report," in *WCRE*. IEEE, 2013, pp. 441–448. [Online]. Available: https://doi.org/10.1109/WCRE.2013.6671320

[8] D. Blasband, "Compilation of Legacy Languages in the 21st Century," in *GTTSE*, ser. LNCS, vol. 7680. Springer, 2011, pp. 1–54. [Online]. Available: https://doi.org/10.1007/978-3-642-35992-7_1

[9] D. Cassel, "COBOL Is Everywhere. Who Will Maintain It?" https://thenewstack.io/cobol-everywhere-will-maintain/, May 2017.

[10] T. Hartman, "COBOL blues," Reuters Graphics, http://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/index.html, Apr. 2017.

[11] R. Lämmel and C. Verhoef, "Cracking the 500-Language Problem," *IEEE Software*, vol. 18, pp. 78–88, 2001. [Online]. Available: https://doi.org/10.1109/52.965809

[12] V. Zaytsev, "BabyCOBOL," Software Language Engineering Body of Knowledge, http://slebok.github.io/babycobol, 2019.