From Software Crisis to Informational Money

Jan A. Bergstra

Informatics Institute, University of Amsterdam
email: j.a.bergstra@uva.nl

January 2, 2015

Abstract

Some comments are made on the societal impact of software engineering, with a focus on the impact of software engineering concepts on the development of informational money as well as on the concept of money at large.

1 Introduction

I met Hans van Vliet in 1982 when I moved from Leiden to the Mathematical Centre (now CWI).[1] Together with the late Jaco de Bakker, Paul Klint, and Jan Heering we constituted a fairly heterogeneous community worried about computer software, each in our own particular manner and style. In those years it became increasingly popular to think in terms of a software crisis and scientific style research was assumed to be a critical ingredient on the path towards the solution of that crisis. These five individuals maintained entirely different approaches, and each of us thought that "he got it right", alas for the other four, so to say. Apart from this ideological fragmentation our coexistence was entirely peaceful and often amusing.

2 Different styles and approaches

Paul Klint's approach was, and still is, to act as a software architect and (chief) programmer and to perform research and engineering in a co-evolutionary style. In his view research needs an application, and new software (applications, nowadays simply app's) needs research, and software tools for software construction allow that marriage in an ideal way. Jan Heering was fairly skeptical, interested in all forms of research, and very supportive and instrumental for

[1] This text is a slightly extended version of the contribution that was printed on pages 9–13 in "Software Engineering Tales", the Liber Amicorum for Hans van Vliet edited by Patricia Lago and Remco de Boer for the occasion of Hans' retirement on November 28 of 2014.

that research as well, but he believed that practice always moves its own ways and that the influence of research is fairly limited.

Jaco de Bakker maintained that one must know precisely what those programs mean in order to deal with the chaos that surrounds this so-called software crisis. He turned that view into a reputable brach of research almost singlehandedly. I thought myself that specification and formalization are key to the solution of programming problems, and years later I am still working along such lines.[2]

Hans, however, felt that logical theory is not the core of the problem, neither is the meaning of programs, nor the underdevelopment of programming skills. He maintained and still maintains the following views (as expressed in my own words):

- A software crisis manifests itself as a crisis of products with problems and that measuring these problems, or their absence in quantitative terms for instance with software metrics, is critical, however problematic and even unrewarding the theory of that sort of thing may appear to be.
- If products have problems these problems arise from production processes which are problematic as well. A top down approach to the structure of software production processes is useful. That approach must necessarily be far more general than what can be concluded or observed from one's own hands on programming experience.
- Software production admits a classification in terms of product categories. Once a product class has been taken in focus dedicated software architectures become accessible. There is no way around a systematic though "application dependent" investigation of software architecture and the systematic application of the results of that investigation.

It seems that history has proven Hans right, at least to the extent that the directions that he has been following are at this moment still very active and lively. When looking at the research output of Hans van Vliet as diplayed by Google Scholar I noticed that by far the most visible part of his work has been written in this millennium. He was consistently very active, writing a remarkable sequence of well-cited papers throughout the last 15 years: I mention for instance [1, 11, 12].

3 The power of software

What we (the persons mentioned in the introduction) failed to see so clearly back in 1982 is how critically important software would become for almost all aspects of economic and scientific life. Nowadays the software crisis has different forms and shapes. Security problems abound and most of those are software problems. The ability to deal with security issues has become a decisive factor for obtaining and maintaining a competitive edge in software

[2] For instance, adapting elementary arithmetic [9] by working with an error value as is usual in Abstract Datatypes, in [8] we use algebraic specifications in order to formalize equational reasoning in elementary mathematics. The the teaching of elementary mathematics ("rekenen-wiskunde") shows signs of having its own crisis world wide, and in [4] we suggest "rekenen-informatica" (formalization and specification in disguise) as a potential way out.

manufacturing. Software projects are still running out of budget at an alarming rate but this is now seen as a problem for local government. As software companies have proven to be extremely profitable the idea that "software is in crisis" at large has proven mistaken. Software legacy issues are still al over the place and geological layers of technically outdated but steadily operating software accumulate.

The revolution of these years is the phantastic increase in data storage capacity. The idea of a database management system carefully designed to collect, store, and process precisely the data that an organizations needs seems to have become outdated. Instead software is written that helps an organization to find its way in "big" amounts of data. This data analytics technology constitutes the software side of data science, which is the friendly face of an upcoming data crisis. indeed the impact of outdated legacy data may be even more problematic than that of legacy software.

An important manifestation of the power of software is open source software. By being a descendant of Linux, Android proves beyond any possible doubt that open source software production processes cannot be neglected by commercial software manufacturing companies anymore. Now one might think that open source impact is somehow confined to operating systems, compilers and other systems software but I believe that that is not true and that the real force of open source software production has yet to emerge.

4 Informational money

An interesting case of open source software development is the Bitcoin system. Since the (still) anonymous software authoring agent Satoshi Nakamoto wrote about Bitcoin in [13] and released the first open source implementation of it, the development of Bitcoin has been amazingly fast. I believe that Bitcoin is primarily an achievement in terms of software architecture. Blockchain technology constitutes the core of that architecture.

4.1 Nakamoto architecture

I refer to [5] for an attempt to describe in abstract terms the software architecture of Bitcoin, for which the label Nakamoto architecture in [5] was coined. The blockchain provides a public ledger, and in principle the entire transaction history of an entire user community is supposedly stored by each individual user on private equipment. Without advanced storage technology that is unthinkable, but it not as farfetched as it may seem.

Today's commenters are probably unanimous in the verdict that Bitcoin may disappear (and so Bitcoin holders with high expectations may become disappointed), but blockchain technology may have come to stay. Bitcoin uses the blockchain idea in its simplest form, but much more of that sort is to come. Blockchain technology may become vital for the solution of the data crisis by combining universal accessibility with best possible security in flexible degrees of anonymity.

The open source background of Bitcoin strengthens its acceptance in certain milieus. On the one hand that is seen as a weakness, as those circles are remote from the world of international banking. On the other hand that forces a communal effort to produce, maintain, and use

competitive software for distributed financial (or other) transactions. It is questionable that a Bitcoin-like system could have emerged as the outcome of conventional commercial software production. This sort of question merits further research.

4.2 Reduced product set finance (RPSF)

It seems that, mediated by Bitcoin and similar informational commodities, open source software may influence our perception of money. In [5, 6, 10, 3] we proposed the notion of informational money, and, with Bitcoin in mind, the more extreme notion of exclusively informational money which does away with conventional concepts of ownership. In [2] I have argued that Bitcoin provides a remarkable platform for the implementation of Islamic Finance.

However remote Islamic Finance (IF) many seem from a conventional point of view, in [7] we have argued that as a design option for a money, its built-in limitations may be somehow understood from the perspective of software engineering. By comparing it with the development of new program notations one finds an informative (potential) similarity. A programming style is often characterized by the features it does not favor and which a corresponding program notation for that reason does not (or only reluctantly) offer on purpose. For example functional programming will reluctantly offer assignments, and procedural language may discourage the use of jumps. The idea for such limitations is that as a software engineering tool a program notation is "better" if, through its built-in limitations, it contributes to the production of better quality software.[3]

Now IF may be considered as a system which in comparison with conventional financial systems strongly discourages several interest related features. In [7] this has been phrased as follows: IF is a reduced product set finance (RPSF), it limits the set of financial products that entities in a financial market are allowed to exchange.

As a tool for financial engineering it is conceivable that limitations on the product set (or the offering of financial features) lead to a better engineering of financial agent interactions.

5 Conclusion

Software and in in particular software architecture, the primary focus of the research of Hans, are much more important than merely by having practical and instrumental value, these are conceptual tools for the future. The concept of money is only one of many classical concepts that may be in need of revision with novel software architectures in mind.

References

[1] Per Olof Bengtsson, Nico Lassing, Jan Bosch, and Hans van Vliet. Architecture-level modifiability analysis (ALMA), *J. of Systems and Software* vol. 69 (1-2), pages 129-147 (2004).

[3] RISC (Reduced instruction set computing) architectures are another example taken from IT where "less" means "more" in some sense.

[2] Jan A. Bergstra. Bitcoin and Islamic Finance.
University of Amsterdam, Informatics Institute, Report TCS1406, April 2014, http://www.science.uva.nl/pub/programming-research/tcsreports/TCS1406.pdf, also on vixra.org/1501.0021, (2014).

[3] Jan A. Bergstra. Bitcoin: not a currency-like informational commodity.
University of Amsterdam, Informatics Institute, Report TCS1406, April 2014, http://www.science.uva.nl/pub/programming-research/tcsreports/TCS1406.pdf also on vixra.org/abs/1501.0005, (2014).

[4] Jan A. Bergstra, Inge Bethke, and Alban Ponse. Rekenen-Informatica.
University of Amsterdam, Section Theory of Computer Science, Report TCS1412. https://ivi.fnwi.uva.nl/tcs/pub/tcsreports/TCS1412.pdf (2014).

[5] Jan A. Bergstra and Karl de Leeuw. Bitcoin and Beyond: Exclusively Informational Money. arXiv:1304.4758v2 [cs.CY] (2013).

[6] Jan A. Bergstra and Karl de Leeuw. Questions related to Bitcoin and other informational money. arXiv:1305.5956v2 [cs.CY] (2013).

[7] J.A. Bergstra and C.A. Middelburg. Preliminaries to an investigation of reduced product set finance. *JKAU: Islamic Economics*, 24(1):175–210 (2011).

[8] Jan A. Bergstra and Alban Ponse Division by zero in common meadows arXiv:1406.6878 [math.RA] (2014).

[9] J.A. Bergstra and J.V. Tucker. The rational numbers as an abstract data type. *Journal of the ACM*, 54 (2), Article 7 (2007).

[10] Jan A. Bergstra and Peter Weijland. Bitcoin: a money-like informational commodity. arXiv:1402.4778, (2014).

[11] Jaap Gordijn, Hans Akkermans, and Hans van Vliet, Business modelling is not process modelling. In: Conceptual modeling for e-business and the web, pages 40–51, Springer, (2000).

[12] Philippe Kruchten, Patricia Lago, and Hans van Vliet, Building up and reasoning about architectural knowledge. QoSA 2006, LNCS 4214, pages 43-58 (2006).

[13] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system. http://Bitcoin.org/Bitcoin.pdf (2008).