Analysis of Information Technology Evolution:  
An Entropy Tree Perspective

Seongwoo Kim, Student Member, IEEE, Sunshin An, Member, IEEE

Department of Electrical Engineering, Korea University, Seoul, Korea
Email: (swkim, sunshin)@dsys.korea.ac.kr

Abstract—In this paper we analyze the nature of the evolution of technology, particularly information technology. The pace of evolution in IT demands rapid adaptation by programmers and engineers. This means keeping abreast of changing programming languages and techniques. This paper proposes a technology entropy tree which illustrates the hierarchical relationship between previous and successor technologies. The purpose of our proposal is to give programmers and engineers a deeper understanding of the patterns of technological evolution in IT in order to enable them to effectively keep up with rapid changes in technology. We use examples from programming languages to describe properties of technology evolution. We also present criteria based on a technology entropy tree for comparing technologies and selecting career paths. Finally we propose a feasible strategy to cope with fast-paced technology change and growth using prior results.

Index Terms—Technology evolution, entropy, technology adaptation, programming language.

I. INTRODUCTION

Every year corporations spend billions on IT, based largely on the belief that investing in IT technology will improve productivity, increase sales, and reduce costs. The IT industry has grown dramatically over the decades and it is expected to continue to enlarge. IT workers are in high demand, not merely within the IT industry, but in almost every other industry as well.

However, the industry has described a shortage of IT professionals. The ITAA released [11] a report showing a rising demand for IT workers going unmet by the educational system. Nakayama and Sutcliffe [12] showed that a shortage of programmers has been an on-going problem for decades. Cappelli [13] pointed out that an important contributor to the IT labor shortage is a lack of interest by employers in hiring older workers. Although companies want experienced IT workers, veteran IT professionals command high salaries based on their experience and the expertise they have developed. However, that experience may not be worth as much as they believe. Companies tend to consider older, more experienced IT workers unable to maintain mastery of progressing technology. An employer would prefer to spend more money and time training a younger programmer who will be willing to work longer hours for less money. Nevertheless, a young programmer becomes old in a decade. A fresh young programmer who becomes old may have to switch to a non-technical job.

These phenomena derive from the fact that the field of IT changes rapidly. New ideas, more advanced programming languages, and cutting-edge tools are introduced continuously into the IT industry and marketplace. The latest skills and technology are outdated in a year. Programmers and engineers must keep up with new technologies until their retirement. This aspect of IT has accelerated the IT labor shortage.

Understanding the pattern of technology evolution provides tools to cope with rapid technological change. The issue of technology evolution cannot be easily described because the workings of technology are conceptualized as the outcome of linkages between multiple elements like market situations, historical events, and scientific advancements. There are enormous cases of product development failure in an environment undergoing technological change, even within well-established, sophisticated organizations, e.g., [1], [2].

Understanding the dynamics of technological development has become accepted as a key challenge for success in a technology-intensive environment. The changes and transitions of technology evolution have been explored by several authors. Utterback and Abernathy [3] proposed the product-process life cycle theory, which provides a useful model for understanding the pattern of industrial technology evolution in the relationship between product innovation and process innovation. This proposal is the representative model for technological innovation and evolution in its early stages. Foster [4] used S-curves to elucidate that radically new technologies are often developed and brought into an industry by newly entering firms, rather than by the established leaders. It is called attacker’s advantage. Foster explains why leading firms lose their positions of industry dominance. Leading firms have the tendency to strengthen and refine maturing technological approaches. However the primary source of their erosion comes from the failure to spot emerging successor technolo-
gies in a timely fashion. Christensen [5] explored the strengths and shortcomings of S-curve theory from an individual firm's perspective using the example of the disk drive industry. He showed that patterns of technology evolution are different in the component industry and the architectural industry. This proposal can also be applied to the software industry. Worlton [6] investigated the technological evolution of hardware, recent high-performance computing, and extracted patterns of technological change. Several authors have discussed the patterns of technology evolution and proposed strategies to cope with radical technological changes and framework for analysis such as the ecosystem perspective. (See also [7], [8], [9], [10])

While most of the prior authors discussed technology evolution from a business and industry perspective, we approach it from the perspective of an employee and a single firm. In this paper we propose a Technology Entropy Tree (TE-Tree) and an algorithm and framework for exploiting the TE-Tree: a new method of depicting and analyzing technology evolution. Our approach using the TE-Tree enables IT workers to understand the technology evolution of IT and to keep up with rapid technology change.

The paper is outlined as follows. Section II gives an introduction of a technology entropy tree with examples. Section III describes an algorithm for determining the entropy of each technology. Section IV describes our research methodology and results with programming language. A framework using a TE-Tree and a case study are presented in Section V. Finally, we conclude and summarize our work.

II. TECHNOLOGY ENTROPY TREE

The TE-tree is a useful framework for describing the relationship between previous and successor technologies. The TE-Tree is composed of three components: previous technology, successor technology, and relation arrows. The basic scheme is shown in Figure 1.

Assume that technology A in Figure 1 is in its early stages. As A becomes better understood, applied, and diffused, A-related technology emerges: B and C. Figure 2 shows technology progression from the first to fourth generation. F is a programming language especially suited to numeric computation and scientific computing. J is an object-oriented language and system supporting platform independence. F and J in Figure 2 are different technologies from the third and fourth generation’s point of view. However F and J are cousins from a first generation perspective.

In Figure 2, technology will advance exponentially as time goes by. Each technology evolves into several new and cutting-edge technologies. The entropy of technology will also increase simultaneously. The entropy is proportional to the number of products and users. Due to constraints of time and resources, programmers and engineers have to concentrate on specific areas. They have to become specialists. Technologies, however, disappear into the background and new ones are emerging every day. The evolutionary rate of IT is by far the highest. There are staggering numbers of cases in which a spotlighted technology disappears quickly. In order to survive in an extremely dynamic technology environment, IT workers have to enhance their adaptive flexibility to accommodate technology change.

Figure 3 illustrates the transitive relation between technologies. State D cannot directly transit to state J, because Technology D and J are quite different. In order for State D to transit to state J, the state D should pass through state B. D is a technology that a programmer and an engineer are currently familiar with. J is a relatively emerging one that the IT industry has started adopting. B is a common ancestor of D, E, and J technologies.

For example, MFC is a useful programming framework for development of a Microsoft Windows application. Java is an object-oriented programming language to use for an application that will run on several platforms. They have evolved independently.

Both technologies, however, have a common ancestor, C++. MFC wraps the Windows API in C++ classes. The syntax of Java is largely derived from C++. An MFC programmer who has already studied C++ can understand and use Java technology without reading the first chapter of its manual.

Microsoft Windows, Linux and OS X are also quite different technologies. Not only end users but also programmers and engineers consider these operating systems completely different. If programmers who have programmed on a Windows platform should be involved in a Mac OS X project, it is quite a difficult challenge for them.

Nevertheless, these operating systems also have a common ancestor. The basic design of Linux is definitely derived from UNIX. OS X is directly based on BSD from UNIX. Windows also provides almost the same functions to operate kernels as UNIX does. Not only UNIX but also Windows provides API
based on C. Although Linux and OS X seem very different from Windows, acquirement of UNIX technology enables Windows programmers to gain familiarity with Linux and OS X.

The core idea of Figure 3 is that adaptive flexibility and survivability in ancestor layers is higher than in successor layers. The ancestor technology, however, does not make money in the here and now. Therefore, both ancestor and successor technologies are needed simultaneously. There are also great amounts of ancestor technologies in current IT. A common criterion is necessary to ensure a proper technology which programmers and engineers follow. We propose technology entropy as one criterion.

III. CALCULATION OF TECHNOLOGY ENTROPY

In TE-Tree, technology entropy is symbolized by $E$. $E$ is a state function measuring the amount of enjoyment, passion, interest, professional development, and influence for an input technology. $E$ is described as follows:

$$dE = k \cdot dA$$

(1)

where $dA$ is the amount of technological change. $k$ is a constant factor, mapping from a technology unit to an entropy unit. For constructing a TE-tree using technology entropy, two calculation phases are needed.

1) Phase I: Let $u$ be a technology, $E(u)$ is the entropy of $u$. $C_u$ is the set of successor technologies of $v$. The calculation associated with parent and child is simply:

$$E(u) = \sum_{v \in C_u} E(v)$$

(2)

There is a problem with a simplified equality. A child node that has multiple parents makes a greater contribution than it has. $D_v$ is the set of nodes that have only one parent, $v$. $G_v$ is the set of nodes which have multiple parents including $v$. $P_v$ is the set of nodes whose child is $v$.

$$C_u = D_u \cup G_u$$

(3)

$$E(u) = \sum_{v \in D_u} E(v) + \sum_{v \in G_u} \frac{E(v)}{|P_v|}$$

(4)

The process (4) is repeated recursively until the top. The final acquired value is total entropy $T$. Figure 4 is an example of upward calculation. In this case, every leaf node is set to be 1. Finally, the total entropy is 7.

In many cases, the phase 1 can give appropriate results. To include some exceptions, such as a disappeared technology problem, the calculation of entropy has two phases: upward phase and downward phase.

2) Phase 2: $A_t$ is the set of technologies in time $t$. When we define the total entropy, $T$ is simply:

$$T = k_t \sum_{v \in A_t} E(v)$$

(5)

$k_t$ is a normalization factor. We assume that $T$ is the same at any given time. Actually, $T$ in the past is lower than in the present. In the past, development tools were sparse and acquiring technology was more difficult. Additionally, education systems and curricula are not as well organized as in the present. There appears to be little to no difference in the difficulty of development and research between the past and the present. Therefore technology entropy normalization is a feasible assumption.

Assumption 1: The total entropy $T$ is the same at any time.

Normally, final total entropy includes all disappeared technologies from the present perspective. Therefore the total entropy at a higher layer has higher values than at a lower layer. Practically, if more than two independent trees exist or when assuming every technology in the area under consideration is affecting the current, there is the possibility that the total entropy at a higher layer is lower than at a lower one. The above cases violate Assumption 1.

Normalization with $T$ should be affected.

$$k_t = \frac{T}{\sum_{v \in A_t} E(v)}$$

(6)

The process (6) is repeated until the bottom. It normalizes total entropy from any time perspective. Figure 5 illustrates the final result of calculation.

Figure 4, 5 uses a static allocation of leaf factor with 1. A more accurate model can be acquired by dynamic allocation for leaf node value depending on market share, citation rate,
and countable influence. In this model, we extract key rules as follows.

**Rule 1:** At observing time \( t \), the probability that the higher entropy technology has more flexibility and survivability to technology change is higher.

Higher entropy technology sustains relatively enough entropy, which can be shared with successor technologies. When it shares its entropy with successors at time \( t_i (> t) \), the probability is higher that more entropy still remains in higher entropy technology than in lower at time \( t \).

**Rule 2:** As observing time \( t \) grows closer to the current time, predicting flexibility and survivability to technology change becomes more difficult.

Although the technology entropy at \( t \) is sizable, if the impact of successor technology at \( t_i (> t) \) is significant, the entropy of its previous technology at \( t \) decreases drastically. Therefore, not only entropy value at present but also the tendency of entropy change is needed. Intuitively, lower \( dE \) has higher adaptivity.

By Rule 1 and Rule 2, Rule 3 can be constructed.

**Rule 3:** Follow the technology that had higher technology entropy in the past and is simultaneously maintaining a relatively higher entropy at present.

IV. CASE STUDY: PROGRAMMING LANGUAGES

We set out to find an appropriate area to demonstrate the characteristics of technology evolution with TE-Tree. Finally we targeted programming languages. Programming languages have emerged, developed, matured, and disappeared continuously over five decades.

A. Methodology

We built a TE-tree with using two approaches, the "History of Programming Languages” diagram by O’Reilly, and the number of searching results per programming language in Google as a leaf node allocation.

There have been many programming languages since the computer era began. Our research considers the history from FOTRAN, 1954. We have built the TE-tree based on the O’Reilly poster called “History of Programming Languages” [15]. It plots over 50 programming languages in a multi-layered fashion, and there are over 190 versions of every language. The first version of the diagram was created by Éric Lévénez [16]. The O’Reilly version was built by historical knowledge and personal experience of the events in this poster among peers, authors, and editors.

We have found over 60 items as entropy values of leaf nodes. To avoid ambiguity and overlapping, we used quotation marks with ‘programming’ keywords like “Ada programming”. The number of search results in Google is regarded as a leaf node value. Using the above two methodologies, we built the TE-tree of programming languages.

In order to analyze a specific historical event, it is impossible to consider all the facets of its history. Likewise, it is impossible to include all technologies in this tree. Our approach, however, is sufficient to understand information technology evolution and technology entropy.

B. Total View

In this research, we assume that there is no disappeared technology in programming languages, based on the belief that someone somewhere remembers the technology and is affected by it, even though a programmer who uses it cannot be easily found.

According to our research, completely new technologies emerged in the two decades between 1954 and 1973. After that, the emergence of entirely new technology has been almost impossible. In this context, the phrase “completely new technology” is based on programming language layers. Figure 6 plots the normalization factor as a result of emerging new technology: LISP, APL, SNOBOL, BASIC, ISWIM, MUMPS, Forth, Prolog, and so on. \( k_{-1} \) is proportional to total entropy increase. It asymptotically follows an S-curve. (See also [4], [5], [6])

Our research extracted the result that the period from beginning to saturation of completely new technology is 20 years in the field of programming languages. The amount of entropy increased 43.25% from 1954 to 1973.

C. Early Computer Era

Figure 7 depicts the entropy value of programming languages in the early computer era: FORTRAN, C, ALGOL, Pascal and LISP. Most of the entropy of FOTRAN moved into ALGOL between 1954 and 1960, and over half of the entropy in ALGOL moved into C. From the current perspective, every programming language in this era, with the exception of C, has almost disappeared and exists only in books.

The entropy of each technology is proportional to the influence to the successor technology. Our research found that C is the ancestor technology that continues to affect its descendents the most strongly.

D. Object-Oriented Era

Figure 8 illustrates the entropy of programming languages in the object-oriented era. Java is the most commercially
Our approach provides insight for measuring and predicting the amount of flexibility and survivability of a specific technology. The advantage of our approach is intuitive and not difficult to understand. It provides IT workers, new firms, and established firms with criteria for choosing their technological path. It enhances the adaptivity to rapid technological change.

ACKNOWLEDGMENT

The authors are grateful for the helpful comments from W. Kim, W.J. Park, K.H. Han and the anonymous reviewers. We also would like to thank A.S. Hwang for inspiring us about the issues of technology and innovation.

REFERENCES


successful technology among incumbent OOP languages of that time. The Java project came from the Oak project at Sun. The Oak project was based on several previous technologies: Object C, Ada, C++, and Smalltalk, among others. Other successor technologies, however, except C++ have disappeared from the current perspective. The decreasing value of the entropy of C++ is mainly caused by movement to Java.

V. SUMMARY AND CONCLUSION

In this paper, we have reviewed the nature of technological change from an entropy perspective. We propose the TE-tree for modelling technology evolution. The proposed approach depicts relationships between previous and successor technologies. We extrapolate three rules from characteristics of the TE-tree. Especially Rule 3 provides a feasible strategy for long-term survivability in an extremely dynamic technological environment. In order to enhance these rules, we also suggest entropy as a criterion for measuring the value of technology. This paper additionally provides the calculation algorithm and shows a real example from the field of programming language.