Monitoring Software Technology Evolution, One Trend at a Time

Yanzhi Bai and Ali Mili

College of Computer Science
New Jersey Institute of Technology
Newark NJ 07102-1982
yb8@njit.edu, mili@cis.njit.edu

Abstract

The ability to model the evolution of software technology trends is valuable to many stakeholders in industry, academia, and government. Yet we often depend exclusively on the opinions of alleged experts to make predictions on trend evolution. In this paper we discuss an ongoing long term effort at bringing a measure of quantitative analysis to this discipline. We see our contributions as complementing expert opinions, not substituting for them.

Keywords: software evolution, trend, programming languages, operating systems, middleware, statistical analysis.

1. INTRODUCTION

The ability to understand the evolution of software technology trends carries great stakes for a broad variety of stakeholders, ranging from corporate decision makers to government officials to academic planners to technology consultants to mere individuals. Yet this ability remains rather elusive, as it eludes meaningful modelling. We recognize two broad families of approaches to this problem, which we can simplistically and summarily characterize as follows[1]:

- **Top Down/ Analytical Approaches,** which attempt to apprehend software trends by specializing general theories of scientific/ technical evolution to software technologies. These deductive approaches analyze epistemological theories pertaining to the evolution and dissemination of knowledge, distil their wisdom into a set of simple principles, which they then apply in the context of modern software technology.
- **Bottom Up/ Empirical Approaches,** which attempt to collect data on sample trends, then attempt to generalize sample observations to derive more general evolution laws. These inductive approaches collect empirical data on specific trends, derive specific evolutionary laws without pretending to explain them/ understand them, then attempt to synthesize these specific empirical laws into general, more broadly applicable laws.

[1] This distinction is simplistic, but we adopt it for the sake of argument.

2. BACKGROUND

In this paper, we briefly discuss our experience exploring the second approach. We have applied this approach to two families of software technology artefacts, namely programming languages and operating systems, and are currently applying it to a new family, namely middleware systems. Broadly speaking, our approach proceeds as follows:

- **Identify a Trend.** In this step, we identify a family of artefacts that present the following characteristics: the members of the family offer a unity of purpose and function; the members of the family can be viewed as competing for the same market, by virtue of delivering similar services; the members of the family can be characterized by a common set of attributes; the members of the family have a well documented evolutionary history over a number of years.
- **Intrinsic Characterization.** In this step, we introduce intrinsic attributes that characterize members of the selected family in a meaningful way, i.e. in a way that could conceivably affect their evolution. These attributes are selected according to the following characteristics: they must be orthogonal; they must be of general significance (apply equally to all members of the family); and they must be quantifiable.
- **Extrinsic Characterization.** A cursory analysis of software technology trends reveals easily that the fate of a trend has rather little to do with the intrinsic merits (as reflected by intrinsic factors). Hence we also introduce extrinsic factors, that reflect the environmental conditions in which the trend evolved, and the level of support that the trend has received from various quarters during its evolution. Generally, we consider institutional support (from academic institutions), governmental support (from governmental organizations), corporate support (from corporations), organizational support (from professional organizations and standards bodies), and grassroots supports (from practitioners at large). Hence we have a combination of planned/ centralized/ politicized support, and unplanned / distributed/ spontaneous support.
- **Quantification and Data Collection.** In this step, we quantify the intrinsic and extrinsic attributes, then collect data on them. Intrinsic attributes are typically trend-specific and time-independent; we quantify them and assess them using textbook references, or sometimes survey data. Extrinsic attributes are typically time-dependent; we quantify them and collect data about their quantitative values over a number of years, at specific time intervals (3 years or 5 years, depending on the length of their common history). We use historical records or, most typically, survey data, to collect data on extrinsic factors.
- **Empirical Laws: Trend Specific.** We analyze the empirical data we have collected in the previous step using statistical methods. In particular, we are interested to explore laws that capture the evolution of a trend (as reflected by its vector of extrinsic factors) as a function of its intrinsic attributes and its support history. To this effect, we perform

regression analysis and correlation analysis, using the current extrinsic factors as dependent variables and the past extrinsic factors and the intrinsic factors as dependent variables. Such models can then be used to predict the future of the trends in question: by feeding information about the past and present of extrinsic factors, we can now predict the future of extrinsic factors (in terms of institutional support, governmental support, organizational support, corporate support, grassroots support, etc), three or five years in advance, depending on the step of our data collection.

- **Empirical Laws: Attribute Specific.** All the predictions we can make may fall by the wayside if an event happens in the near future (within the period of our prediction) and alters the landscape pertaining to the trend of interest. To maintain the relevance of our findings against such discontinuities, we may focus, not on specific trends, but on trend attributes. To do so, we revise our statistical data, making our dependent attributes, not the level of support of individual trends, but rather trend characteristics. In other words, we do not quantify the level of support of a specific trend, but rather the level of support of a specific attribute, which we determine by analyzing to what extent the level of success of a trend is correlated to that trend featuring the selected attribute. Consequently, our prediction does not say: trend X will have such level of support 3 (5) years from now, but rather, attribute Y will be prominent, to a specified level in trends that are successful, to a specified level, in terms of some extrinsic factor. As a result, we do not characterize successful trends by name, but provide the profile of successful trends, regardless of their name, indeed regardless of whether these are existing trends or future trends.

We conducted two studies according to the pattern just described: one study dealing with programming languages [1]; and one study dealing with Operating Systems [2]. Currently, we are conducting a similar study on middleware systems. We will briefly present them below.

3. PROGRAMMING LANGUAGES

We have chosen programming languages as the object of this first experiment for a number of diverse reasons: First, they are important artefacts in the history of software engineering. Second, they represent a unity of purpose and general characteristics, across several decades of evolution. Third, they offer a wide diversity of features and a long historical context, thereby affording us precise analysis. Fourth, their history is relatively well documented, and their important characteristics relatively well understood. We have selected a set of 17 third generation languages as our sample, chosen for their diversity and their technical or historical interest: ADA, ALGOL, APL, BASIC, C, C++, COBOL, EIFFEL, FORTRAN, JAVA, LISP, ML, MODULA, PASCAL,

PROLOG, SCHEME, SMALLTALK. In order to model the evolution of these languages, we have resolved to represent each language by a set of factors, which we divide into two categories: intrinsic factors and extrinsic factors.

3.1. Intrinsic factors:

Intrinsic factors are the factors that can be used to describe the general design criteria of programming languages. We have identified eleven such factors [3][4]. We claim is that it is sufficiently rich to enable us to capture meaningful aspects of programming language evolution. Due to space constraints, we do not present detailed definitions of these factors, relying on the reader's understanding of these concepts, and referring the interested reader to [5]. These factors are: Generality, Orthogonality, Reliability, Maintainability, Efficiency, Simplicity, Machine Independence, Implementability, Extensibility, Expressiveness, and Influence/ Impact.

3.2. Extrinsic factors:

Whereas intrinsic factors reflect properties of the language itself, extrinsic factors characterize the historical context in which the language has emerged and evolved; these factors evolve with time, and will be represented by chronological *sequences* of values, rather than single values. We have identified six extrinsic factors for the purposes of our study: Institutional support; Industrial support; Governmental support; Organizational support; Grassroots support; Technology support. For example, the factor *grassroots support* reflects the amount of support that the language is getting from practitioners, regardless of institutional/ organizational/ governmental pressures. We decompose and define the other extrinsic factor in a similar manner, using quantitative questions.

3.3. Quantifying factors

Most of the intrinsic factors we have introduced above are factors for which we have a good intuitive understanding, but no accepted quantitative formula. In order to quantify these factors, we have chosen, for each, a set of discrete features that are usually associated with this factor. Then we rank these features from 1 (lowest) to N (highest), where N is the number of features. The score of a language is then derived as the sum of all the scores that correspond to the features it has. For example, to quantify generality, we consider ten features, ranging from *offering constant literals* (score: 1) to *offering generic ADT's* (score: 10). A detailed explanation of how all other intrinsic factors are computed is given in [5].

3.4 Model prototype

Before we present our summary statistical model, we consider the following premises: 1) We adopt intrinsic factors as independent variables of our model, as they influence the fate of a language but are themselves constant (time independent); 2) Because many extrinsic factors feed unto themselves and may influence others, we adopt *past values* of extrinsic factors as

independent variables; 3) We adopt (present or future values of) extrinsic factors as dependent variables of our model; 4) We do not represent the status of a language by the simple binary premise of *successful/ unsuccessful*, as this would be arbitrarily judgmental. Rather, we represent the status of a language by the vector of all its current extrinsic factors.

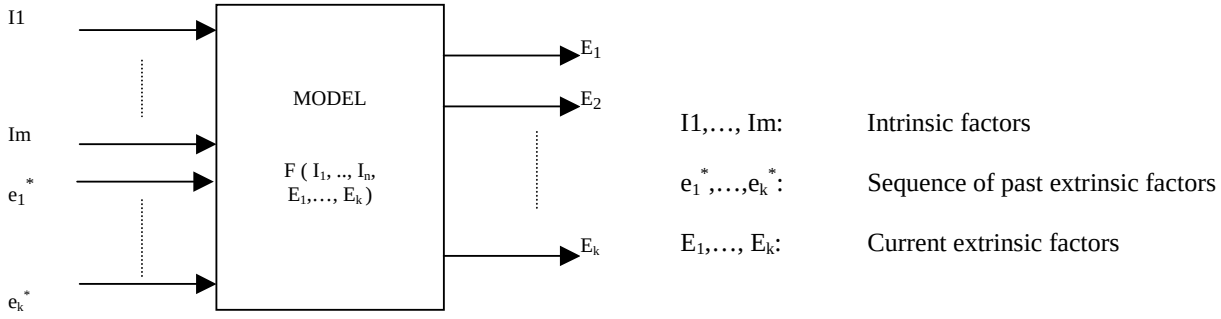

Figure 3.1 Statistical Model for Programming Language Trends

To show the evolutionary law of a language, we construct the following multivariate regression models by using the independent intrinsic and extrinsic factors. The multivariate regression equation has the form:

$$Y = A + B_1X_1 + B_2X_2 + ... + B_kX_k + E$$

where: Y is the predicted value on the dependent variable, A is the Y intercept, X is the various independent variables, B is the various coefficients for regression, and E is an error term. Overall, the independent variables of our model include the intrinsic factors and the past history of extrinsic factors, and the dependent variables include the current (or future) values of the extrinsic factors; see Figure 3.1. To evaluate intrinsic factors, we use the quantification procedures discussed in section 3.3. To this effect, we refer to the original language manual and determine whether each relevant feature is or is not offered by the language. To collect information about extrinsic factors, we have set up a web-based survey. We have publicized our survey very widely through professional channels (for example, google, yahoo, and other computer professional newsgroups) to maximize participation. The information we request from participants pertains to their knowledge/familiarity/practice of relevant languages, as well as levels of organizational commitment to specific languages.

3.5. Main Empirical Conclusion

In this project, factor analysis is used to investigate the latent factors in intrinsic and extrinsic factor groups. Canonical analysis is used as an advanced stage of factor analysis. According to the data we collected, the 5 most popular languages (most people consider them as their primary programming languages) in 1993 are: C (22.47%), PASCAL (17.81%), BASIC (16.19%), FORTRAN (9.51%), C++ (6.88%). The 5 most popular languages in 1998 are: C (22.03%), C++ (18.31%), SMALLTALK (8.64%), FORTRAN (8.47%), PASCAL (7.79%). The 5 most popular languages in 2003 are: C++ (19.12%), JAVA (16.26%),

SMALLTALK (13.32%), ADA (10.38%), FORTRAN (9.34%). Figure 3.2 shows the trends of most popular programming languages from 1993 to 2003. This figure presents a sample factor for grassroots support.

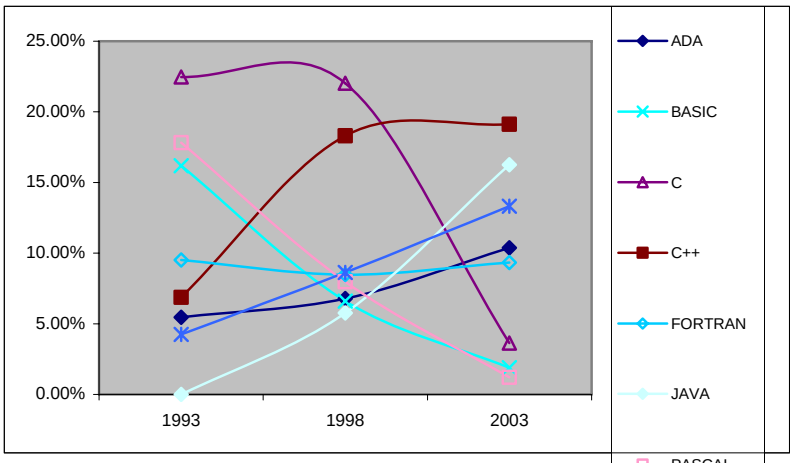

Figure 3.2 Trends of "How many people consider this language as their primary programming language" from 1993 to 2006

The analysis also shows the meaningful relationships between the intrinsic factors of a language and the value of its dependent variables. As an example, we consider the impact of intrinsic factors on the number of developers who consider the language as their primary development language. The results are summarized in Table 3.1. It shows that machine independence, extensibility and generality have the greatest impact on this extrinsic factor. By analyzing the tables for all factors, we find that the most important intrinsic factors are generality, reliability, machine independence, and extensibility.

How many developers consider this language as primary language?			
Generality	0.6913	Implementability	-0.3390
Orthoganality	0.0199	Machine Independence	0.8876
Reliability	0.3199	Extensibility	0.7625
Maintainability	0.0470	Expressiveness	0.3024
Efficiency	0.0703	Influence/Impact	0.0552
Simplicity	-0.4703		

Table 3.1 Sample Correlation Results for Intrinsic Factors Only

3.6. Validation

We construct this derivative model by using 12 languages and will use 5 languages to validate it. We consider the extrinsic factor of "*What percentage of people know this programming language in 2003"* and compare the actual value collected from our survey against the predicted value produced by our regression model. The results are shown in Table 3.2. F-Statistic, which is a standard

statistical method to check if there are significant differences between 2 groups, is used to validate the prediction. In the F-table, for a=0.05, F must be greater than 4.49 to reject the hypothesis of statistical correlation. Because our F value is 0.235, which is much less, the hypothesis is validated. Same here:

Languages	Actual Value	Predictive Value
ADA	5.19%	6.94%
EIFFEL	5.90%	7.16%
LISP	7.68%	7.74%
PASCAL	54.29%	48.81%
SMALLTALK	10.06%	8.48%

Table 3.2 Difference between Actual & Predictive Value

3.7 Predictive Model

We build a regressive model of our data by instantiating the equation of section 3.4 for the factors that are relevant to our study. Specifically, we posit a linear equation that formulates the extrinsic factors as of 2003 as a function of the intrinsic factors and the extrinsic factors for the years 1998 and 1993, and we let the regression methods derive the linear coefficients and error term. We find:

$$E_{2003} = A * I + B * E_{1998} + C * E_{1993} + D$$

where: E_{2003} is the vector of extrinsic factors in 2003 , I is Value of intrinsic factors, A is the Parameter matrix for intrinsic factors, E_{1998} is vector of extrinsic factors in 1998; B is the parameter matrix for extrinsic factors in 1998, E_{1993} is the vector of extrinsic factors in 1993, C is the Parameter matrix for extrinsic factors in 1993, and D is a Constant value. We generalize this model to obtain a predictive model, by letting the year be a parameter, based on the assumption that the evolution of extrinsic factors will continue to follow the same linear law. To predict the value of each extrinsic factor in 2008, use data from 1998 and 2003 in lieu of 1993 and 1998 in the equation above. The results are shown in Figure 3.3.

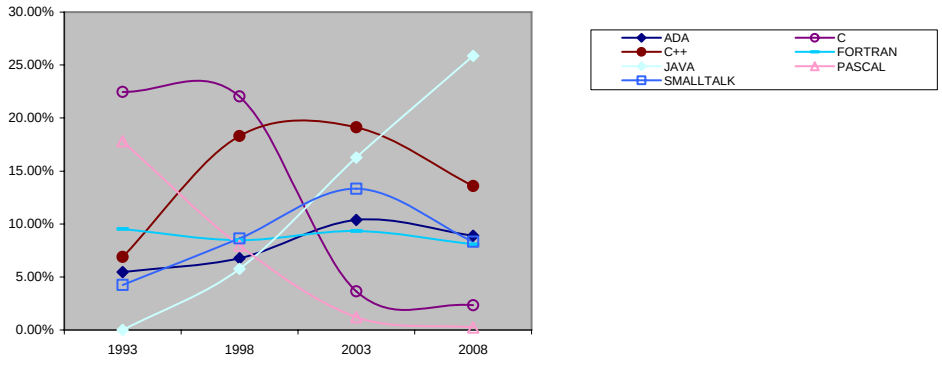

Figure 3.3 Trends of most popular languages from 1993 to 2008.

4. OPERATING SYSTEM TREND ANALYSIS

4.1. Experimental Setup

We conducted a similar study of the evolution of operating systems. The operating systems we have chosen to this effect are: Unix, Solaris, Sun/OS, BSD, Windows, MS-DOS, MAC OS, Linux, Netware, HP_UX, GNU Hurd, IBM AIX, Compaq/ DEC VMS, OS/2. We have used the same extrinsic (environmental) factors as for programming languages, but we have, of course, chosen different, OS-specific, intrinsic factors. However, in preparation for a possible merger with results from other trends, we have divided the set of factors into seven broad classes, which we briefly introduce below:

- ***Resource Management Factors.*** This category includes *scalability* (ability to make use of added hardware); *CPU management; memory management; I/O Management.*
- ***Usability.*** This category includes ease *of learning; ease of use; and consistency of interaction protocols.*
- ***Usefulness: Functional.*** Whereas usability reflects ease of use/ congeniality to the user, usefulness reflects the level of service provided to the user. Functional usefulness includes: *Range of services; Range of programming language support; distributed computing; network services; and deadlock management.*
- ***Usefulness: Operational***. Operational attributes of usefulness include *reliability*, and *security and protection*.
- ***Versatility.*** Versatility of an operating system is its ability to run on a wide range of platforms, under a wide range of distinct operating conditions. We have identified three dimensions of versatility: *Portability*, *Compatibility;* and *Openness.*
- ***Design.*** This factor reflects design qualities of the operating system, such as *integrity*, *economy of concept*, *orthogonality*, and *adherence to design principles*.
- ***Cost.*** This factor considers *acquisition costs*, *maintenance costs,* and *operating costs* of an operating.

Due to data collection constraints and shorter time spans where accurate data could be collected, we have resolved to let the step of our time-dependent data be 3 years rather than 5 years (as we had for programming languages). Also, as we discussed earlier, in the study of operating systems we went beyond modelling the evolution of individual operating systems, to model the evolution of operating systems attributes, which we represent by the intrinsic factors. In other words, we are not content with identifying successful operating systems; rather we also want to identify the attributes that successful operating systems have in common. This allows us to make predictions that are immune to discontinuities in the evolution of trends: while we may not know what

operating system will be most successful three years from now (by whatever extrinsic metric we choose), we may know what features will characterize a successful operating system.

4.2. Experimental Results

The factor analysis shows that six extracted components are sufficient to account for more than 95.903% of the variance of the dependent variables (intrinsic factors and past extrinsic factors). From the factor analysis, we find that intrinsic variables do represent important features of the operating system. The multivariate regression model is a useful approach to predict evolving trends based on existing data. We use canonical correlation (specifically, Pearson's correlation) as an additional procedure for assessing the relationships between dependent variables and independent variables. Different intrinsic factors of an operating system do have different impacts on the overall performance by using this model. Table 4.1 shows, as an example, the results of correlation analysis of government support and intrinsic factors. Figures 4.1 and 4.2 show the evolution of operating systems from 1997 to 2006 with respect to organizational support and grassroots support.

Intrinsic Factor	Governmental Support	Intrinsic Factor	Governmental Support
Security Protection	0.883	Compatibility	0.589
Scalability	0.789	Consistency of Interaction Protocol	0.578
Design	0.750	Ease of Learning	0.553
Network Service	0.747	Reliability	0.455
Deadlock	0.709	Openness	0.426
IO	0.666	Ease of Use	0.425
CPU	0.643	Distributed Computing	0.408
Range Of Programming Languages	0.612	System Services	0.291
Memory	0.589		

Table 4.1: Correlation Analysis

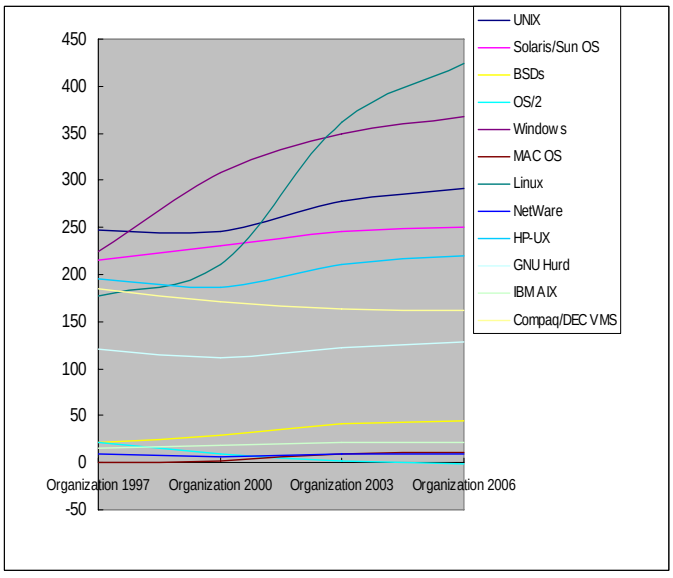

Figure 4.1 Evolution of Organizational Support from 1997 to 2006

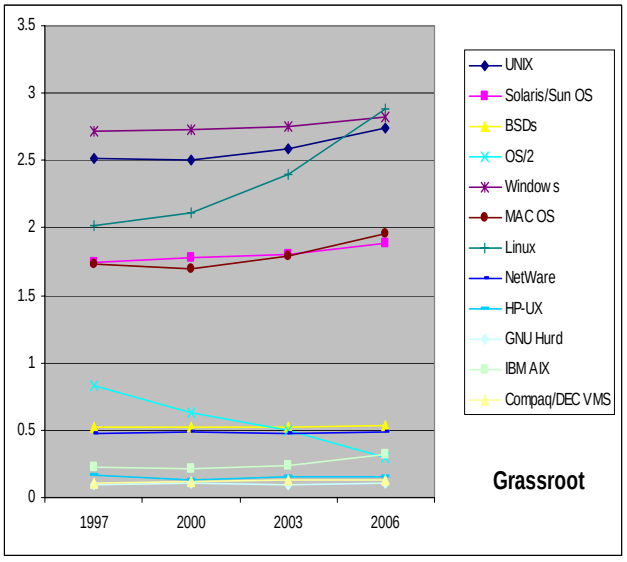

Figure 4.2 Trends of grassroots support from 1997 to 2006.

4.3. Statistical Validation

The method that we have used for statistical validation is empirical, and it consists of applying the predictive model on past data, to predict current (as of 2006) data, then comparing the data produced by the predictive model against actual data. Table 4.2 provides this data for the extrinsic factor of government support.

OS	Government Support		OS	Government Support	
	Actual Data	Predicted Data		Actual Data	Predicted Data
UNIX	2.575	2.532	Linux	2.735	2.747
Solaris	2.061	2.013	NetWare	1.052	1.020
BSDs	1.561	1.449	HP-UX	2.262	2.204
OS/2	0.515	0.483	GNU Hurd	0.492	0.432
Windows	2.655	2.674	IBM AIX	2.316	2.268
MAC OS	0.343	0.095	Compaq DEC VMS	1.493	1.426

Table 4.2: Comparison of Actual Value and Predictive Value For Government Support

4.4. Predictive Model for Features of Operating System

Space limitations preclude us from giving a detailed account of how we charted the evolution of operating systems features; the interested reader is referred to [6]. The overall idea is to represent operating systems attributes by intrinsic factors and to represent measures of success by extrinsic factors. Then we quantify the preponderance of each particular attribute (for example: *protection and security*) in successful (for example: with respect to grassroots support) operating systems by the correlation between the two events: "an operating system has that attribute" and "an operating system is successful". By charting the evolution of this correlation from year to year, we get a sense of the importance of the selected attribute for the selected measure of success (in this example: importance of *protection and security* for *grassroots support*); by extrapolating into the future, we get a sense of the profile (represented by a vector of attributes) of successful operating systems of the future.

Our investigation of operating systems attributes provide that the most important attributes that characterize operating systems of the future, ranked by order of decreasing importance, are as follows: Range of services (functional usefulness); Ease of Learning (usability); Ease of use (usability); Openness (family of versatility); and Reliability (family of operational usefulness).

5. MIDDLEWARE

In the computer industry, *middleware* is a general term for any programming that serves to "glue together" or mediate between two separate and often already existing programs. A common application of middleware is to allow programs written for access to a particular database to access other databases. Typically, middleware programs provide messaging services so that different applications can communicate. The systematic tying together of disparate applications, often through the use of middleware, is known as enterprise application integration (EAI).

5.1. Selecting Middleware Systems

We have selected object middleware according to their chronology, diversity and technical interests. Below we list some middleware protocols and products

1, ODBC/JDBC: connecting programming languages (Windows based languages and Java) to databases.

2, JMS: java message service integrates with existing messaging systems.

3, JavaBean is software component written in the Java programming language that allows programmers build re-useable applications blocks called components that can be deployed in a network on any major operating system platform.

4, Enterprise Java Bean (EJB) is a managed, server-sided component for modular construction of enterprise applications.

5, J2EE is an industrial standard/product initiated by Sun Microsystems. It specifies a programming platform for developing and running distributed multi-tier architecture Java applications, based largely on modular software components running on an application server.

6, MSMQ is essentially a messaging protocol from Microsoft that allows applications running on disparate servers to communicate in a failsafe manner.

7, MQSeries is an IBM software family whose components are used to tie together other software applications so that they can work together.

8, COM/COM+/DCOM, which are promoted by Microsoft, provide heterogeneity across languages only on Windows operating system.

9, MTS: Microsoft Transaction Server is a program that runs on an Internet or other network server and manages application and database transaction requests on behalf of a client computer user.

10, MS dot NET, a Framework of software component which can be added to the Microsoft Windows operating system, providing a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

11, CORBA (Common Object Request Broker Architecture) is published by Object Management Group. It allows remote method invocations on objects. CORBA offers heterogeneity across programming language and vendor implementations.

12, Jini is an architecture for distributed computing, especially unreliable mobile computing.

13, Jboss, referred to JBoss Enterprise Middleware Suite (JEMS) is an extensible and scalable suite of products for creating and deploying e-business applications. Jboss Middleware simplifies J2EE.

14, The BEA WebLogic Platform is the application platform suite for developers service-enabling their applications. It enables enterprises to achieve faster time-to-value for critical business applications using open standards, web services and a Service-Oriented Architecture (SOA).

15, IBM WebSphere as a brand refers to a group of IBM software products. From a technical perspective, WebSphere typically means the WebSphere Application Server (WAS). WAS provides a bunch of services-- J2EE that Java applications use such as database access, mail services and security services. It is designed to set up, operate and integrate e-business applications across multiple computing platforms using Web technologies. It includes both the run-time components (like WAS) and the tools to develop applications that will run on WAS.

16, Apache Geronimo is to produce a server runtime framework that pulls together the best Open Source alternatives to create runtimes that meet the needs of developers and system administrators.

17, Oracle Fusion Middleware—a family of products ranging from application development tools and integration solutions to identity management, collaboration, and business intelligence reporting—provides the critical software foundation for business growth and control.

We can see this middleware follow the rule of chronology. Some of them are out-dated; some are the mainstream middleware platform nowadays; some are still being developed and changing.

This middleware also varies in terms of the programming abstractions they provide and the kinds of heterogeneity they provide beyond network and hardware. They are divided into the following categories:

- Distributed database Access: ODBC/JDBC

- Distributed Computing Environment (Remote procedure call): RMI, (Jini)
- Message oriented middleware: JMS, MSMQ, MQSeries,
- Distributed object middleware: COM/DCOM/COM+, JavaBean, Enterprise JavaBean, Corba
- Transaction process monitor: MTS
- Enterprise Service Bus: Dot Net and J2EE
- Application Server (Java/.Net): JBoss, WebLogic, WebSphere, Fusion, Apache Geronimo

Although it is said that the approach of layering middleware was problematic and unproductive, such an approach is useful in explaining their genealogy. The domain and hierarchy of these categories are listed below. The lower hierarchies provide services to the higher hierarchy, or they are contained in the services of the higher hierarchies as further development. By this way, we can ensure that the middleware family is well represented by the middleware we selected.

5.2. Selecting the attributes to analyze

Intrinsic Factors

Factor Names	Sub-attributes
Functionality	Breadth of applicability; tools supporting development and management
Generality	• Breadth of applicability • Tools supporting the development and management. • OS supported • Languages Supported: • Standard Support: • Support for existing applications: does it provide an efficient way to preserve or reuse legacy application systems? • Interoperability: between different languages and vendors. • Scalability: how extendible and runable on platforms from low end to high end.
Usability	Ease of learning; Ease of use
Cost	Acquisition Cost; Operation Cost
Operational Quality	• Availability • Security and Protection • Maintenance and management: how easy to manage, include service monitoring and administration. • Running Performance:

Extrinsic Factors

We choose the same extrinsic factors as other studies, namely: institutional support, corporate support, governmental support, , organizational support and grassroots support.

5.4. Quantifying attributes

Some of the attributes can be quantified easily because they are numeric themselves, such as maintenance cost. Some of the attributes are rated on a 1-5 scale either from online survey or analysis. The attribute can also be expressed as a composite of a list of sub-attributes, which are in turn quantified from above methods.

We are currently in the early stages of this project, whereby we are setting up the online infrastructure at http://swtech.njit.edu/ to collect empirical data. Meanwhile, to build a new model, we are exploring the data mining methods rather than multi-linear regression analysis. New methods do not assume the functional form is known, and the algorithm discovers it. The function forms may be non-linear or arbitrarily complex.

The online survey is currently being set up at the URL given above. We anticipate that respondents will be solicited to assess some of the intrinsic factors, as well as the extrinsic factors. Some intrinsic factors, such as ease of use, ease of learning, could conceivably be assessed offline by analyzing their contributing attributes and quantifying them; we envision to still solicit user feedback, and decide subsequently how to integrate these heterogeneous metrics. As for extrinsic factors, we envision to dispatch respondents to different forms depending on two factors: first, which type of organization they belong to (academic institution, governmental institution, industry, etc) and second whether they respond on behalf of their institution (accounting for their specific institutional support) or as individual users (accounting for grassroots support). The same person can fill out more than one form, under different roles.

6. PROSPECTS

We envision to build on our individual empirical studies to produce more general laws of technology evolution. One possible venue we are considering is to proceed as follows:

- **Merge data about extrinsic factors.** All our empirical studies have the same set of extrinsic factors: institutional support, governmental support, corporate support, organizational support, grassroots support. Consequently, we can merge the data collected on all three trends (programming languages, operating systems, middleware systems).
- **Classify then merge data about intrinsic factors.** The extrinsic factors that characterize a family of trends clearly depend on the family; hence we could not merge trend specific intrinsic factors directly. We envision to classify it first, into broad categories such as: usability; usefulness; user-friendliness; level of services provided; portability;

effectiveness; cost; etc. Once data from each trend is classified in a way that is no longer trend specific, then we can merge the data from all trends into one set.

- **Performs Generic Statistical Analysis.** The purpose of this step is to revisit the statistical analyses we had conducted for each trend and carry them out across trends, using the generic classification of intrinsic factors.
- **Derive, Analyze and Validate General Laws.** We can deploy statistical techniques to the combined data, producing analysis tools and predictive tools, which we can then validate empirically by seeing how faithfully they model existing studies (programming languages, operating systems, middleware systems) then how faithfully they model new trend data (dealing, for example, with database packages, web browsers, etc).

This, we believe, is a viable alternative to the highly speculative methods of top down analysis, and provides complementary perspectives.

Bibliography

[1] Yaofei Chen, Rose Dios, Ali Mili, Lan Wu, Kefei Wang: An Empirical Study of Programming Language Trends. IEEE Software 22(3): 72-78 (2005).

[2] Yi Peng, Ali Mili, Fu LiMin. Modeling the evolution of operating systems: An empirical study. Journal of Systems and Software. January 2007.

[3] Kenneth C. Louden. Programming Language Principles and Practice. PWS Publishing Company, Boston, MA. 1993.

[4] U.S. Department of Defense. June 1978. “Department of Defense Requirements for High Order Computer Programming Languages: “Steelman” ”

[5] Yaofei Chen, PhD thesis: Programming language trends : an empirical study, 2003, New Jersey Institute of Technology

[6] Yi Feng, PhD thesis: Characterizing the evolution of operating systems. 2005, New Jersey Institute of Technology

[7] Nikos Passas, Sarantis Paskalis1, Enabling technologies for the ‘always best connected’ concept, WIRELESS COMMUNICATIONS AND MOBILE COMPUTING, 2005; 5:175–191

[8] Andrew S.Tanenbaum. Modern Operating System. Second Edition ed. Upper Saddle river, New Jersey: Prentice Hall, 2001.

[9] David G.Kleinbaum, Lawrence L.Kupper, Keith E.Muller, Azhar Nizam. Applied Regression Analysis and Multivariable Methods. 3rd edition ed. Duxbury Press, 1997.

[10] Middleware product specifications from Oracle, IBM, Microsoft.