Information Sciences 181 (2011) 4993–5009

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A quantitative model for software engineering trends

Latifa Ben Arfa Rabai *, Yan Zhi Bai, Ali Mili

Institut Superieur de Gestion, Université de Tunis, Bardo 2000, Tunisia
New Jersey Institute of Technology, Newark, NJ 07102, USA

ARTICLE INFO

Article history:
Received 3 November 2009
Received in revised form 2 May 2011
Accepted 1 July 2011
Available online 14 July 2011

Keywords:
Bottom up approach
Intrinsic factors
Extrinsic factors
Historical trends
Software technology trends
Successful trends

ABSTRACT

Many decision-makers in industry, government and academia routinely make decisions whose outcome depends on the evolution of software technology trends. Even though the stakes of these decisions are usually very high, decision makers routinely depend on expert opinions and qualitative assessments to model the evolution of software technology; both of these sources of decision-making information are subjective, are based on opinions rather than facts, and are prone to error. In this paper, we report on our ongoing work to build quantitative models of the evolution of software technology trends. In particular, we discuss how we took specific evolutionary models and merged them into a single (general-purpose) model. The original specific models are derived empirically using statistical methods on trend data we had collected over several years, and have been validated individually; in this paper we further validate the generic (general-purpose) model.

1. Introduction

Many decision-makers in industry, government and academia routinely make decisions whose outcome depends on the evolution of software technology trends. For example, a corporate manager may take decisions pertaining to the adoption of a particular technology, the adherence to a particular standard, the selection of a particular development environment, etc. A government official may take decisions pertaining to mandating a particular standard, adopting a particular technology, or acquiring a particular product. An academic officeholder may take decisions pertaining to curriculum content, to research direction, or to platform adoption. All these decisions carry important stakes for the organizations at hand and sometimes for the objects of the decisions; yet, they are often made with relatively little hard data, without any duly validated models, relying instead on expert opinions and qualitative assessments.

The work we present in this paper aims to develop quantitative models for the evolution of software technology trends. As we envision them, these models should help us understand what factors drive the evolution of a software technology, and to what extent; in practice, we envision that they complement expert analysis, and provide quantitative measurements that experts may use as a basis for their recommendations and assessments. To this effect, we represent the evolution of a software technology product by a set of relevant quantitative factors, including intrinsic factors (that are static) and environmental factors (that are time-dependent); we use these factors to collect factual quantitative data about historical trends, then use the data to perform statistical analysis.

The history of software technology is replete with examples where technology trends evolve in unpredictable/irrational ways, giving us ample motivation to develop quantitative models. For the sake of argument, we consider the following anomalies, taken from the rich history of programming languages.

* Corresponding author at: Institut Superieur de Gestion, Université de Tunis, Bardo 2000, Tunisia.
E-mail address: latifa.rabai@isg.rnu.tn (L.B. Arfa Rabai).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.07.004

- Fortran has had much more success and a much deeper impact than Algol, a language from the same generation (late 50s/early 60s), which is much more structured, much more orthogonal, and much better designed [2].
- Pascal, a language that was designed by a lone Professor as a teaching tool in a first programming course, was much more successful and had a much deeper impact than PL1, a language of the same generation (mid to late sixties), that was developed and promoted by IBM, one of the most influential organizations in the computing field at the time [10,26].
- The C programming language, a special purpose (systems oriented) language developed with limited ambitions (to accompany a nascent operating system) by two researchers has evolved into a major milestone in programming language design, influencing a wide range of subsequent languages, including C++, C#, Object-C, BitC, D, Java, JavaScript, Perl, PHP, etc. [13].
- Despite being developed as part of a worldwide competition (in the late seventies), despite embodying the most advanced concepts of its time (ADT's, exception handling, genericity, information hiding, specification vs implementation, etc.), and despite enjoying the long term backing of one of the most powerful governmental organizations in the world (the US Department of Defense), Ada had very limited success and made relatively little lasting impact on the discipline of programming language design or the discipline of software engineering [12].
- Despite being the focal point of a worldwide research effort in fifth generation computing (eighties and early nineties), despite boasting significant attributes in terms of ease of use, and despite tireless support from many governmental agencies worldwide (e.g. Japan's MITI), Prolog had very limited success as a programming language, and is used in precious few applications [5].
- Even though it was designed by a lone researcher, as a language to support a specific project on a very specific computing devise (a set-top device), Java has quickly evolved into a very widely used programming languages, that is widely adopted as a teaching tool, and is a de facto standard for web applications [9].

The history of operating systems is no less rich in paradoxes, with systems such as Unix, Linux and DOS starting from relatively modest means to become great successes, whereas systems such as Multics and OS 360 emerging with massive backing from influential organizations in industry, government and academia, to end up with relatively little impact in the long range. A decision-maker in industry, government, or academia would be forgiven for betting on the wrong horse when the laws that determine success or failure are so mysterious: success arises in the most unlikely, most modest quarters, and eludes the most likely, best supported contenders.

The models we discuss in this paper are primarily empirical, rather than analytical, hence they will not give us any insights into how these anomalies came about. What our models try to do, instead, is attempt to capture all the relevant factors that determine the evolution of a software technology trend, and attempt to derive evolutionary laws based on factual observations and statistical analysis. Some of the applications we envision for our models include: the ability to predict how much popularity a software product will have with a given segment of the stakeholder community (academia, government, industry, independent users, professional organizations, etc.); the ability to characterize the attributes of successful products with each segment of the community; the ability to analyze the impacts that the various segments of the community have on each other's choices (for example, to what extent does the success of a software technology with one segment of the community influence its success with another, and with what time lag?). For all their flaws, the opinions and judgments of experts remain the main resource for making decisions that depend on the evolution of software technologies; what our models add to the mix are quantitative, empirical estimates that stem from factual data and from validated scientific analysis, and allow the decision maker to base her/his decision on hard facts rather than mere speculation.

In section 2 we briefly discuss alternative approaches to modeling software technology evolution and outline the main attributes of the approach we propose. In section 3 we present the empirical background of our project, and in section 4 we present our quantitative approach, along with its preliminary results. In the conclusion, we summarize and assess our main findings, then outline directions of future research.

2. Approaches to modeling software technology trends

We distinguish, broadly, between two families of approaches to modeling the evolution of software engineering trends; we study them below, in turn.

2.1. Top down approach

The first approach we have considered can be characterized as being analytical, and proceeding top down. This approach proceeds by adopting a general evolutionary model (pertaining to technology transfer, or to scientific evolution, etc.), then specializing it to software technologies; we can characterize it as being *deductive*. Cowan et al adopt this approach as illustrated in Fig. 1 when they break down the lifecycle of a product or idea into three partially overlapping phases [6]: Research phase, Technology phase, and Market phase. We explore evolutionary models for each phase.

- Research phase. To model this phase, we have considered research on epistemology [21,14] and tried to specialize it to software.