



Experimentation in Software Engineering: Reading Studies



The Software Engineering Discipline



Software techniques, methods, models, etc. need to be validated via experimentation
refined and tailored to the application environment
logically or physically integrated
be easily transferred into practice

There is a need to understand the relationships between process and product
learn and evolve our knowledge based upon experience

We need an experimental, evolutionary software development framework
that deals with the symbiotic relationship between research and development
so that learning can take place in a practical way

Experimentation can take many forms



Research Paradigms



Need research to establish a scientific and engineering basis for software engineering

Required research methods involve the need to build, analyze, and evaluate models of the software processes and products
various aspects of the environment, e.g the people, the organization and the interactions of these models.

The goal is to develop the conceptual scientific foundations of software engineering upon which future researchers can build.

This is often a process of discovering and validating small but important concepts that can be applied in many different ways and that can be used to build more complex and advanced ideas rather than merely providing a tool or methodology without experimental validation of its underlying assumptions or careful analysis and verification of its properties



Research Paradigms Definitions



A **fact** is information obtained through direct observation

A **hypothesis** is an educated guess that precedes an experiment

An **experiment** is

- a test, trial or tentative procedure policy;
- an act or operation for the purpose of discovering something unknown or of testing a principle, supposition, etc.;
- an operation carried out under controlled conditions in order to discover an unknown effect or law, to test or establish a hypothesis, or to illustrate a known law



Research Paradigms Definitions



A **theory** is a possible explanation based upon many facts and reason

A **law** is a description/observation of behavior used for prediction based upon facts and reason

A **model** is a simplified representation of a system or phenomenon with any hypotheses required to describe the system or explain the phenomenon, often mathematically. A model can be a theory or a law

A **paradigm** is conceptual filter that determines how we perceive/interpret

A **truth** is what really is



Research Paradigms



Research methods from other disciplines include various forms of experimental or analytic paradigms

Experimental paradigms require an experimental design, observation, data collection and validation on the process or product being studied

The Scientific Method:

- observe the world,
- propose a model or a theory of behavior,
- measure and analyze,
- validate hypotheses of the model or theory,
- and, if possible, repeat the procedure.



Research Paradigms



The Scientific Method

is an inductive paradigm which can be used to:
understand the software process, product, people, environment
extract models from the world that explain underlying phenomena, and
evaluate if the model is representative of the phenomenon observed

Example: an attempt to understand the way software is developed by an organization to see if their process model can be abstracted or a tool can be built to automate the process

There are two variations of the inductive paradigm which we will call the engineering method
empirical method



Research Paradigms



The Engineering Method:

observe existing solutions,
propose better solutions,
build/develop,
measure and analyze, and
repeat the process until no more improvements appear possible.

This version of the paradigm is

an evolutionary improvement oriented approach
assumes models exist
modifies model to improve the thing being studied

Example: study improvements to methods or demonstrate that a tool performs better than its predecessor relative to certain characteristics



Research Paradigms



The Empirical Method:

propose a model,
develop statistical/qualitative methods,
apply to case studies,
measure and analyze,
validate the model and repeat the procedure.

This version of the paradigm is

a revolutionary improvement oriented approach
proposes a new model
studies effects of process or product suggested by the new model

Example: proposal of a new method or tool and validation that the model or tool is an advance over current models or tools



Research Paradigms



There must be a rationale for collecting data.

Experiments must be designed to acquire information useful for the building of a suitable description (model or theory) of the systems under study. It is an approach to model/theory/law building.

Experimentation alone is of no value if there is no underlying framework or context where experimental results can be interpreted.

Other issues involved in these inductive, experimental methods include

- the types of experimental design appropriate for different environments,
- whether the experiment is exploratory or confirmatory,
- the validity of the data collected,
- the cost of the experiment,
- the problems of reproducibility, etc.



Research Paradigms



An analytic paradigm is:

The Mathematical Method:

propose a formal theory or set of axioms,
develop a theory,
derive results, and
if possible compare with empirical observations.

This is a deductive analytical model which

does not require experimental design in the statistical sense, but
provides a framework for
developing models and understanding their boundaries
based upon manipulation of model itself

Example: the treatment of programs as mathematical objects and the analysis of the mathematical object or its relationship to the program



Research Paradigms



These paradigms serve as a basis for distinguishing research activities from development activities

If one of these paradigms is not being used in some form, the study is most likely not a research project

Many projects that claim to be research are simply developments, e.g., building a system or tool alone is development, not research

Research involves gaining understanding about how and why a certain type of tool might be useful, and by validating that a tool has certain properties or effects by carefully designing an experiment to measure the properties or to compare it with alternatives

The scientific method can be used to understand the effects of a particular tool in some environment and to validate hypotheses about how software development can best be accomplished



The Experimental Discipline



How do we combine experiments?

There are several different approaches to experimenting in the software domain

The approaches vary in

- Level of variable relationship
- Level of confidence in results, insights gained
- Experience of subjects
- Environmental setting
- Balance between quantitative/qualitative research
- Cost



The Experimental Discipline



Classes of Experimental Studies

Experiment Classes

		#Projects	
		One	More than one
# of	One	Single Project	Multi-Project Variation
Teams			
per Project	More than one	Replicated Project	Blocked Subject-Project



The Experimental Discipline



Sign of maturity in a field:

level of sophistication of the goals of an experiment
understanding interesting things about the discipline

For software engineering that might mean:

Can we build models that allow use to measure and differentiate processes and products?

Can we measure the effect of a change in a particular process variable on the product variable?

Can we predict the characteristics of a product (values of product variable) based upon the model of the process (values of the process variables), within a particular context?

Can we control for product effects, based upon goals, given a particular set of context variables?



The Experimental Discipline



Sign of maturity in a field:

a **pattern of knowledge** built from a **series of experiments**

Does the discipline build on prior (knowledge, models, experiments).

Was the study an isolated event?

Did it lead to other studies that made use of the information obtained from it

Have studies been replicated under similar or differing conditions?

Does the building of knowledge exist in one research group or environment, or has it spread to others - researchers building on each other's experimental work?

For example, inspections, in general, are well studied experimentally

However, there has been very little combining of results, replication, analysis of the differentiating variables



Studying Process Effects



Methods and Techniques

A **technique** is a technical procedure for constructing or assessing software, that requires skill, and produces a technical result, e.g., reading, testing

A **method** is a management procedure for applying software techniques, with a set of rules stating how and when to apply and when to start and stop applying the technique (entry and exit criteria), which technique is appropriate, and how to evaluate it (management support), e.g., design inspections, test plans.

We need to understand

the relationship between techniques and methods
the dimensions of both
how to improve them for a particular environment



Reading Techniques



Reading is a **key technical activity** for analyzing and constructing software artifacts

Reading is a **model for writing**

Reading is **critical for reviews, maintenance, reuse, ...**

What is a reading technique?

a concrete set of instructions given to the reader saying how to read and what to look for in a software product

More Specifically, software reading is

the individual analysis of a software artifact

e.g., requirements, design, code, test plans

to achieve the understanding needed for a particular task

e.g., defect detection, reuse, maintenance



Dimensions of a Reading Technique



- Input object: Requirements, specification, design, code, test plan,...
- Output object: Set of anomalies
- Approach: Sequential, path analysis, stepwise abstraction,...
- Formality: Reading, correctness demonstrations,...
- Emphasis: Fault detection, traceability, performance,...
- Method: Walk-throughs, inspections, reviews,...
- Consumers: User, designer, tester, maintainer,...
- Product qualities: Correctness, reliability, efficiency, portability,...
- Process qualities: Adherence to method, integration into process,...
- Quality view: Assurance, control,...



Reading Techniques



Early experiments (Hetzel, Meyers) showed very little difference between reading and testing

But reading was simply reading, without a technological base

We discuss a series of experiments at the University of Maryland and at NASA used to learn about, evaluate, and evolve reading techniques

This example

- shows **multiple experimental designs**
- provides a combination of **evaluation approaches**
- offers insight into the **effects of different variables** on reading

The experiments start with the early reading vs. testing experiments to various Cleanroom experiments to the scenario based reading techniques currently under study



EXPERIMENTAL LEARNING MECHANISMS



Series of Studies

		# Projects	
		One	More than one
# of Teams	One	3. Cleanroom (SEL Project 1)	4. Cleanroom (SEL Projects, 2,3,4,...)
per Project	More than one	2. Cleanroom at Maryland	1. Reading vs. Testing 5. Scenario reading vs. ...



EVALUATION OF A PROCESS



When introducing any form of process, method or tool, the organization needs to evaluate its effectiveness

That effectiveness of a process can be measured by

- higher than normal quality
- cheaper development costs
- improved cycle time to delivery
- improved product functionality
- more predictable behavior

It is important to understand the relationship between the process and the product

It is important to have a data as a basis of comparison



Blocked Subject Project Study



Testing/Reading Strategies Comparison

Goals:

Analyze code reading, functional testing and structural testing to evaluate and compare them with respect to their effect on fault detection effectiveness, fault detection cost and classes of faults detected from the viewpoint of quality assurance

Environment:

NASA/CSC and the University of Maryland
Text formatter, plotter, abstract data type, database
Seeded with software faults (9, 6, 7, 12)
145 - 365 LOC

Experimental design:

Blocked subject-project: Fractional factorial design
Three applications
74 subjects: 32 NASA/CSC, 42 UM



Blocked Subject Project Study Testing/Reading Strategies Comparison



Technique Definition

Code Reading: Reading by Stepwise Abstraction

read a sequence of statements and abstract the function they compute repeat until the function of the entire program has been abstracted and can be compared with the specification

Functional Testing: Boundary Value Equivalence Partition Testing

divide the requirements into valid and in valid equivalence classes and make up tests that check the boundaries of the classes

Structural Testing: Achieving 100% statement coverage

make up a set of tests that guarantee that 100% of the statement in the program have been executed



Blocked Subject Project Study Testing/Reading Strategies Comparison



Fractional Factorial Design

Each Subject applies each of the treatments (techniques) on a project

The techniques are the **independent** variable (the item being studied)

The effects on the product (number and type of faults identified, time to identify the faults, ...) are the **dependent** variables

The **context** variables are the elements of the environment being studied, e.g., the experience of the people applying the techniques

The design allow us to minimize the effect of the individual

- Each subject uses each technique and tests each program
- We can block according to experience level and program tested
 - this allows us to identify the effects of each of those variables



Blocked Subject Project Study Testing/Reading Strategies Comparison



Fractional Factorial Design

		Code Reading			Functional Testing			Structural Testing		
		P1	P2	P3	P1	P2	P3	P1	P2	P3
Advanced Subjects	S1			X			X			X
	S2	X			X					X
	:									
	S8	X				X				X
Intermediate Subjects	S9		X		X			X		
	S10	X			X					X
	:									
	S19	X				X				X
Junior Subjects	S20		X		X			X		
	S21	X			X					X
	:									
	S32	X				X				X

Blocking according to experience level and program tested
Each subject uses each technique and tests each program

NASA/CSC



Blocked Subject Project Study Testing/Reading Strategies Comparison



Major Conclusions (NASA/CSC)

Fault Detection Effectiveness

4.0 Faults found on average (SD = 1.9, 50.0%)
Code reading > (functional > structural)

Fault Detection Rate

1.82 faults/hour average (SD = 1.80)
Code reading > (functional ~ structural)

Total Fault Detection Time

3.3 hours testing on average (SD = 2.19)
No difference in techniques

Classes of Faults Detected

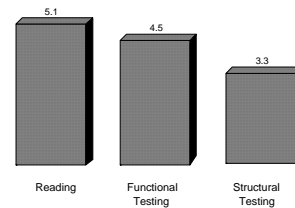
Omission: (code reading ~ functional) > structural
Initialization: (code reading ~ functional) > structural
Interface: code reading > (functional ~ structural)
Computation: code reading > structural
Control: functional > (code reading ~ structural)



Blocked Subject Project Study Testing/Reading Strategies Comparison



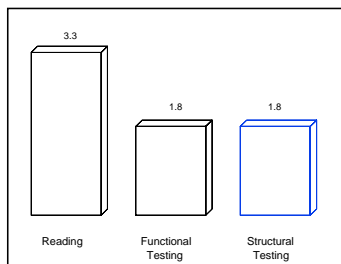
Major Conclusions (NASA/CSC) Fault Detection Effectiveness (Mean)



Blocked Subject Project Study Testing/Reading Strategies Comparison



Major Conclusions (NASA/CSC) Fault Detection Rate (Faults/hour)



Blocked Subject Project Study Testing/Reading Strategies Comparison



Major Conclusions

Self-estimates during study:

code reading > structural > functional

After completion of study:

over 90% of the participants thought functional testing worked best

Based upon this study

reading was implemented as part of the SEL development process

But - reading appeared to have very little effect

Hypothesis 1: People did not read as well as they should have as they believed that testing would make up for their mistakes

Experiment: If you read and cannot test you do a more effective job of reading than if you read and know you can test.

Hypothesis 2: There is a confusion between reading and the method in which it is embedded, e.g., inspections



Blocked Subject Project Study Testing/Reading Strategies Comparison



Lessons Learned

Reading using a particular technique is more effective and cost effective than specific testing techniques

**The reading technique is important
but**

Different approaches may be more effective for different types of defects

Reader needs to be motivated to read better

The reading motivation is important

We may need to better support the reading process

The reading technique may be different from the reading method

The Cleanroom approach seemed to cover a couple of these issues so we tried a controlled experiment at the University of Maryland



Reading-Based Life Cycle Model Cleanroom Process



Key components:

- Mathematically-based design methodology
- Function specification for programs
- State machine specification for modules
- Reading by stepwise abstraction
- Correctness demonstrations when needed
- Top-down development

Implementation without any on-line testing by developer

Independent testing

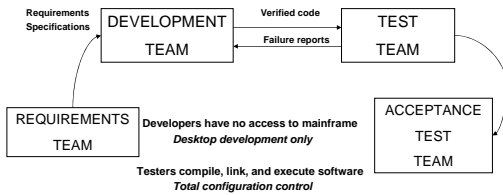
- Statistically based on anticipated operational use
- Quality assurance orientation



EVALUATION OF A PROCESS Process Definition



Cleanroom Process



PROCESS MODEL DEFINITION



Cleanroom Process

Mathematically-based design methodology

Function specification for programs

function: $[f] = f$
 sequence: $[f] =]g;h] = \{(x,y) \mid y = h(g(x))\}$
 if then: $[f] = [if\ p\ then\ g] = \{(x,y) \mid (p(x) = True\ and\ y = g(x))\ or\ p(x) = False\ and\ y = h(x)\}$
 if then else: $[f] = [if\ p\ then\ g\ else\ h] = \{(x,y) \mid (p(x) = True\ and\ y = g(x))\ or\ (p(x) = False\ and\ y = h(x))\}$
 while do: $[f] = [while\ p\ do\ g] = [if\ p\ then\ g; f]$



PROCESS MODEL DEFINITION



Cleanroom Process

Mathematically-based design methodology

State machine specification for modules

State machine, m , a function from an ordered pair of inputs into ordered pair of outputs
 $(newstate,output) = m(current\ state,\ input)$

Interpretation: after machine has operated, the newstate becomes the oldstate for next operation.

User: Only inputs & outputs "observable", so operation may appear non-functional.

Designer: Can observe the state data, so the behavior is completely functional.

A module state machine is the state machine defined by a module

- the input variables and their types define the inputs
- the output variables and their types define the outputs
- the retained variables and their types
- the text of the module programs define the transition rules
- a distinguished input may cause a transition from any state to a known initial state, or the presence of initial data may create an initial state

PROCESS MODEL DEFINITION

Cleanroom Process

Mathematically-based design methodology

Reading by stepwise abstraction

sequence:

Read the successive parts and summarize their sequential effect in the final outcome

ifthenelse:

Read the if test true and the then part, then read the if test false and the else part, and summarize the two outcomes into a single outcome

whiledo:

Read the while test true and the do part (be sure to verify that the whiletest eventually evaluates to false), then read the whiletest false (no operation), and summarize the outcome at exit





PROCESS MODEL DEFINITION



Cleanroom Process

Mathematically-based design methodology

Correctness demonstrations when needed

Correctness demonstrations are done, when needed, by building trace tables of the function variables and doing symbolic execution

Because a state machine is a special type of function, we can use the function methodology in the design and verification of modules



Replicated Project Study



Cleanroom Study

Study Goal:

Analyze the Cleanroom process in order to evaluate and compare it to a non-Cleanroom process with respect to the effects on the process, product and developers from the viewpoint of quality assurance

Environment:

University of Maryland
Electronic message system, ~ 1500 LOC

Experimental design:

Replicated project, 15 three-person teams (10 used Cleanroom)
3 to 5 test submissions
Data collected: Background,
Attitude survey
On-line activity
Testing results



Replicated Project Study Cleanroom Evaluation



Major Results

Effect on the Software Development Process

Cleanroom developers

felt they more effectively applied off-line review techniques, while others focused on functional testing
spent less time on-line and used fewer computer resources
tended to make all their scheduled deliveries

Effect on the Delivered Product

Static properties: less dense complexity, higher percentage of assignment statements, more global data, more comments

Operational properties: product more completely met requirements, higher percentage of test cases succeeded

Effect on Developers

Missed program execution
Modified their development style
Would use it again



Cleanroom in the SEL



Using the QIP for Cleanroom in the SEL

Characterize: What are the relevant models, baselines and measures? What are the existing processes? What is the standard cost, relative effort for activities, reliability? What are the high risk areas?

Set goals: What are the expectations, relative to the baselines? What do we hope to learn, gain, e.g., Cleanroom with respect to changing requirements?

Choose process: How should the Cleanroom process be modified and tailored relative to the environment? E.g., formal methods hard to apply, require skill; may have insufficient data to measure reliability. Allow back-out options for unit testing certain modules.

Execute: Collect and analyze data based upon the goals, making changes to the process in real time

Analyze data: Try to characterize and understand what happened relative to the goals; write lessons learned

Package experience: modify the process for future use



Single Project Study



Cleanroom in the SEL

Study Goal:

Analyze the Cleanroom process in order to evaluate and compare it to a standard SEL development process with respect to the effects on the effort distribution, cost, and reliability from the viewpoint of quality assurance

Environment:

NASA/SEL
40 KLOC Ground Support System

Experimental design:

Case Study
Data collected: effort distribution, change profile, productivity, level of rework, impact of spec changes, error rate, error distribution, error source



Single Project Study Cleanroom in the SEL



Sample Measures, Baselines, and Expectations

	Sample Measures	Sample Baseline	Sample Expectation	
PROCESS	Effort distribution Change profile		Increased design % due to emphasis on peer review process	
	Productivity			
COST	Level of rework Impact of spec changes	Historically, 26 DLCC per day	No degradation from current level	
	<td>RELIABILITY</td> <td>Error rate Error distribution Error source</td> <td>Historically, 7 errors per KDLOC</td> <td>Decreased error rate</td>	RELIABILITY	Error rate Error distribution Error source	Historically, 7 errors per KDLOC



Single Project Study Cleanroom in the SEL



Major Results

Can use for up to 40KLOC
 Can use with changing requirements
 Failure rate during test reduced by 25%
 Reduction in rework effort: 95% as opposed to 58% took <1 hour to fix
 Productivity increased by about 30%
 Effort distribution changes: more time in design
 50% of code time spent reading
 Code appears in library: later than normal, more like a step function
 Less computer use by a factor of 5
 Concerns: Only 26% of faults found by both readers
 Formal Methods not applied effectively
 No payoff in reliability modeling

Evolution: Better training needed for methods and techniques
 Better mechanisms needed for uploading code to testers
 To allow testers to add requirements for output analysis of code

Side effect: Caused more emphasis on requirements analysis



Multi-Project Analysis Study

Cleanroom in the SEL



Study Goal:

Analyze the Cleanroom process in order to evaluate and compare it to a standard SEL development process with respect to the effects on the effort distribution, cost, and reliability

Changes:

Better training for methods and techniques - use box structure approach
 Fix uploading problem
 Allow clean compile
 Allow testers to add requirements for output analysis of code

Environment:

Project 2: 22 KLOC Flight Dynamics System (in-house)
 Project 3: 160 KLOC Flight Dynamics System (contractor)
 Project 4: 140 KLOC Flight Dynamics System (contractor)

Experimental design:

Multi-Project Study
 Data collected: effort distribution, change profile, productivity, level of rework, impact of spec changes, error rate, error distribution, error source



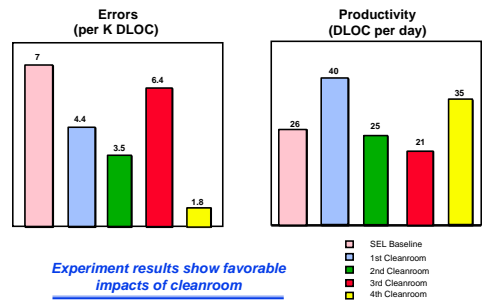
Multi-Project Analysis Study Cleanroom in the SEL



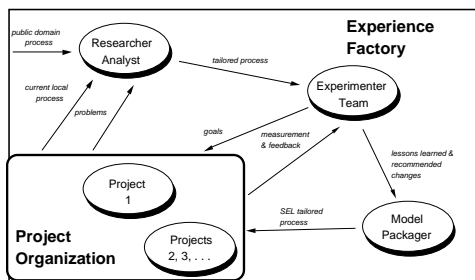
	1st Cleanroom	2nd Cleanroom	3rd Cleanroom	4th Cleanroom
Project	ACME	SAMPEX	WIND/POLAR	SOHO AGSS
Size (developed lines)	40 KSLOC	23 KSLOC	160 KSLOC	141 ICSLOC
Size (total lines)	60 KLOC	39 KLOC	201 KLOC	485 KLOC
Dates	1988-90	1990-91	1990-92	1993-1995
Function	Attitude determination (Math)	Telemetry processing (data processing)	Full attitude (two missions)	Full attitude



Multi-Project Analysis Study Cleanroom in the SEL



Multi-Project Analysis Study Improving via the Experience Factory Process Evolution/Evaluation



Multi-Project Analysis Study Cleanroom in the SEL



Major Results:

In-house, smaller project
 very effective, reduced defects, slightly lower cost
 Contractor, larger project
 not as effective at first, very effective second time
 Uploading Fixed
 Formal Methods still not effectively applied
 There was a duplication of documentation

Evolution:

Although Cleanroom successful,
 there are still problems reading and abstracting code formally
 there are problems reading requirements and other documents
 Area for study in the continual evolution of Cleanroom include:
 improve reading techniques for requirements and design documents
 To improve the existing reading techniques, we first spent time
 understanding how they were reading the document
 asked for problems and suggestions

Based upon these ideas, we have been working on reading approaches



References



- V. Basili, The Experimental Paradigm in Software Engineering, Lecture Notes in Computer Science 706, *Experimental Software Engineering Issues: Critical Assessment and Future Directives*, H.D. Rombach, V. Basili, and R. Selby editors, Proceedings of Dagstuhl-Workshop, September 1992, published by Springer-Verlag, #706, Lecture Notes in Computer Software, August 1993.
- D. Campbell and J. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Houghton Mifflin Co., Boston: 1963.
- V. Basili, R. Selby, and D. Hutchens, Experimentation in Software Engineering, IEEE Transactions on Software Engineering (invited paper), July 1986.
- R. Selby, V. Basili, and T. Baker, Cleanroom Software Development: An Empirical Evaluation, IEEE Transactions on Software Eng. , pp. 1027-1037, September 1987.
- V. Basili and R. Selby, Comparing the Effectiveness of Software Testing Strategies, IEEE Transactions on Software Engineering, pp. 1278-1296, December 1987.
- V. Basili and R. Selby, Paradigms for Experimentation and Empirical Studies in Software Engineering, Reliability Engineering and System Safety, vol. 32, no. 1-2, pp. 171-193, 1991.



REFERENCES



- V. Basili and S. Green, Software Process Evolution at the SEL, IEEE Software, pp. 58-66, July 1994.
- A.A. Porter, L.G. Votta, and V. Basili, Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment, IEEE Transactions on Software Engineering, Volume 21, Number 6, pp. 563-575, June 1995.
- V. Basili, S. Green, O. Laitenberger, F. Shull, S. Sorumgaard, and M. Zelkowitz, The Empirical Investigation of Perspective-based Reading, Empirical Software Engineering, An International Journal, Volume 1, Number 2, pp. 133-164, Kluwer Academic Publishers, October 1996.
- V. Basili, Evolving and Packaging Reading Technologies, The Journal of Systems and Software, Volume 38, Number 1, pp. 3-12, July 1997.
- V. Basili and R. Selby, Data Collection & Analysis in Software Research and Management (invited paper), Proceedings of the American Statistical Association, July 1984.
- V. Basili, L. Briand, S. Condon, Y. Kim, W. Melo, and J. Valett, Understanding and Predicting the Process of Software Maintenance Releases, 18th International Conference on Software Eng., (ICSE'18), Berlin, Germany, March 25-29, 1996.