Toxic Concepts in Systems Analysis and Design: The Systems Development Lifecycle

Paul Ralph
Lancaster University
paul@paulralph.name

ABSTRACT

This position paper argues that the the Systems Development Lifecycle is a Toxic Concept, i.e., an idea that is both false and harmful. SDLC is defined and its criticisms are summarized. A process theory, the SCI Framework, is suggested as an alternative.

Keywords

Systems Development Lifecycle, process theory, toxic concepts, design science

INTRODUCTION

This paper is motivated by the many dichotomous discussions I have had with academics and practitioners concerning The Systems Development Lifecycle (SDLC). Anecdotally speaking, for every person I have met who believes that SDLC is an absurdity no one takes seriously, I have met another who believes that SDLC is the fundamental basis of all systems development. Many people in each group appear unaware that the other group even exists. This necessitates an open discussion of the role of SDLC in design research, practice and education. I open this discussion by asserting the following.

Position: *The Systems Development Lifecycle is a toxic concept*.

SDLC (Figure 1) is a somewhat nebulous concept that may refer to:

1. A process theory (Van de Ven et al. 1995) that describes systems development in terms of a discrete number of sequential phases, including *planning*, *analysis*, *design* and *coding*, or variants thereof.
2. A systems development method (SDM) (Wynekoop et al. 1997) resembling one or more variants of the Waterfall Model (Royce 1970)
3. Any set of steps for creating a technological artifact

For the purposes of this paper, *toxic concepts* are ideas that are both *false* and *harmful* (defined formally below). For example, in educational psychology, the blank slate hypothesis (the view that the mind lacks innate traits) is a toxic concept as it has been refuted by neurobiological studies and deleteriously affects educational methods (Pinker 2002).

Toxic Concept: *a theory, construct, argument, technology, or other idea that 1) is untrue, inaccurate or refuted and 2) causes confusion, practical hardship or deleterious action*.

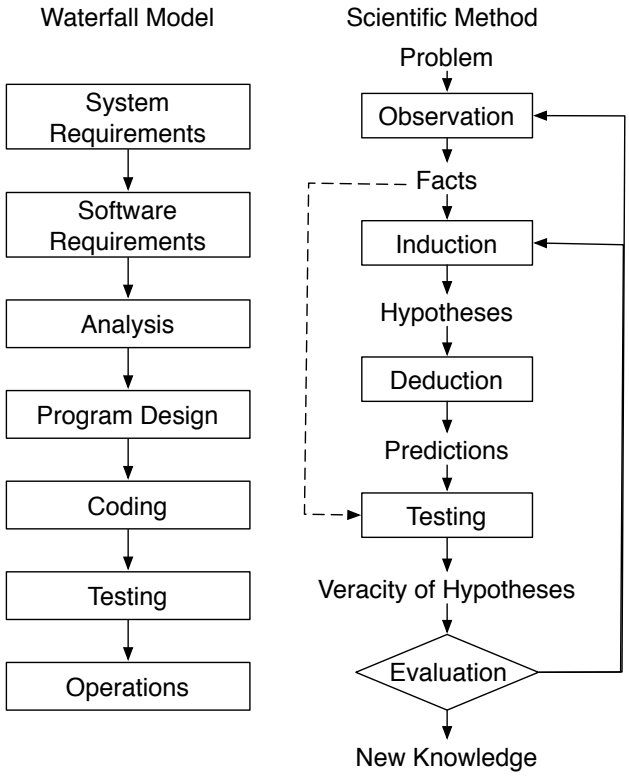

Figure 1. SDLC (left) and Cycle of Scientific Inquiry (Roozenburg et al. 1995) (right)

SDLC AS A THEORY

A process theory is an explanation of how and why an entity changes and develops (Van de Ven et al. 1995). Coupling a model of SDLC (as in Figure 1) with a claim that it either describes all systems development or (equivalently) that its elements or structure are inherent to development is commensurate with claiming that SDLC is a process theory.

Though rarely stated, implicit claims that SDLC is a veracious process theory pervade research, teaching and practice. For example, in a well-cited paper in MIS Quarterly, Fitzgerald (2006) states that "in conventional software development, the development lifecycle in its most generic form comprises four broad phases: planning, analysis, design, and implementation" (p. 3) and then

describes the presence of these phases in open-source software development. In a popular introductory MIS textbook, Laudon et al. (2009) state that "systems development … consist[s] of systems analysis, systems design, programming, testing, conversion and production and maintenance … which usually take place in sequential order." Similarly, at the time of writing, the SDLC Wikipedia article states that "SDLC adheres to important phases that are essential for developers, such as planning, analysis, design, and implementation." Moreover, the traditional SDLC phases are explicitly adopted in the official IEEE Guide to the Software Engineering Body of Knowledge (Bourque et al. 2004). In summary, implicit and explicit claims of SDLC universality remain prevalent in research, teaching and practice.

The claim that SDLC describes all systems development has been unequivocally refuted by empirical research. This finding is independent of the precise phases employed or their sequence. For example, in field studies of expert designers, Schön found evidence indicating that designers do "not keep means and ends separate" or "separate thinking from doing" (1983, p. 69). Meanwhile, Bansler & Bødker (1993) found that developers may claim to follow a method while practically ignoring it. Additionally, in a study of "a large scale system development effort", Zheng et al. (2007) found that "home-gown methods and ad hoc activities appear to dominate the day-to-day practices of systems development" (p. 1). Furthermore, Ralph (2010a) found that the a generalized model of SDLC does not accurately represent software design practice. Moreover, the XP and Agile Development Conferences feature multitudinous experience reports irreconcilable with SDLC-thinking. More generally, "any form of life cycle is a project management structure imposed on system development. To contend that a life cycle scheme, even with variations, can be applied to all system development is either to fly in the face of reality or to assume a life cycle so rudimentary as to be vacuous" (McCracken et al. 1982, p. 30).

SDLC AS A METHOD

Some argue that a waterfall-like SDLC is a SDM, i.e., it is one way to build software. This view is common in MIS research (e.g., Lee et al. 2010; Sircar et al. 2001). Furthermore, Royce originally proposed SDLC as "a more grandiose approach to software development" than a method comprised only of analysis and coding (1970, p. 328). This view was elaborated by Boehm (1988). SDLC is often contrasted with various Agile methods (Abrahamsson et al. 2002), and a case is made that each is effective in different circumstances. This is the approach taken by several introductory MIS textbooks (e.g., Baltzan et al. 2008; Kroenke et al. 2010). Moreover, proponents of agile methods often position them as more effective alternatives to SDLC (e.g., Beck 2005; Schwaber et al. 2001).

Positioning SDLC as a method involves two claims: 1) that SDLC is in some way effective; 2) that it is possible, in principle, to develop systems using it.

The claim that SDLC is an effective method lacks empirical support. I have never encountered an experimental study comparing SDLC to alternative methods. I have found no multiple-case studies contrasting teams using SDLC with teams employing other methods. I have identified no analyses of secondary data evaluating the effect of SDLC on outcome variables such as project success or software success. I did find one survey evaluating SDLC against a prototyping methodology (Palvia et al. 1990); however, it explicitly assumed that SDLC describes all software development, thus its support for SDLC as a method is circular. In summary, I found no credible evidence that SDLC is effective in any sense. While this does not refute the claim, we have several reasons to believe SDLC is ineffective. The author generally credited with proposing SDLC affirmed that its simplest version "has never worked on large software development efforts" (Royce 1970, p. 335). Furthermore, SDLC ignores end-user development and end-user involvement outside of requirements specification and "rigidifies thinking", increasing developers' resistance to change (McCracken et al. 1982, p. 31). Moreover, the tightly-coupled nature of the life cycle stages exacerbates problems by making it difficult to modify either requirements or the software without setting off complex downstream or upstream revisions (Gladden 1982, p. 36). Additionally, SDLC is "risky and invites failure" because testing occurs at the end and many of the phenomena of interest are "not precisely analyzable" (Royce 1970, p. 329). Also, SDLC justifies intensive upfront analysis by citing a steep cost-of-change curve, but the steepness of the curve is not a feature of software projects but a feature of waterfall-like processes (Ambler 2002; Beck 2002). Finally, SDLC assumes that human developers are capable of correctly getting the requirements, design and tests correct on the first try. As the burden of proof (of effectiveness) for any method lies with its proponents, no proof has been provided, and we have many strong reasons to question SDLC's potential effectiveness, on the balance of evidence, this claim is unsupported.

The claim that SDLC describes *any* systems development practice can be challenged on several grounds. First, "the development process itself changes the user's perceptions of what is possible, increases his or her insights into the applications environment, and indeed often changes that environment itself;" therefore, "systems requirements cannot ever be stated fully in advance, not even in principle" (McCracken et al. 1982, p. 31). Second, the descriptions of the stages of SDLC are "imprecise, ambiguous, incomprehensible" (Curtis et al. 1992, p. 75). Third, SDLC separates analysis from design, where the former generates an understanding of the problem and the latter generates a solution, without providing any guidance as to how the solution is generated. Since software problems are unbounded (unlike arithmetic problems), even a deep understanding of the problem does not necessarily make the solution evident. Fourth, a waterfall-like SDLC confuses "phases" with activities; for example, analysis is not a phase, it is an activity that is necessary not only for requirements modeling but also for

coding and testing. Therefore, on a balance of evidence, this claim is also unsupported.

SDLC AS A CLASS OF PHENOMENA

Some suggest that a SDLC simply describes a development project's stages (Alexandrou 2010); hence, different projects have different SDLCs. In this interpretation, SDLC obviously cannot be false (and therefore cannot be toxic) as it is not coupled with any empirical claim. I return to this issue below.

HOW SDLC CAUSES HARM

SDLC causes identifiable harm in many ways. First, as SDLC is presented as either a valid description or an effective method of software design in many SA&D courses and texts, it confuses students regarding the true nature of software development and encourages unjustified faith in a deeply flawed approach. Second, it creates conflicts between managers (who try to drive projects through phases, schedules and costs) and developers (who do not adopt these phases and cannot accurately estimate costs) (Beck 2005). Third, insofar as SDLC-thinking underlies design methods, tools and practices, their practical usefulness is hampered. Fourth, the prevalence of SDLC-thinking impedes publishing engineering and behavioral research on design aides rooted in more realistic design theories. Finally, I suggest that as "SDLC" has become inextricably confounded with the stages of the waterfall model, using the same term to denote any sequence of stages resulting in a technological artifact only exacerbates the confusion and conflict described above.

CONCLUSION

In conclusion, if SDLC is considered a theory, substantial empirical findings refute its veracity. If SDLC is considered a method, no scientific evidence supports its effectiveness, and many sound arguments that it is impossible in principle exist. These arguments hold regardless of precisely how the stages are divided (e.g., five-stage model, seven-stage model) and whether backtracking or loops are included. Moreover, although using SDLC to denote any development process is not wrong, when combined with its historical usage, this too exacerbates confusion. Furthermore, as software development literature is replete with unreferenced, unsupported empirical claims regarding the centrality of SDLC concepts, SDLC causes significant harm and confusion among practitioners, managers and students alike. Therefore, *the Software Development Lifecycle is a toxic concept.*

AN ALTERNATIVE TO SDLC

Identifying problems with SDLC is of limited usefulness without suggesting alternatives. Fortunately, better alternatives are available. SDLC may be replaced by an alternative software design process theory, specifically the *Sensemaking-Coevolution-Implementation (SCI) Framework* (Ralph 2010a; Ralph 2010b). Whereas SDLC is a *lifecycle* process theory (Van de Ven et al. 1995), SCI (Figure 2, Table 1) is a *teleological* process theory, i.e., an explanation of how and why an entity changes wherein change is manifested by a goal-seeking agent that engages in activities in a self-determined sequence (Churchman 1971; Singer 1959; Van de Ven et al. 1995). The core claim of SCI is that developers engage in three activities to produce software – making sense of the project context, iterating between ideas about the context and artifact, and implementing the artifact in code. Using a questionnaire study, Ralph (2010a) found that SCI describes software development practice more accurately than SDLC.

BIBLIOGRAPHY

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. *Agile software development methods: Review and analysis* VTT Publications, Espoo, 2002.

Alexandrou, M. "Systems development life cycle (SDLC),"[online]: http://www.mariosalexandrou.com/methodologies/systems-development-life-cycle.asp, 2010.

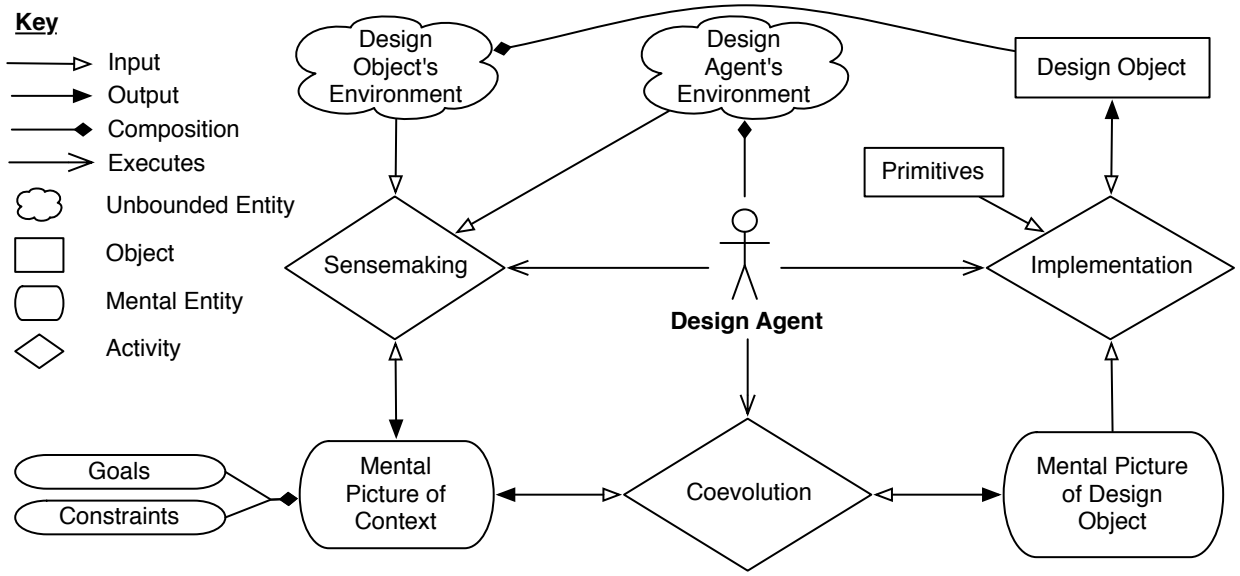

Figure 2. The Sensemaking-Coevolution-Implementation Framework

Concept / Activity	Meaning
Constraints	a restriction on a structural or behavioral property of the design object
Design Agent	an entity or group of entities that is capable of forming intentions and goals and taking actions to achieve those goals, and that specifies the structural properties of the design object
Design Object's Environment	the totality of the surroundings where the design object exists or is intended to exist
Design Agent's Environment	the totality of the surroundings of the design agent
Design Object	a (possibly incomplete) manifestation of the mental picture of design object
Goals	optative statements about the effects the design object should have its environment
Mental Picture of Context	the collection of all the design agent's beliefs about its and the design object's environments
Mental Picture of Design Object	the collection of all the design agent's beliefs about the design object
Primitives	the set of entities from which the design object may be composed
Sensemaking	the process by which the design agent perceives its and the design object's environments and organizes these perceptions to create or refine the mental picture of context
Coevolution	the process by which the design agent simultaneously refines its mental picture of design object based on its mental picture of context, and vice versa
Implementation	the process by which the design agent generates or updates a design object using its mental picture of design object

Table 1. Concepts and Activities of the SCI Framework (adapted from Ralph 2010b)

Baltzan, P., and Phillips, A. *Business driven information systems* McGraw-Hill Irwin, 2008.

Beck, K. *Extreme programming explained: Embrace change*, (2nd ed.) Addison Wesley, Boston, MA, USA, 2005.

Boehm, B. "A spiral model of software development and enhancement," *IEEE Computer* (21:5), May 1988, pp 61-72.

Bourque, P., and Dupuis, R. (eds.) *Guide to the software engineering body of knowledge (SWEBOK)*. IEEE Computer Society Press, 2004.

Churchman, C.W. *The design of inquiring systems: Basic concepts of systems and organization* Basic Books, New York, 1971.

Curtis, B., Kellner, M.I., and Over, J. "Process modeling," *Communications of the ACM* (35:9) 1992, pp 75-90.

Fitzgerald, B. "The transformation of open source software," *MIS Quarterly* (30:3) 2006.

Gladden, G.R. "Stop the life-cycle, I want to get off," *SIGSOFT Software Engineering Notes* (7:2) 1982, pp 35-39.

Kroenke, D., Gemino, A., and Tingling, P. *Experiencing MIS*, (Second Canadian ed.) Pearson Prentice Hall, Toronto, 2010.

Laudon, K., Laudon, J., and Brabston, M. *Management information systems: Managing the digital firm*, (Fourth Canadian ed.) Pearson, Prentice Hall, Toronto, 2009.

Lee, G., and Xia, W. "Toward agile: An integrated analysis of quantitative and qualitative field data," *MIS Quarterly* (34:1) 2010, pp 87-114.

McCracken, D.D., and Jackson, M.A. "Life cycle concept considered harmful," *SIGSOFT Software Engineering Notes* (7:2) 1982, pp 29-32.

Palvia, P., and Nosek, J. "An empirical evaluation of system development methodologies," *Information Resource Management Journal* (3:3) 1990, pp 23-32.

Pinker, S. *The blank slate: The modern denial of human nature* Penguin, 2002.

Ralph, P. "Comparing two software design process theories," International Conference on Design Science Research in Information Systems and Technology (DESRIST 2010), Springer, St. Gallen, Switzerland, 2010a.

Ralph, P. "The sensemaking-coevolution-implementation framework of software design," *MIS Quarterly* ((under review)) 2010b, p 76 pages.

Roozenburg, N., and Eekels, J. *Product design: Fundamentals and methods* Wiley, Chichester, UK, 1995.

Royce, W.W. "Managing the development of large software systems: Concepts and techniques," Proceedings of Wescon, 1970.

Schwaber, K., and Beedle, M. *Agile software development with scrum* Prentice Hall, 2001.

Singer, E.A. *Experience and reflection* University of Pennsylvania Press, 1959.

Sircar, S., Nerur, S.P., and Mahapatra, R. "Revolution or evolution? A comparison of object-oriented and structured systems development methods.," *MIS Quarterly* (25:4), December 2001, pp 457-471.

Van de Ven, A.H., and Poole, M.S. "Explaining development and change in organizations," *The Academy of Management Review* (20:3), July 1995, pp 510-540.

Wynekoop, J., and Russo, N. "Studying system development methodologies: An examination of research methods," *Information Systems Journal* (7), January 1997, pp 47-65.