A Novel Dynamic Risk Assessment for Communication Network System Software

Authors:
Bo Li*

Center for Advanced Computing and Communication
North Carolina State University
Raleigh, NC 27695,USA
Tel: (919) 515-2051
Fax: (919) 515-5108
Email: bli@eos.ncsu.edu

Mo-Yuen Chow

Center for Advanced Computing and Communication
North Carolina State University
Raleigh, NC 27695,USA
Tel: (919) 515-7360
Fax: (919) 515-5108

Arne Nilsson

Center for Advanced Computing and Communication
North Carolina State University
Raleigh, NC 27695,USA
Tel: (919) 515-5130

Wendell Jones and Gregory Kaszycki

Metrics, Tools and Analysis
Nortel Networks
RTP, NC 27709,USA

*: Corresponding author
For General Conference
Key Topic: Communication Software

A Novel Dynamic Risk Assessment for Communication Network System Software

Bo Li Mo-Yuen Chow Arne Nilsson Wendell Jones Gregory Kaszycki

bli@eso.ncsu.edu chow@eos.ncsu.edu nilsson@ncsu.edu
IEEE Student Member IEEE Senior Member IEEE Senior Member IEEE Member

Center for Advanced Computing and Communication
North Carolina State University
Raleigh, NC 27695,USA

Metrics, Tools and Analysis
Nortel Networks
Research Triangle Park, NC 27709, USA

Abstract

Communication network systems' software can easily contain millions of lines of code, and it is almost unavoidable to introduce bugs/faults during its design and development process. These faults can in turn affect the performance of the communication systems. Thus, communication system reliability is one of the major concerns in today's communication industry. At Nortel Networks, a static reliability measure called OPRISK, used in the Emerald package that estimates the reliability of communication software, has been developed to evaluate the reliability of each individual software module. Since software modules rarely operate alone (they invoke and interact with other modules during operation), this paper extends the software reliability measure OPRISK to a dynamic measure DNRISK to include the interactions among software modules and to estimate the corresponding reliability. Field data test results show that the DNRISK measure can provide a much better indicator for type II prediction error for interactive software modules and can be a reliable indicator to assist software engineers to effectively improve their products.

Keywords*: Communication network system performance, Software reliability, Static risk measure-OPRISK, Dynamic risk measure-DNRISK*

1. Introduction

With the advancement of communication and networking technology, modern society is becoming more and more dependent on software-intensive products and systems. Poor software reliability can threaten both equipment and personal safety, risk a company's business, and alienate potential customers. Ignoring software reliability during software development stage leads to poor system performance and eventually higher costs [1-9]. Traditionally system reliability is measured in terms of time between failures [10]. A software system failure can be caused by a fault or defect in the executable software product [1-4, 9]. One of the communication network software reliability measure is its probability of failure-free operation in a given environment. One may want to know the number and location of faults in each module in order to guide the development and modification. However, predicting the number and location of faults may not be achievable based on available information. Yet, we can identify and rate the most troublesome modules for likelihood of failure, which can be used as a valuable indicator to assist software engineers to focus their efforts to upgrade their software.

The inputs of a software risk rate model, are usually product metrics such as the static attributes of the source code and early-development process metrics [8]. The output of the model is a risk rating indicates whether a module is fault-prone. This type of risk rate model is usually used to evaluate individual modules and is called *static module risk rate*. Emerald (Enhanced Measurement for Early Risk Assessment of Latent Defects) which is jointly developed by BNR, Nortel Networks and Bell Canada is a decision support system designed to improve communications software reliability [4, 5, 7, 11]. Given a software module, Emerald can provide different static risk rate such as [4, 11]:

AIM - metric with a composite complexity score.

TTRISK - metric with a composite complexity index made up of AIM and other factors.

OPRISK - metric for field failure risk.

Emerald not only can be used by software engineers to improve software reliability, but also can facilitate the possible correction of problems. However, Emerald at present does not evaluate the reliability contribution of a module to other modules or to the whole system. How does the modification of a module affect the risk rate to other modules or to the whole system during the execution of a software is a new and challenging problem for communication network software engineers.

In this paper, a novel communication network software dynamic risk (DNRISK) evaluation with considering the interrelationship of modules in the calling trace from the risk rate of each individual module is presented. This approach can be applied to evaluate the dynamic risk of a software system or a module at different calling location in the software system. Dynamic risk rate can help the system analyzer to locate the high risk critical calling pattern, narrow down the faulted modules when there is a software system failure. The study of existing Emerald tool and the calling trace file render the insight into which uses the static risk rate and the calling relation in a calling trace file to develop the DNRISK for modules and system in this trace file. Field data validation demonstrate that the DNRISK can be used as an effective indicator for software engineers to reduce type II error for communication network software fault prediction.

Corresponding Author: Bo Li

2. Emerald risk models

This section briefly describes the Emerald software, of which DNRISK will base on. Emerald, developed jointly by BNR, Nortel Networks and Bell Canada, is a tool that can apply different risk rate assessment strategies at the earliest possible stages of software development. Emerald provides decision support by assessing risk rate and predicting software faults based on several source code characteristics. The goal of Emerald is to predict which module will be most probably to receive modification and update due to changes or "fixes" resulting from field Customer Service Reports (CSRs) by assigning a risk rate to each module. Emerald can evaluate risk rate with different metrics such as the AIM, TTRISK and OPRISK mentioned previously.

In Emerald, the basic inputs into the risk rate models are static code module complexity metrics (derived from procedural-based metrics and is showed in Table 1). Other more dynamic aspects of the code including the usage (actually, the percentage of North American sites at which the module is deployed) and the number of problems found during development that were used to change the module.

Table 1. Code metrics

Metric	Definition	Metric	Definition
C	Number of logical Comments	NL	Number of Loop constructs
DS	Number of declarative Statements	NP	Number of Independent Paths
E	Number of arcs	TCT	Total calls to others
NC	Mean conditional arc Complexity	V	Number of summits
NCN	Number of conditional arcs	VCD	Comments volume in Declarations
NDI	Number of distinct include files	VCS	Comments volume in Structure
NELMAX	Maximal nesting level	VS	Structure volume
NELW	Weighted mean nesting level	OUT-RANGE	Number of violations of thresholds for each DATRIX metric for all Procedures

The Emerald tool has evolved over the last several years. Initially, some of the metrics in Table 1 were "combined" into a single complexity metric called AIM which was found to be correlated with software defects and was one of the first metrics used to predict defective modules. Later, TTRISK was developed as a metric that was better at predicting CSRs and patches than the original AIM metric. TTRISK takes into account additional code metrics (such as NELMAX, the maximal nesting level, and NP, the number of paths). Finally, TTRISK and additional metrics which indicate the environment in which the code is executed are combined through a logistic regression model to yield a new metric for risk rate called OPRISK. These environmental metrics include a rough measure of the usage of the module (based upon the number of sites in which the module is deployed) and the two other aspects involve the number of problems found during development. TTRISK, Usage, Internal PRSs (number of problems found by designer), and VO_PRSs (number of beta-test problems found in this module during beta testing of the latest release) were found to play a significant role in predicting whether the module would have an update due to a field CSR. The relationship and evolution of the different metrics are shown in Figure 1 [11].

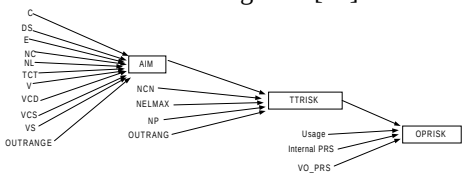

Figure 1. The relationship and evolution of different metrics in Emerald.

3. Analysis of calling trace files

Different tests on communication system software generate different calling trace files, which can be used to keep track of the test cases, an example of the calling trace file is shown in Table 2.

Table 2. An example of the calling trace file.

Ta	Tt	Td	Tc	P
Actual Time	Total time	Time Since last event	Time in this proc	Procedure
0:48.256	1	0.58	3	040DEC90 MISCMACH STOP_ERA
0:48.256	12	6.05	84	059886A0 TRKSTR TRUNK_STARTER
0:48.256	23	10.66	71	. 056E3780 EDTKCPUI SETUP_TO
0:48.256	24	1.08	68	. . 056E3461 EDTKCUPUT EVENT_DR
0:48.256	26	1.86	1	. . . 056E3964 EDTKCPUI DEFAULT_
0:48.256	37	1.00	1	. . . 056E3964 EDTKCPUI DEFAULT_
0:48.256	38	0.33	5	. . . 04F56004 AINTDPUI REAL_TDP
0:48.256	47	4.01	37	. . . 058083C0 TRKITHDL TRKM_IT_F
0:48.256	82	0.90	2	. . . . 058B0464 EDTKUTIL ST_STAT
0:48.256	86	0.96	5	. . . 04F56004 AINTDPUI REAL_TDP
0:48.256	98	2.35	-	042A44 CMCALLP CM_RESET
0:48.256	101	3.23	3	. 040C0A48 INTSYS READ_UPD
0:48.256	105	0.95	3	. 040DE350 MISCMACH SUBCPUAC

The first four columns track several timing information corresponding to the procedure listed in the fifth column.

1.The first column lists the actual time when the procedure listed in the fifth column, is called. This time is from a real time clock and has a precision of milliseconds.

2.The second column shows cumulative time since the start of the first procedure call. This is the actual CPU-time and the unit of measurement is micro-second.

3.This column tracks the time passed for last event.

4.The fourth column lists the time spent on executing the current procedure. The time spent in the current procedure includes the time for all subsequent procedures called by the current procedure. In other words, this time for top level procedure takes care of all the nested procedure calls under it.

5.The fifth column lists the procedure calls along with the module name, a module ID and calling relation of this procedure with other modules.

Notice that the fifth column of the trace file has the following format – zero or more numbers of dots, followed by the module-ID, the module name and finally the procedure name. For example:

```
.045D2AE0 TRK GET_TRUNK_SUBGROUP_DATA
```

There is a single dot, the module-ID is 045D2AE0, the module name is TRK and the procedure name is GET _ TRUNK _ SUBGROUP_DATA. The number of dots in a line indicates its relationship of modules. To explain this let us consider two consecutive rows of data from the fifth column.

Corresponding Author: Bo Li

i) If both rows have the same number of dots, then the procedure calls corresponding to both rows are serially executed. In other words, the subsequent procedure is executed only after the earlier procedure has finished.

ii)If the subsequent row has one more dot than the previous one, then it means a nested call. In other words, the previous procedure calls the subsequent one. For example,

```
. 045D2AE0 TRK GET_TRUNK_SUBGROUP_DATA
  . . 056E3464 EDTKCPUI EVENT_DR
```

where the procedure GET_ TRUNK_ SUB-GROUP_DATA of module TRKDUI calls the procedure EVENT_DR of the module EDTKCPUI.

If the subsequent row has one or more dot lesser than the earlier one, then it means one or more nested calls has ended. The number of dots lesser than the earlier one is equal to the number of the nested procedure calls that have been executed.

A calling tree is constructed with a tree root. The trace file is then scanned in order to construct different branches of the calling tree. While Emerald concentrates on individual module risk rate, DNRISK evaluates risk rate by including the interaction among modules. One example of the calling tree, which is constructed from part of an actual calling trace file, is depicted in Figure 2. The nodes of the tree represent modules and the tree branches represent the calling relationship between modules. The root represents the caller of the trace files. The levels in the tree can be used to find out the order of relationship. Same modules can appear more than once in the tree.

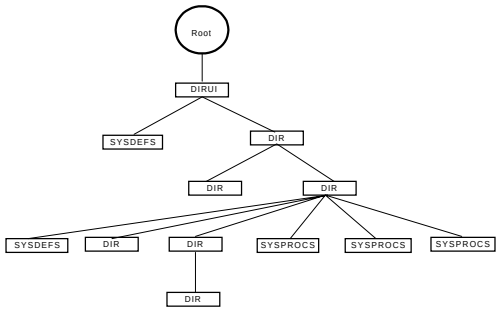

Figure 2. An example of a calling tree.

Some trace files have more than thousand modules involved and the structure of the corresponding tree can be very complex. Nevertheless, all the relations of modules can in general be classified into two categories: a vertical relation and a parallel relation. The vertical relation between two modules is shown in Figure 3 where module B is a leaf, and there is no module called by module B. The parallel relation of modules is shown in Figure 4, where modules B, C and D have the same relation with module A and they are parallel to each other.

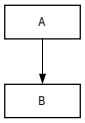

Figure 3. A vertical relationship of two modules.

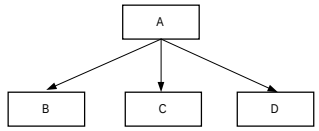

Figure 4. A horizontal relationship of modules.

4. DNRISK calculation of a calling tree

This section illustrates how to compute the DNRISK for both the vertical and parallel relations from which the DNRISK for the whole calling system and subsystems can be computed. In Figure 3, module B is the leaf, and its only relation is called by module A. Thus, its static risk rate derived from Emerald is the same as its DNRISK in this relation. With the calling of module B, the DNRISK for module A may be different from its static risk rate. If we treat the vertical relation as a subsystem, then the DNRISK of this subsystem will be the DNRISK of module A.

Conventional reliability approach will take multiplication of the static risk rate of A and B as the new risk rate for the system by assuming these two components are independent. The assumption of independence is generally invalid in communication network software system with calling and called relationship. A modified metrics is introduced here to calculate the DNRISK of the vertical relation. This modified metrics approach takes the inter-relationship of the source code and the calling relation into consideration. Notice that the system DNRISK for module A calling module B and module B calling module A may be different.

In order to derive the DNRISK for vertical relation, a virtual module T is constructed to represent the "combined" module of A and B during calling time, as shown in Figure 5. The evaluation of DNRISK for the vertical relation of module A and B becomes the prediction of the static risk rate for module T.

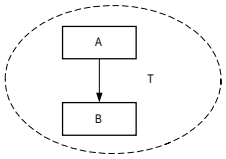

Figure 5. The virtual module T for the vertical relation.

Figure 6 shows the relation of risk rate OPRISK with module source and other environmental related metrics. OPRISK can be derived from TTRISK, Usage, Internal PRS and VO_PRS. To evaluate DNRISK of module T, all these metrics need to be modified to represent the calling relation of module A and B.

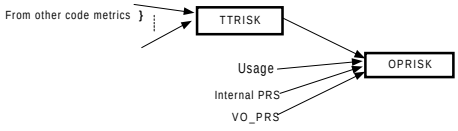

Figure 6. The relationship between code metrics, TTISK and OPRISK.

In Figure 6, TTRISK is solely determined by source code metrics. Thus a weighted average of TTRISK for module A and B is used for T. One approach to choose the weighting of A and B is to use the sizes of the

Corresponding Author: Bo Li

corresponding modules. Let S_A, S_B be the code size of module A and module B respectively, then the modified TTRISK for module T can be calculated as:

$$TTRISK_T = \frac{S_A \cdot TTRISK_A + S_B \cdot TTRISK_B}{S_A + S_B}. \quad (1)$$

The environmental metrics Internal PRS and VO_PRS are the numbers of problems found in a module. These values can be transferred into module T as:

$$Internal\ PRS_T = Internal\ PRS_A + Internal\ PRS_B, \quad (2)$$

$$VO\_PRS_T = VO\_PRS_A + VO\_PRS_B. \quad (3)$$

The metric Usage is defined as the maximum usage of module A and B:

$$Usage_T = \max(Usage_A, Usage_B). \quad (4)$$

Let n_{Int}, n_{VO}, V_{TTRISK} and U represent the value of $Interal\ PRS_T$, VO_PRS_T, $TTRISK_T$, and $Usage_T$ respectively, then the DNRISK for this calling pattern derived from the OPRISK model becomes:

$$DNRISK = \frac{1}{1+e^{4.17374-0.18636 \cdot n_{\text{Int}}-0.2357 \cdot V_{\text{TTRISK}}-1.02238 \cdot n_{\text{VO}}}}, \text{ for } U < 0.8;$$

or (5)

$$DNRISK = \frac{1}{1+e^{3.3352-0.1847 \cdot n_{\text{Int}}-0.3842 \cdot V_{\text{TTRISK}}-1.3262 \cdot n_{\text{VO}}}}, \text{ for } U \geq 0.8.$$

The DNRISK for parallel relation can be derived from the vertical relation and propagation as shown in Figure 7. Firstly, module A and B is combined to construct module T1, then module T1 and C is used to build module T2, finally module T2 and D are combined to get the subsystem T for this parallel calling relation. The DNRISK for T1, T2, T is the same as that illustrated in equation (5). Thus, the modified metrics approach can be used in the parallel case also. Note that the whole calling trace file which is presented as a calling tree, bottom to top propagation along with modified metrics technology can provide not only the system DNRISK but also the DNRISK for different subsystems.

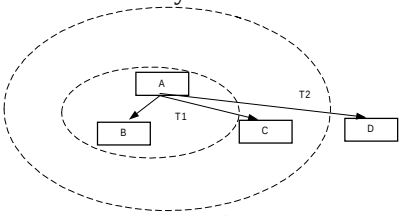

Figure 7. The interpretation of the parallel relationship.

5. Field data validation of DNRISK

A set of field measurement data is provided from the tracing of a switching software release by Nortel Networks in order to demonstrate the improvement of type II error prediction with the proposed DNRISK.

Let A_1 be the set containing the modules which are sound both in risk rate prediction and actual field data and N_1 be the number of module in set A_1;

A_2 be the set containing modules which are accused in risk rate prediction and turns out to be sound in field measurement and N_2 be the number of set A_2 (type I error);

A_3 be the set containing modules that are predicted to be sound but actually are failed from real measurement and N_3 be the number of set A_3 (type II error);

A_4 be the set having modules both accused by risk rate prediction and failed from actual measurement and N_4 be the number of set A_4.

A good risk rate prediction algorithm will reduce the value of N_2 and N_3. In many fault diagnosis applications reduce the value of N_3 (type II error)[12] is more important than reduce the value of N_2 (type I error)[12]. The relation of these defined sets is shown in Figure 8.

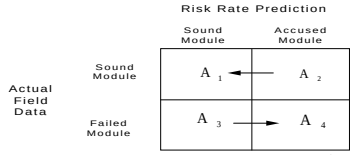

Figure 8. The illustration of defined sets.

The type I error Err_I represents the case where a module is predicted fault-prone and actually is sound in field test and is defined as:

$$Err_I = \frac{N_2}{N_1 + N_2}. \quad (6)$$

The type II error Err_{II} represents the case where a module is predicted sound and found problem from field test and is defined as:

$$Err_{II} = \frac{N_3}{N_3 + N_4}. \quad (7)$$

These two error types can be used to evaluate the accuracy of different risk rate prediction approach.

DNRISK can evaluate the risk rate for a module at different calling level in the trace file. The maximum, mean and minimum of these different risks defined as $DNRISK_{max}$, $DNRISK_{mean}$, and $DNRISK_{min}$ respectively, are used to indicate different aspects of DNRISK for a module:

$$DNRISK_{max} = \max(a_1, a_2, ..., a_n), \quad (8)$$

$$DNRISK_{mean} = \text{mean}(a_1, a_2, ..., a_n), \quad (9)$$

$$DNRISK_{min} = \min(a_1, a_2, ..., a_n), \quad (10)$$

where $a_1, a_2, ..., a_n$ is the DNRISK value evaluated for a module at different calling trace pattern and different calling point.

There are total 418 different modules obtained from the field data for the validation process. Figure 9 to Figure 12 show the distribution of the number of different modules in set A_1, A_2, A_3 and A_4 for the risk rate from OPRISK, $DNRISK_{max}$, $DNRISK_{mean}$, and $DNRISK_{min}$ respetively.

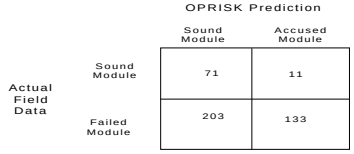

Figure 9. Distribution of number of modules in different set for OPRISK.

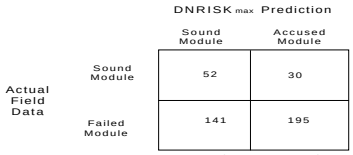

Figure 10. Distribution of number of modules in different set for DNRISKmax.

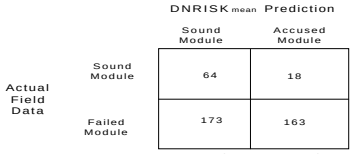

Figure 11. Distribution of number of modules in different set for DNRISKmean.

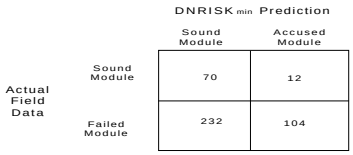

Figure 12. Distribution of number of modules in different set for DNRISKmin.

Table 3 summarizes the corresponding error type I and error type II for OPRISK, $DNRISK_{max}$, $DNRISK_{mean}$, and $DNRISK_{min}$. Table 3 indicates that $DNRISK_{max}$, and $DNRISK_{mean}$ can provides much better performance than OPRISK in terms of type II error for the risk rate prediction, which is more important than type I error in many communication network software fault finding. Though the values for type I error are increased, the difference of the actual number of modules involved is acceptable. For a communication software system which the failure of a module can cause great damage and bring economic lost to the customer, the approach proposed for the DNRISK based on independent module static risk rate OPRISK can provide good indicator to help improving the reliability level of the system operation. Comparing the performance of $DNRISK_{max}$, $DNRISK_{mean}$ and $DNRISK_{min}$, $DNRISK_{max}$ is recommended for the overall DNRISK of a dependent module.

Table 3 Error type I and II for static and DNRISK model.

	OPRISK	$DNRISK_{max}$	$DNRISK_{mean}$	$DNRISK_{min}$
Err_I	0.1341	0.3659	0.2195	0.1463
Err_{II}	0.6042	0.4196	0.5149	0.6905

6. Conclusion

This paper has presented an novel approach to evaluate the dynamic risk rate (DNRISK) for a communication network system software based on the static risk rate OPRISK provided by Emerald tool used in Nortel Networks. Through the investigation of the calling trace file and its corresponding calling tree, two basic module relations, vertical and horizontal relations, are studied. The merge of calling relationship into modified metrics reduces the evaluation of dynamic risk rate problem into a static risk rate problem. Thus the exiting logistical regression model in Emerald can be used to calculated DNRISK value for both the entire system and a module at different tree node level. Field data validation shows that $DNRISK_{max}$ provides a significant improvement in terms of type II error reduction and can be used by the software engineers to help increasing the system reliability.

6. Acknowledgement

The authors would like to acknowledge the support of this work from the Center of Advanced Computing and Communication (CACC) for Grant 5-30336 for the project "Communication System Network Control Software Performance Modeling and Fault Detection".

References

1. Chatterjee, S., R.B. Misra, and S.S. Alam, *Prediction of software reliability using an auto regressive process.* International Journal of Systems Science, 1997. **28**(2): p. 211-216.
2. Chen, M., A.P. Mathur, and V.J. Rego, *Effect of testing techniques on software reliability estimates obtained using a time-domain model.* IEEE Transactions On Reliability, 1995. **44**(1): p. 97-103.
3. Hlady, M., R. Kovacevic, and e.a. J. J. Li. *An Approcach to Automatic Detection of Software Failures.* in *IEEE 6th International Symposium on Software Reliability Engineering*. 1995.
4. Hudepohl, J.P., *et al.*, *Emerald: Software Metrics and Models on the Desktop*, in *IEEE Software*. 1996. p. 56-60.
5. Hudepohl, J.P., *et al. Integrating Metrics and Models for Software Risk Assessment*. in *7th International Symposium on Software Reliability Engineering*. 1996. Los Alamitos, CA.
6. Jones, W. and D. Gregory, *Infinite-Failure Models for a Finite World: A Simulation Study of Fault Discovery.* IEEE Transactions on Reliability, 1994. **43**(4): p. 520-526.
7. Khoshgoftaar, T.M., *et al.*, *Early Quality Prediction: A Case Study in Telecommunicaions*, in *IEEE Software*. 1996. p. 65-71.
8. Krishmamurthy, S. and A.P. Mathur. *On the Estimation of Reliability of a Software System Using Reliabilities of its Components*. in *Eighth International Symposium on Software Reliability Engineering*. 1997.
9. Molin, P. *Designing Reliable Systems from Reliable Components Using the Context-Dependent Constraint Concept*. in *Seventh International Sysmposium on Software Reliability Engineering*. 1996. White Plains, NY: IEEE Computer Society.
10. Weerahandi, S. and R.E. Hausman, *Software Quality Measurement Based on Fault-Detection Data.* IEEE Transactions on Software Engineering, 1994. **20**(9): p. 665-676.
11. Henley, E.J. and K. Hiromitsu, *Reliability engineering and risk assesment*. 1981, Englewood Cliffs, New York: Prentice-Hall.
12. Jones, W., *Emerald Risk Models version 2.0*, . 1996, Nortel: RTP.
13. Chorafas, D.N., *Statistical Processes and Reliability Engineering*. 1960: D. Van Nostrand Company, INC.

Corresponding Author: Bo Li