光るソフトウェア工学研究

大阪大学 大学院情報科学研究科／
産業技術総合研究所 情報技術研究部門

井 上　克 郎

「光るソフトウェア工学研究」とは

光る研究　→

自分がワクワクし元気になる研究

・井上の研究の振り返り

・RQ1：　光るソフトウェア工学研究とは？

・RQ2：　どうすれば光るのか？

~~萌えるSE研究~~
光る

1978年　嵩 忠雄先生の研究室へ

1999 IEEE Claude E. Shannon Award

- 符号理論、暗号
- 形式的検証、項書換え系
- ソフトウェア開発環境
- マイクロプログラム計算機

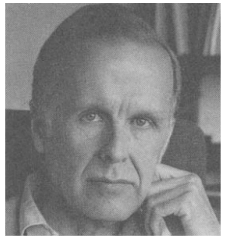
1981年　修士論文

. . . Backus headed a small IBM group in New York City during the early 1950s. The earliest product of this group's efforts was a high-level language for scientific and technical com-

widely used programming languages in the world. Almost all programming languages are now described with some type of formal syntactic definition.' "

Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs

John Backus
IBM Research Laboratory, San Jose



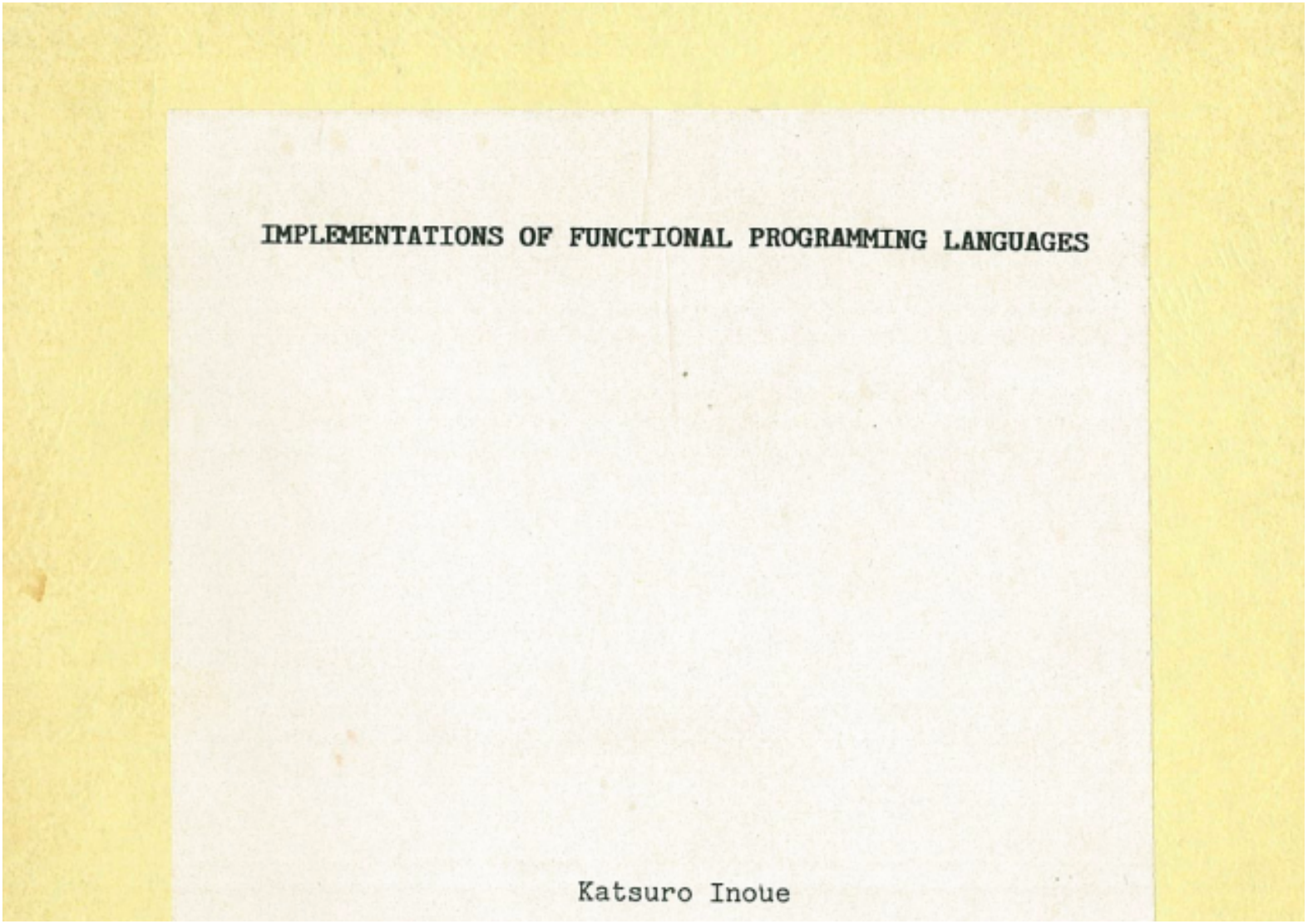
Author's address: 91 Saint Germain Ave., San Francisco, CA 94114.

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak: their primitive word-at-a-time style of programming inherited from their common ancestor—the von Neumann computer, their close coupling of semantics to state transitions, their division of programming into a world of expressions and a world of statements, their inability to effectively use powerful combining forms for building new programs from existing ones, and their lack of useful mathematical properties for reasoning about programs.

An alternative functional style of programming is founded on the use of combining forms for creating programs. Functional programs deal with structured data, are often nonrepetitive and nonrecursive, are hierarchically constructed, do not name their arguments, and do not require the complex machinery of procedure declarations to become generally applicable. Combining forms can use high level programs to build still higher level ones in a style not possible in conventional languages.

1977 ACM Turing Award受賞者のJohn Backusが提案する関数型言語のインタプリタ作成

1984年　博士論文

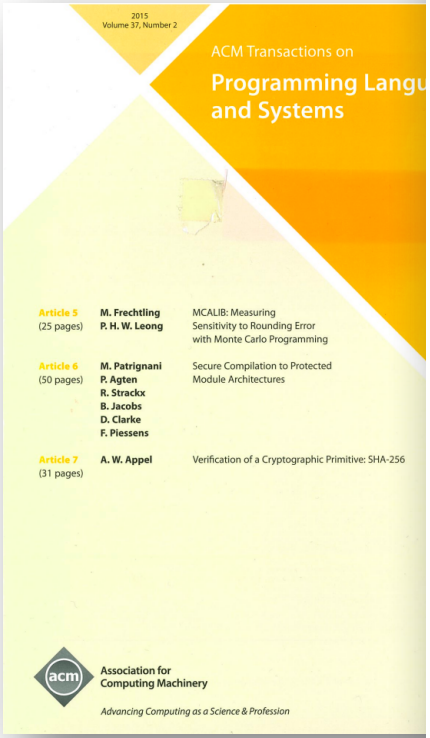


IMPLEMENTATIONS OF FUNCTIONAL PROGRAMMING LANGUAGES

Katsuro Inoue

そうだ
ハワイ、
行こう。

ハワイでの2年間の研究成果



Analysis of Functional Programs to Detect Run-Time Garbage Cells

KATSURO INOUE
University of Hawaii at Manoa, Honolulu
and
HIROYUKI SEKI and HIKARU YAGI
Osaka University, Osaka

We propose a method for detecting the generation of garbage cells by analyzing a source text written in a functional programming language which uses ordinary linked lists to implement list-type values. For a subexpression such as $F(G(\cdots))$ in a program where the function values of F and G are of list type, if a cell c is created during the computation of G and if c does not appear in a list-type value of F, then c becomes a garbage cell at the end of the computation of F. We discuss this problem on the basis of formal languages derived from the functional program text and show some sufficient conditions that predict the generation of garbage cells. Also, we give an efficient algorithm to detect at compile time the generation of garbage cells which are linearly linked. We have implemented these algorithms in an experimental LISP system. By executing several sample programs on the system, we conclude that our method is effective in detecting the generation of garbage cells.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*applicative languages*; D.3.4 [**Programming Languages**]: Processors—*optimization*; E.2 [**Data**]: Data Storage Representations—*linked representation*

General Terms: Languages, Performance

Additional Key Words and Phrases: Created occurrences, garbage collection, noninherited occurrences

1. INTRODUCTION

Much attention is now focused on functional programming languages because of their mathematical elegance. As a result, studies have been conducted on the implementation of functional languages. Since garbage collection is one of the most expensive processes in many implementations, numerous garbage collection algorithms have been proposed and actually implemented [5]. However, little has ...cting the generation of ...eration of such cells in ... functional program. A

Authors' current address: Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, Toyonaka, Osaka, 560, Japan.

79 citations

1986年　大阪大学に戻り鳥居研究室へ

鳥居宏次先生

- エキスパートシステム
- 囲碁プログラム
- 開発環境
- ソフトウェアプロセス

ICSE'87, Monterey, CA

SOFTWARE PROCESSES ARE SOFTWARE TOO

Leon Osterweil

University of Colorado Boulder, Colorado USA

1. The Nature of Process.

The major theme of this meeting is the exploration of the importance of .ul process as a vehicle for improving both the quality of software products and the the way in which we develop and evolve them. In beginning this exploration it seems important to spend at least a short time examining the nature of process and convincing ourselves that this is indeed a promising vehicle.

We shall take as our elementary notion of a process that it is a systematic approach to the creation of a product or the accomplishment of some task. We observe that this characterization describes the notion of process commonly used in operating systems-- namely that a task executing on a single computi zation is much broader, however, used to carry out work or achieve a goal in an orderly way. Our processes need not even be executable on a computer.

It is important for us to recognize that the notion of process is

description defines a class or set of objects related to each other by virtue of the fact that they are all activities which follow the dictated behavior. We shall have reason to return to this point later in this presentation.

For now we should return to our consideration of the intuitive notion of process and study the important ramifications of the observations that 1) this notion is widespread and 2) exploitation of it is done very effectively by humans. Processes are used to effect generalized, indirect problem solving. The essence of the process exploitation paradigm seems to be that humans solve problems by creating process descriptions and then instantiating processes to solve individual problems. Rather than repetitively and directly solving individual instances of problems, humans prefer to create descriptions and make them available ers) to solve individual prob-

One significant danger in this approach is that the process itself is a dynamic entity and the process description is a static entity. Further, the static process description is often

1,243 citations

SDA: A Novel Approach to Software Environment Design and Construction†

Kouichi Kishida* , Takuya Katayama‡, Masatoshi Matsuo*, Isao Miyamoto††,
Koichiro Ochimizu**, Nobuo Saito‡‡, John H. Sayler†††, Koji Torii***, Lloyd G. Williams‡‡‡

Abstract

A Software Designer's Associate (SDA) is a workstation-based collection of tools which support: 1) the description, evaluation and comparison of software system architectural designs, and 2) cooperation among, and management of, a team of software designers [Ridd87]. Each Software Designer's Associate is a specific instance of a generic facility which supports a team member's design activities, cooperation among team members, and overall team management. It provides a framework for the integration of *tools* supporting the use of various *notations* within the context of a particular set of technical and managerial *methods*. These tools, notations and methods may be adapted to support the needs of a particular project or the habits of an ind
selecting the particular tools to be added t

The Software Designer's Associate pro
involving a consortium of researchers
industrial organizations in both Japan and the United States. This paper describes the concept of Software Designer's Associates and the cooperative, international project which is

and several projects incorporate one or more of these technologies. This paper describes one such effort, the Software Designer's Associate project.

A Software Designer's Associate (SDA) is a workstation-based collection of tools which support: 1) the description, evaluation and comparison of software system architectural designs, and 2) cooperation among, and management of, a team of software designers [Ridd87]. Each Software Designer's Associate is a specific instance of a generic facility which supports a team member's design activities, cooperation among team members, and overall team management. It provides a framework for the integration of *tools* supporting the use of various *notations* within the context of a particular
nagerial *methods*. These tools,
y be adapted to support the needs of
habits of an individual developer by
ls to be added to the generic facility.

A Software Designer's Associate may be viewed as a specialized software environment, one whose facilities are restricted to support for preliminary design activities. Each

26 citations

ICSE'89 Pittsburg, PA

A Formal Adaptation Method for Process Descriptions

Katsuro Inoue, Takeshi OGIHARA, Tohru Kikuno, and Koji Torii

Department of Information and Computer Sciences, Faculty of Engineering Science
Osaka University, Toyonaka, Osaka 560, Japan

Abstract

Requirement to describing software development processes in formal manners has been increased, and demand for altering and tailoring the process descriptions has been emerged. In this paper, we propose a functional language PDL (Process Description Language), designed to describe various development processes under a certain environment. To create and modify the PDL scripts easily and correctly, we propose a method of stepwise refinement from abstract scripts into concrete scripts. By this method, the abstract definitions of software process flow and product flow initially given as function definitions in PDL, are tr
definitions of the tool activations
on. We also discuss an architec
development environment (Adapt
Environment), which can be adapted in many ways for designer's requirements.

them is generally limited. However, the software developers may want to change their working SDE's for various reasons.

For example, different development tasks will require different sets of supporting tools, and different developer habits will require different facilities of tools. Since these differences are quite extensive, it is unrealistic to construct a single system that will support all tools and all habits. It is much more effective to provide a generic system that can be customized to meet the developer habits, tuned to increase
d new tools, and ported to change
ftware, according to the need of
ecific development tasks [9].

These operations to create a specific system from a generic

プロセス研究

1990年代に入っても継続

・いろいろなプロセス定義言語

・プロセス実行環境

International Software Process Workshop

- １週間の合宿 （Napa, ... ）
- 20-30名程度の参加者
  - Osterweil, Balzer, 片山先生、...
- プロセスの言語、記述法、適応法、メタ言語、…
- 自分は１週間ほとんど発言できず...

すこしきっちりしたことやりたいな

役立つツール作りたいな

PROGRAM SLICING*

Mark Weiser

Computer Science Department
University of Maryland
College Park, MD 20742

Abstract

Program slicing is a method used by experienced computer programmers for abstracting from programs. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a "slice", is an independent program guaranteed to faithfully represent the original program within the domain of the specified subset of behavior.

Finding a slice is in general unsolvable. A dataflow algorithm is presented for approximating slices when the behavior subset is specified as the values of a set of variables at a statement. Experimental evidence is presen... slices are used by programmers ... Experience with two automatic s... summarized. New measures of pr... are suggested based on the organization of a program's slices.

KEYWORDS: debugging, program maintenance, software tools, program metrics, human factors, data-

behavior is of interest. For instance, during debugging a subset of behavior is being corrected, and in program modification or maintenance a subset of behavior is being improved or replaced. In these cases, a programmer starts from the program behavior and proceeds to find and modify the corresponding portions of program code. Code not having to do with behavior of interest is ignored. Gould and Dronkowski (1974) report programmers behaving this way during debugging, and a further confirming experiment is presented below.

A programmer maintaining a large, unfamiliar program would almost have to use this behavior-first approach to the code. Understanding an en... only a small piece would ... Since most program mainte... ...ons other than the program ...67 percent of programming ...tenance (Zelkowitz, Shaw, and Gannon 1979), decomposing programs by behavior must be a common occurence.

Automatic slicing requires that behavior be specified in a certain form. If the behavior of

4,346 citations

TSE '84

Program Slicing

MARK WEISER

Abstract—**Program slicing is a method for automatically decomposing programs by analyzing their data flow and control flow. Starting from a subset of a program's behavior, slicing reduces that program to a minimal form which still produces that behavior. The reduced program, called a "slice," is an independent program guaranteed to represent faithfully the original program within the domain of the specified subset of behavior.**

Some properties of slices are presented. In particular, finding *statement-minimal* slices is in general unsolvable, but using data flow analysis is sufficient to find approximate slices. Potential applications include automatic slicing tools for debugging and parallel processing of slices.

Index Terms—**Data flow analysis, debugging, human factors, parallel processing, program maintenance, program metrics, slicing, software tools.**

Introduction

LARGE computer programs must be
derstanding and manipulation by pe
decomposition is useful to people, but s
position into procedures and abstract data types—are very useful. Program slicing is a decomposition based on data flow and control flow analysis.

1,087 citations

Definitions

This section considers programs without procedure calls. Procedures are discussed later. The first few definitions review the standard definitions of digraph, flowgraph, and computation in terms of state trajectory. Finally, a slice is defined as preserving certain projections from state trajectories.

The next few definitions simply establish a terminology for graphs, and restrict attention to programs whose control structure is single-entry single-exit ("hammock graphs").

Definition: A *digraph* is a structure ⟨*N*
of nodes and *E* is a set of edges in *N* X
then *n* is an *immediate predecessor* of *m*
successor of *n*. A *path* from *n* to *m* of le
$p_0, p_1, \cdots, p_k$ such that $p_0 = n$, $p_k = m$

a structu
a member
odes in *N*.
the initial node. If *m* and *n* are two no
n if *m* is on every path from n_0 to *n*.

Definition: A *hammock graph* is a str


IEEE TRANSACTIONS ON
SOFTWARE ENGINEERING
FEBRUARY 1985 VOLUME SE-11 NUMBER 2 (ISSN 0098-5589)
A PUBLICATION OF THE IEEE COMPUTER SOCIETY
PAPERS
Parallel Processing
Task Scheduling on the PASM Parallel Processing System ..... D. L. Twomokos and H. J. Siegel 145
Practice
Evaluating Software Development by Analysis of Changes: Some Data from the Software Engineering Laboratory ..... D. M. Weiss and V. R. Basili 157
Applying Formal Specification to Software Development in Industry ..... J. J. Hayes 169
IOTA: A Modular Programming System ..... T. Yuasa and R. Nakajima 179
Performance Evaluation
Some Empirical Observations on Program Behavior with Applications to Program Restructuring ..... J. R. Peachey, R. B. Bunt, and C. J. Colbourn 188
Product-Form Synthesis of Queueing Networks ..... S. Balsamo and G. Iazeolla 194
Database
Evaluation of the File Redundancy in Distributed Database Systems ..... S. Muro, T. Ibaraki, H. Miyajima, and T. Hasegawa 199
Implementing Distributed Read-Only Transactions ..... A. Chan and R. Gray 205
On the File Design Problem for Partial Match Retrieval ..... H-C. Du 213
Software Models and Metrics
Hardware-Related Software Errors: Measurement and Analysis ..... R. K. Iyer and P. Velardi 223
An Experimental Study of Software Metrics for Real-Time Software ..... H. A. Jensen and K. Vairavan 231
CORRESPONDENCE
On the Asymptotic Optimality of First-Fit Storage Allocation ..... E. G. Coffman, Jr., T. T. Kadota, and L. A. Shepp 235
Calls for Papers 240

Scientific American '91

The Computer for the 21st Century

Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence

by Mark Weiser

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Consider writing, perhaps the first information technology. The ability to represent spoken language symbolically for long-term storage freed information from the limits of individual memory. Today this technology is ubiquitous in industrialized countries. Not

14,966 citations

ground presence of these products of

is approachable only through complex jargon that has nothing to do with the tasks for which people use computers. The state of the art is perhaps analogous to the period when scribes had to know as much about making ink or baking clay as they did about writing.

The arcane aura that surrounds personal computers is not just a "user interface" problem. My colleagues and I at the Xerox Palo Alto Research Center think that the idea of a "personal" com- and that the , dynabooks rs" is only a chieving the on technology. Such machines cannot truly make

The idea of integrating computers seamlessly into the world at large runs counter to a number of present-day trends. "Ubiquitous computing" in this context does not mean just computers that can be carried to the beach, jungle or airport. Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box. By analogy with writing, carrying a superlaptop is like owning just one very important book. Customizing this book, even writing millions of other books, does not begin to capture the real power of literacy.

Furthermore, although ubiquitous computers may use sound and video

ICSE'99 Los Angeles, CA

Call-Mark Slicing: An Efficient and Economical Way of Reducing Slice

Akira Nishimatsu[†] **Minoru Jihira**[‡] **Shinji Kusumoto**[†] **Katsuro Inoue**[†]

[†] Graduate School of Engineering Science,
Osaka University
1-3 Machikaneyama, Toyonaka,
Osaka 560-8531, Japan
+81 6 6850 6571
{a-nisimt, kusumoto, inoue}@ics.es.osaka-u.ac.jp

[‡] Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma,
Nara 630-0101, Japan
+81 743 72 5236
minoru-j@itc.aist-nara.ac.jp

ABSTRACT

When we debug and maintain large software, it is very important to localize the scope of our concern to small program portions. Program slicing is one of promising techniques for identifying portions of interest. There are many research results on the program
static slice, which is a collection of pr
possibly affecting a particular variable
scope, but the resulting collections are often still large. A dynamic slice, which is a collection of executed pro-

55 citations

central theme of software engineering research and practice.

Various ways of analyzing large programs and extracting abstracted information of the target software have been

the difficulty of handling large
a developer's attention to specific parts of the program that are directly and indirectly related to the developer's concerns.

論　文

フォールト位置特定におけるプログラムスライスの実験的評価

西松　顯[†]　西江　圭介[†]　楠本　真二[†]　井上　克郎[†]

論　文

制限された動的情報を用いたプログラムスライシング手法の提案

高田　智規[†]　井上　克郎[††]　大畑　文明[††]　芦田　佳行[†††]

論　文

スライス計算効率化のためのプログラム依存グラフの節点集約法

大畑　文明[†]　横森　励士[†]　西松　顯[†,††]　井上　克郎[†,†††]

Node Merging Method of Program Dependence Graph for Efficient Slice Computation

Fumiaki OHATA[†], Reishi YOKOMORI[†], Akira NISHIMATSU[†,††], and Katsuro INOUE[†,†††]

あらまし　プログラム依存グラフ（PDG）とはプログラム文間の依存関係を表す有向グラフであり，プログラムスライスに利用される．これまで提案されている多くのPDG構築法は，スライスの精度向上を目標としてきた．しかし，大規模ソフトウェアにスライスを適用する場合，その精度だけでなく抽出の効率を考慮する必要がある．
本研究では，依存情報が文単位で保持される従来手法に対し，より大きな粒度で保持させることで効率向上を行ない，精度にも留意したPDG構築手法を提案する．また，実際に提案手法の評価を行い，空間コスト10－40%，時間コスト5－60%の削減を得た．

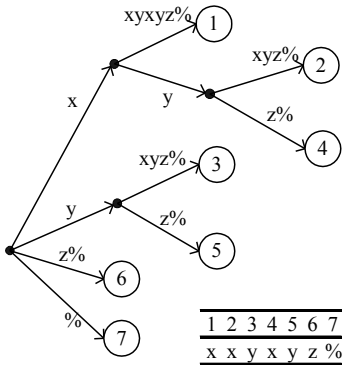
キーワード　プログラムスライス，プログラム依存グラフ，効率，精度

プログラムスライスの限界

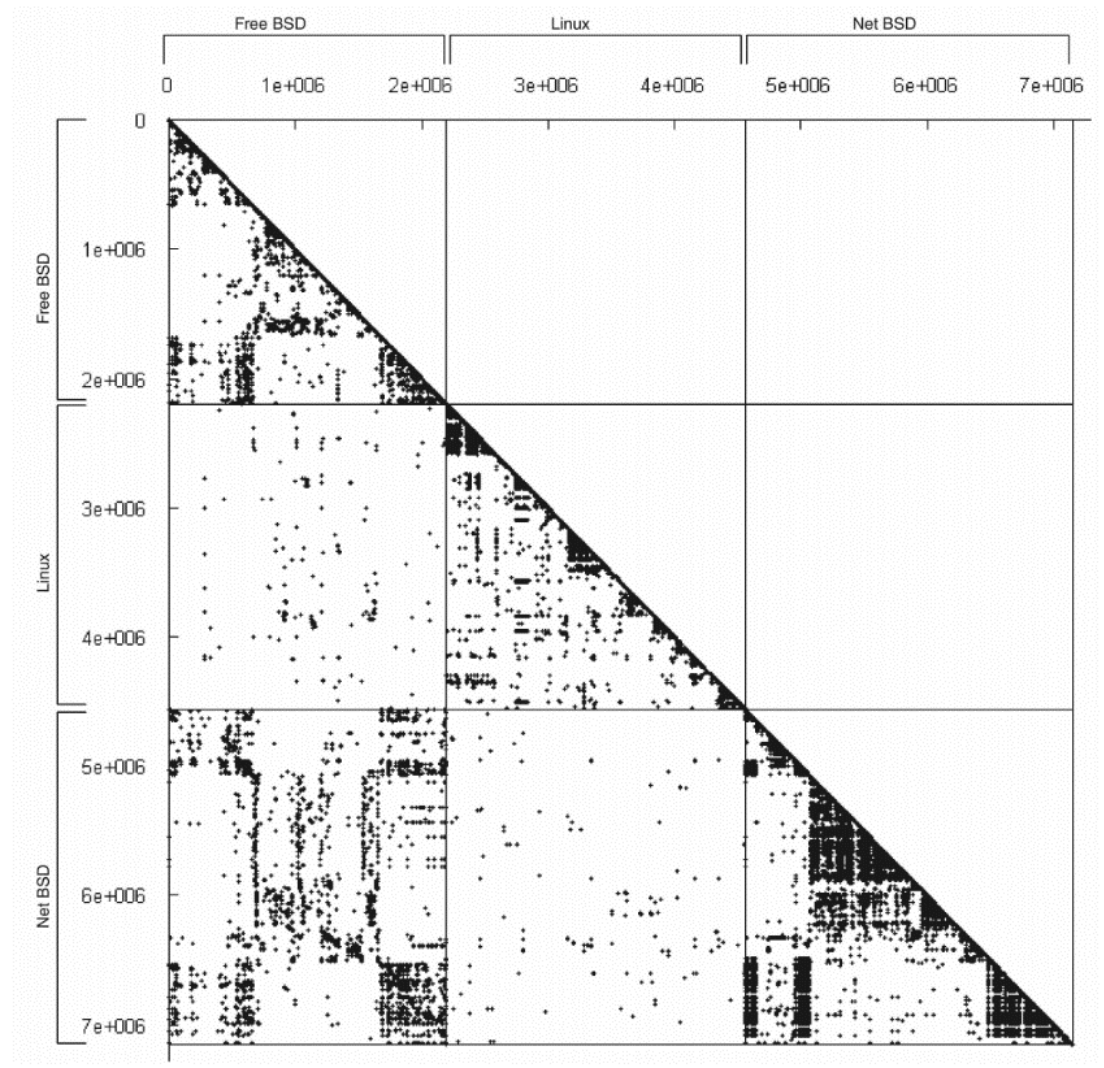
- ポインタ解析、配列解析などきれいでない世界
- スケーラビリティの限界
- 実用的なツールの開発、入手しんどい

コードクローン研究

- 1999年ごろ、企業の人との雑談がきっかけ
  - 人手による同一コードの管理の限界（数百万行、20年以上管理）
- いい方法ないか、調査
  - いくつかのクローン研究、システムはあったが、実用性は？
- 自分で作ろう！
  - 生物情報の先生にいい本とアルゴリズム教えてもらう
  - パワーあふれる学生



クローン検出ツールCCFinderの開発



そうだ、TSE、書こう！

- 2000年7月　最初の投稿
- 2001年1月　Major Revisionの要求
- 2001年8月　Minor Revisionの要求
- 2001年9月　採録通知
- 2002年7月　TSEに掲載

TSE'02

CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code

Toshihiro Kamiya, *Member*, *IEEE*, Shinji Kusumoto, *Member*, *IEEE*, and Katsuro Inoue, *Member*, *IEEE*

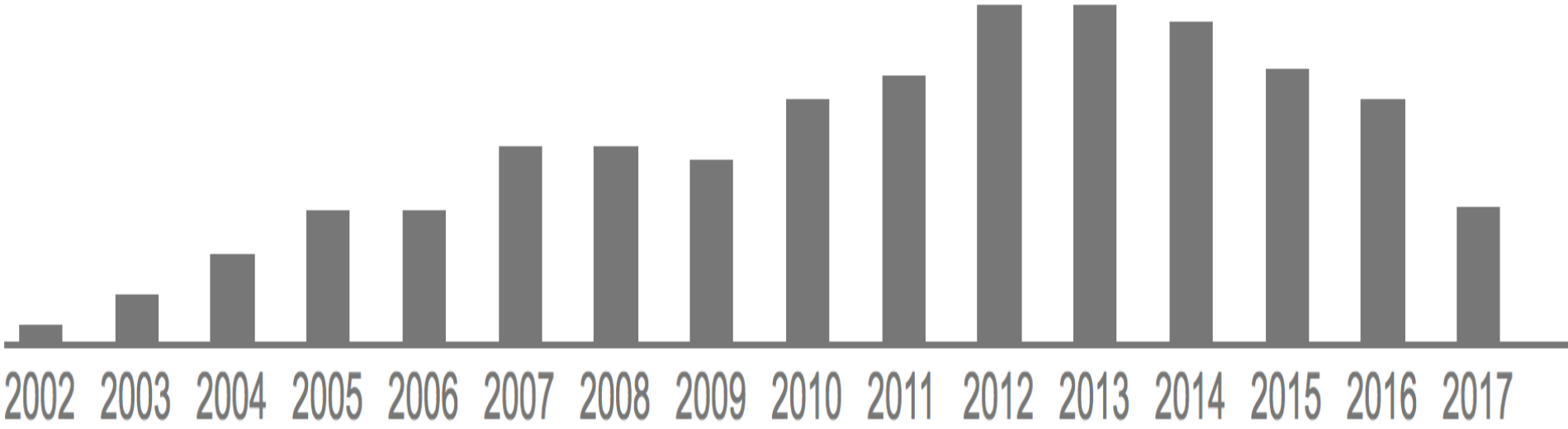
Abstract—A code clone is a code portion in source files that is identical or similar to another. Since code clones are believed to reduce the maintainability of software, several code clone detection techniques and tools have been proposed. This paper proposes a new clone detection technique, which consists of the transformation of input source text and a token-by-token comparison. For its implementation with several useful optimization techniques, we have developed a tool, named CCFinder, which extracts code clones in C, C++, Java, COBOL, and other source files. As well, metrics for the code clones have been developed. In order to evaluate the usefulness of CCFinder and metrics, we conducted several case studies where we applied the new tool to the source code of JDK, FreeBSD, NetBSD, Linux, and many other systems. As a result, CCFinder has effectively found clones and the metrics have been able to effectively identify the characteristics of the systems. In addition, we have compared the proposed technique with other clone detection techniques.

Index Terms—Code clone, duplicated code, CASE tool, metrics, maintenance.

1 Introduction

CCFinderの論文の推移

TSE　2002年7月号掲載



1,392 citations

(2017年8月29日現在)

SE分野での被参照数の高い論文100


ELSEVIER

Highly-cited papers

Vahid Garousi [a,*], João M.

[a] Software Engineering Research Group, De
[b] Department of Informatics / Centro ALGO

ARTICLE INFO

Article history:
Received 27 May 2015
Revised 11 November 2015
Accepted 11 November 2015
Available online 23 November 2015

Keywords:
Software engineering
Highly-cited papers
Top cited
Most cited
Most frequently cited
Bibliometrics

Table 6
Top-100 papers by total number of citations.

#	Title	Year	Cited by
1	A metrics suite for object oriented design	1994	1817
2	QoS-aware middleware for Web services composition	2004	1696
3	The model checker SPIN	1997	1669
4	Complexity measure	1976	1304
5	Graph drawing by force-directed placement	1991	1162
6	An intrusion-detection model	1987	1055
7	A classification and comparison framework for software architecture description languages	2000	973
8	Program slicing	1984	903
9	Uppaal in a nutshell	1997	875
10	4 + 1 view model of architecture	1995	698
11	Developing multi-agent systems: The Gaia methodology	2003	663
12	A validation of object-oriented design metrics as quality indicators	1996	661
13	Two case studies of open source software development: Apache and Mozilla	2002	635
14	Understanding code mobility	1998	627
15	Reverse engineering and design recovery: A taxonomy	1990	605
16	A formal basis for architectural connection	1997	600
17	Software risk management: Principles and practices	1991	598
18	Towards modelling and reasoning support for early-phase requirements engineering	1997	494
19	Modeling and verification of time dependent systems using time Petri nets	1991	490
20	Search-based software test data generation: A survey	2004	488
21	Preliminary guidelines for empirical research in software engineering	2002	487
22	Testing software design modeled by finite-state machines	1978	486
23	The STATEMATE semantics of Statecharts	1996	482
24	CCFinder: A multilinguistic token-based code clone detection system for large scale source code	2002	479
25	The pragmatics of model-driven development	2003	475
26	Goal-oriented requirements engineering: A guided tour	2001	470
27	DiamondTouch: A multi-user touch technology	2001	463
28	Software function, source lines of code, and development effort prediction: a software science validation	1983	463
29	FORM: A feature-oriented reuse method with domain-specific reference architectures	1998	462
30	A taxonomy and survey of grid resource management systems for distributed computing	2002	449
31	Discovering models of software processes from event-based data	1998	434
32	A critical success factors model for ERP implementation	1999	432
33	Adaptive service composition in flexible processes	2007	427
34	Empirical studies of agile software development: A systematic review	2008	423
35	Open[illegible]h visualization system and its applications to softw[illegible] engineering	2000	423
36	The AE[illegible] s[illegible] approa[illegible] [illegible] design	1997	421
37	A critic[illegible] [illegible] defe[illegible]	1999	416
38	An empirical [illegible] of speed and communication in globally distributed software development	2003	414
39	Toward reference models for requirements traceability	2001	408
40	Object-oriented metrics that predict maintainability	1993	406
41	Dynamically discovering likely program invariants to support program evolution	2001	405
42	HyTech: A model checker for hybrid systems	1997	405
43	A survey of software refactoring	2004	401
44	Recovering traceability links between code and documentation	2002	401
45	The Tame project: Towards improvement-oriented software environments	1988	401
46	Patterns in property specifications for finite-state verification	1999	393
47	System structure for software fault tolerance	1975	393
48	Guidelines for conducting and reporting case study research in software engineering	2009	392
49	Prioritizing test cases for regression testing	2001	390
50	Estimating software project effort using analogies	1997	390

Top Author ?

Table 7
Top-100 papers by average annual number of citations.

#	Title	Year	Cited by	Annual average	# in the other
1	QoS-aware middleware for Web services composition	2004	1696	154.2	2
2	CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms	2011	371	92.8	56
3	The model checker SPIN	1997	1669	92.7	3
4	A metrics suite for object oriented design	1994	1817	86.5	1
5	Guidelines for conducting and reporting case study research in software engineering	2009	392	65.3	48
6	A classification and comparison framework for software architecture description languages	2000	973	64.9	7
7	Empirical studies of agile software development: A systematic review	2008	423	60.4	34
8	Developing multi-agent systems: The Gaia methodology	2003	663	55.3	11
9	Adaptive service composition in flexible processes	2007	427	53.4	33
10	Two case studies of open source software development: Apache and Mozilla	2002	635	48.8	13
11	Uppaal in a nutshell	1997	875	48.6	9
12	Graph drawing by force-directed placement	1991	1162	48.4	5
13	Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems	2007	387	48.4	52
14	KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera	2011	181	45.3	-
15	Search-based software test data generation: A survey	2004	488	44.4	20
16	Data mining static code attributes to learn defect predictors	2007	329	41.1	71
17	The pragmatics of model-driven development	2003	475	39.6	25
18	A systematic review of software development cost estimation studies	2007	303	37.9	87
19	An intrusion-detection model	1987	1055	37.7	6
20	Preliminary guidelines for empirical research in software engineering	2002	487	37.5	21
21	Understanding code mobility	1998	627	36.9	14
22	CCFinder: A multilinguistic token-based code clone detection system for large scale source code	2002	479	36.8	24
23	A survey of software refactoring	2004	401	36.5	43
24	Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact	2005	354	35.4	62
25	X10: An object-oriented approach to Non-Uniform Cluster Computing	2005	351	35.1	64
26	4 + 1 view model of architecture	1995	698	34.9	10
27	A validation of object-oriented design metrics as quality indicators	1996	661	34.8	12
28	A taxonomy and survey of grid resource management systems for distributed computing	2002	449	34.5	30
29	An empirical study of speed and communication in globally distributed software development	2003	414	34.5	38
30	Model-based performance prediction in software development: A survey	2004	379	34.5	54
31	The physics of notations: Toward a scientific basis for constructing visual notations in software engineering	2009	204	34.0	–
32	Goal-oriented requirements engineering: A guided tour	2001	470	33.6	26
33	A complexity measure	1976	1304	33.4	4
34	A formal basis for architectural connection	1997	600	33.3	16
35	DiamondTouch: A multi-user touch technology	2001	463	33.1	27
36	The FRACTAL component model and its support in Java	2006	295	32.8	97
37	The Palladio component model for model-driven performance prediction	2009	196	32.7	–
38	On the unification power of models	2005	322	32.2	78
39	Recovering traceability links between code and documentation	2002	401	30.8	44
40	Systematic literature reviews in software engineering - A systematic literature review	2009	185	30.8	–
41	Empirical validation of object-oriented metrics on open source software for fault prediction	2005	299	29.9	92
42	An analysis and survey of the development of mutation testing	2011	119	29.8	–
43	Regression testing minimization, selection and prioritization: A survey	2012	89	29.7	–
44	Model checking programs	2003	352	29.3	63
45	Toward reference models for requirements traceability	2001	408	29.1	39
46	Program slicing	1984	903	29.1	8
47	Eliciting security requirements with misuse cases	2005	290	29.0	100
48	Dynamically discovering likely program invariants to support program evolution	2001	405	28.9	41
49	Benchmarking classification models for software defect prediction: A proposed framework and novel findings	2008	202	28.9	–

Total Number of Citations

Average Annual Number of Citations

コードクローン分析技術の普及活動

- 2002年から2007年にかけて企業向けコードクローン分析技術の解説と演習を行うセミナー開設
- 企業での活用例などの紹介
- 技術の普及、ツールCCFinderの知名度向上に貢献
  （論文の被引用数にも...）
- 多くの企業との共同研究。国際共同研究も(Samsung, MS Research -> Xiao, Visual Studioのクローン分析機能)

Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder

Simone Livieri† Yoshiki Higo† Makoto Matushita† Katsuro Inoue†

†Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
E-mail: {simone, y-higo, matusita, inoue}@ist.osaka-u.ac.jp

Abstract

The increasing performance-price ratio of computer hardware makes possible to
at code clone analysis. Th
a distributed approach at large-scale code clone analysis.

can analyze, in the ideal case, up to 5.2 million of lines of C code in about 18 minutes on a PC-based workstation (Intel Xeon 2.8GHz CPU with 2 GB memory).

paper, we have chosen, as the analysis target,
on of open source software used for FreeBSD
(hereinafter called *"the FreeBSD target"*), which consists

153 citations

- コードクローンの論文いっぱい書いた、満足！
- やり尽くした感。残ったテーマはコスパが悪い…
- 次はコード検索だ！
  - コードクローンの研究の延長線
  - GoogleのPage Rankの手法に感化、ソフトウェアの部品の重要度も同じように評価

コード検索ツールSPARS-J

ICSE'03

Component Rank: Relative Significance Rank for Software Component Search

Katsuro Inoue †, Reishi Yokomori †, Hikaru Fujiwara†,
Tetsuo Yamamoto ††, Makoto Matsushita † and Shinji Kusumoto †
† Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
†† Japan Science and Technology Corporation, 4-1-8, Honmachi,
Kawaguchi, Saitama 332-8531, Japan
{inoue, yokomori, t-yamamt, matsusita, kusumoto}@ist.osaka-u.ac.jp

Abstract

Collections of already developed
tant resources for efficient developm
systems. In this paper, we propose a
ing software components, called C
on analyzing actual use relations
are ranked high. Using the Comp

128 citations

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 31, NO. 3, MARCH 2005

TSE'05

Ranking Significance of Software Compo
Based on Use Relations

Katsuro Inoue, *Member, IEEE*, Reishi Yokomori, *Member, IEEE*, Tetsuo Yamamoto, *Me*
Makoto Matsushita, and Shinji Kusumoto, *Member, IEEE*

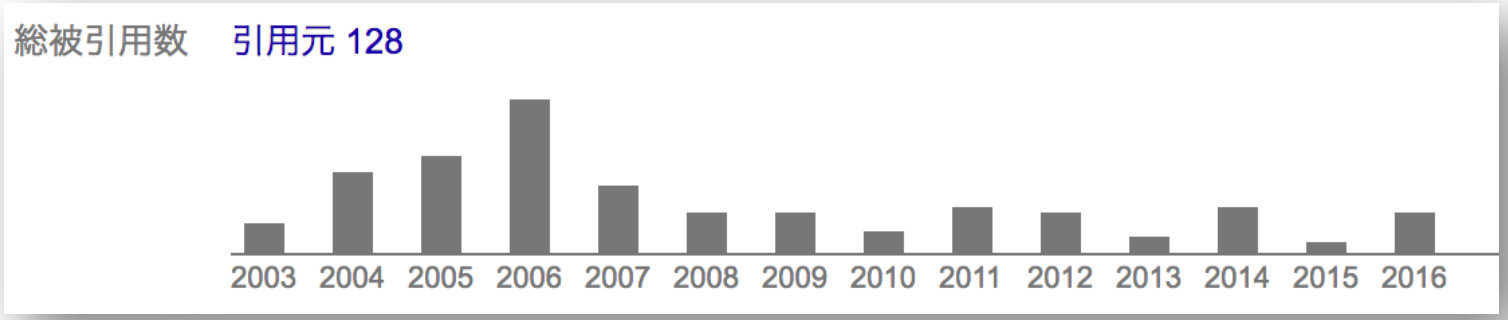
Abstract—Collections of already developed programs are important resources for efficient development of reliable softwa
n this paper, we propose a novel graph-representation model of a software component library (repository), called *comp*
model. This is based on analyzing actual usage relations of the components and propagating the significance through th
relations. Using the component ra PARS-J and applie
to various collections of Java files that are used more
As a result, software engineers l ARS-J has been u
companies, and has produced pr

182 citations

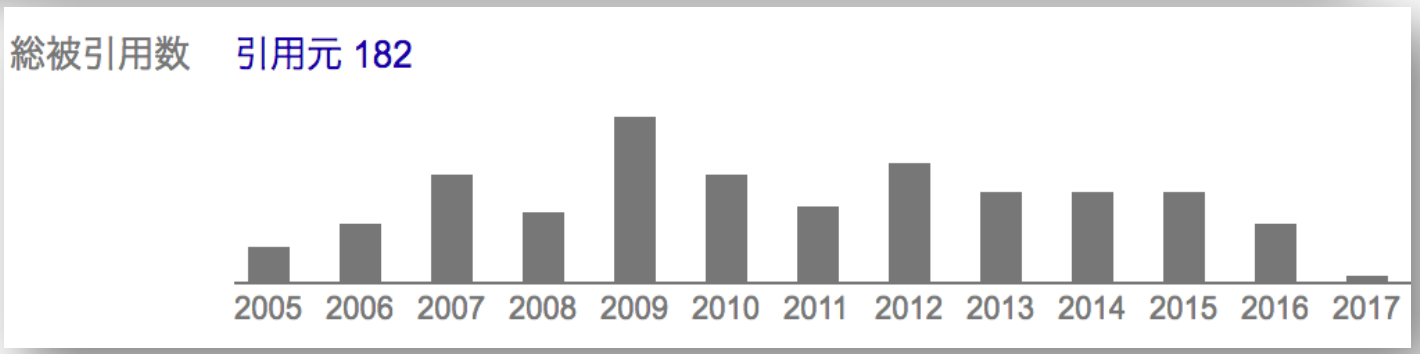
Index Terms—Component rank, graph representation model, reuse models, program analysis, reusable libraries.

1 INTRODUCTION

SPARS-J論文の被引用数の推移



ICSE'03



TSE'05

クローンと検索の2つの研究

クローンの研究

- 1999年～
- 会社のニーズが出発点
- 方式の提案、実装
- ツールCCFinder
- TSE, ICSE, MSR, ...
  (総計>1,500参照)
- ツールの配布、保守
- 活発な応用(自分、他人も)

検索の研究

- 2003年～
- 自分の発想が出発点
- モデルの提案、実装
- ツールSPARS-J
- ICSE, TSE, ...
  (総計>300参照)
- いくつかのテスト運用、Webサービス
- 他への技術転移ほぼなし

コード検索をきっかけにMining Software Repositoriesへ

1st International Workshop on Mining Software Repositories MSR 2004, Edinburg, Scotland, U.K., 2004.

Organizers

- Ahmed E. Hassan
- Richard C. Holt
- Audris Mockus

Program Committee

- Harald Gall
- Les Gasser
- Daniel German
- James Herbsleb
- Katsuro Inoue
- Philip Johnson
- Dewayne Perry
- Andreas Zeller

計測・検索　評価・分析　フィードバック

対象

広

target

既存のソフトウェア工学技術

狭

対象

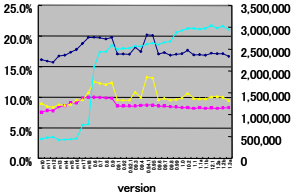
プロジェクト群（企業内全資産、全オープンソース等）

目的

パターンや部品、知見の抽出、利益予測

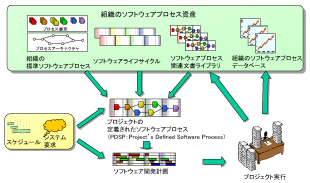
形態

部品共通化、リファレンスモデル・標準化

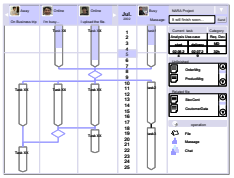


Mega Software Engineering

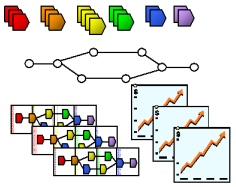
単一プロジェクト



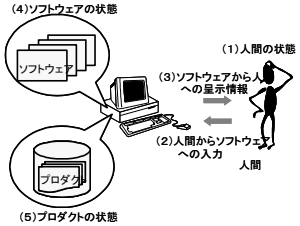
進捗把握、コスト管理



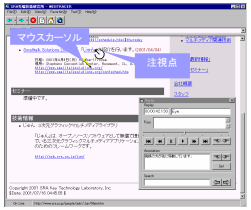
プロセス改善、資産再利用



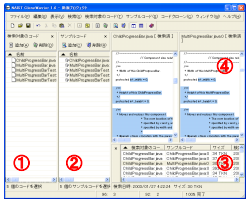
各開発者の作業や生産物



ユーザビリティ・問題把握



ヘルプ、ツール・ガイドライン



ネット上のコード検索

自前のレポジトリの限界

ネット空間にあるソフトウェア資産の活用

ICSE'12 Zurich, Switzerland

Where Does This Code Come from and Where Does It Go?
- Integrated Code History Tracker for Open Source Systems -

Katsuro Inoue, Yusuke Sasaki, Pei Xia, and Yuki Manabe
Osaka University
Osaka, Japan
{*inoue, peixia, y-manabe*}*@ist.osaka-u.ac.jp*

***Abstract*—When we reuse a code fragment in an open source system, it is very important to know the history of the code, such as the code origin and evolution. In this paper, we propose an integrated approach to code history tracking for open source repositories. This approach takes a query code fragment as its input, and returns the code fragments containing the code clones with the query code. It utilizes publicly available code search engines as external resources.** **have designed and implemented a** ***Ichi Tracker*. Using Ichi Tracker, we** **studies. These case studies show the**

19 citations

Current software engineering tools do not provide sufficient support to explore code history. To know the code origin, we have to specify project names and/or URLs. Also, to know the code evolution, we have to understand the interrelations of open source projects.

Code search engines such as Google Code Search [10] ery useful tools to explore open source rigin and evolution of code. However, engines only allows to get keywords

Google "code clone"

Scholar 約 3,440 件 (0.13 秒)

記事
マイ ライブラリ

期間指定なし
2017 年以降
2016 年以降
2013 年以降
期間を指定...

関連性で並べ替え
日付順に並べ替え

すべての言語
英語 と 日本語のページを検索

特許を含める
引用部分を含める

アラートを作成

CCFinder: a multilinguistic token-based **code clone** detection system for large scale source code
T Kamiya, S Kusumoto, K Inoue - IEEE Transactions on ..., 2002 - ieeexplore.ieee.org
Abstract: A **code clone** is a code portion in source files that is identical or similar to another. Since code clones are believed to reduce the maintainability of software, several **code clone** detection techniques and tools have been proposed. This paper proposes a new clone
引用元 1378 関連記事 全 14 バージョン Web of Science: 397 引用 保存済み

[PDF] osaka-u.ac.jp
Find it @Osaka University

[HTML] Comparison and evaluation of **code clone** detection techniques and tools: A qualitative approach
CK Roy, JR Cordy,
Over the last decad
proposed. In this pa
state-of-the-art in cl
引用元 630 関連記

[HTML] sciencedirect.com

An empirical st
M Kim, V Sazawal,
ABSTRACT It has b
eliminating clones b
the validity of this as
引用元 507 関連記

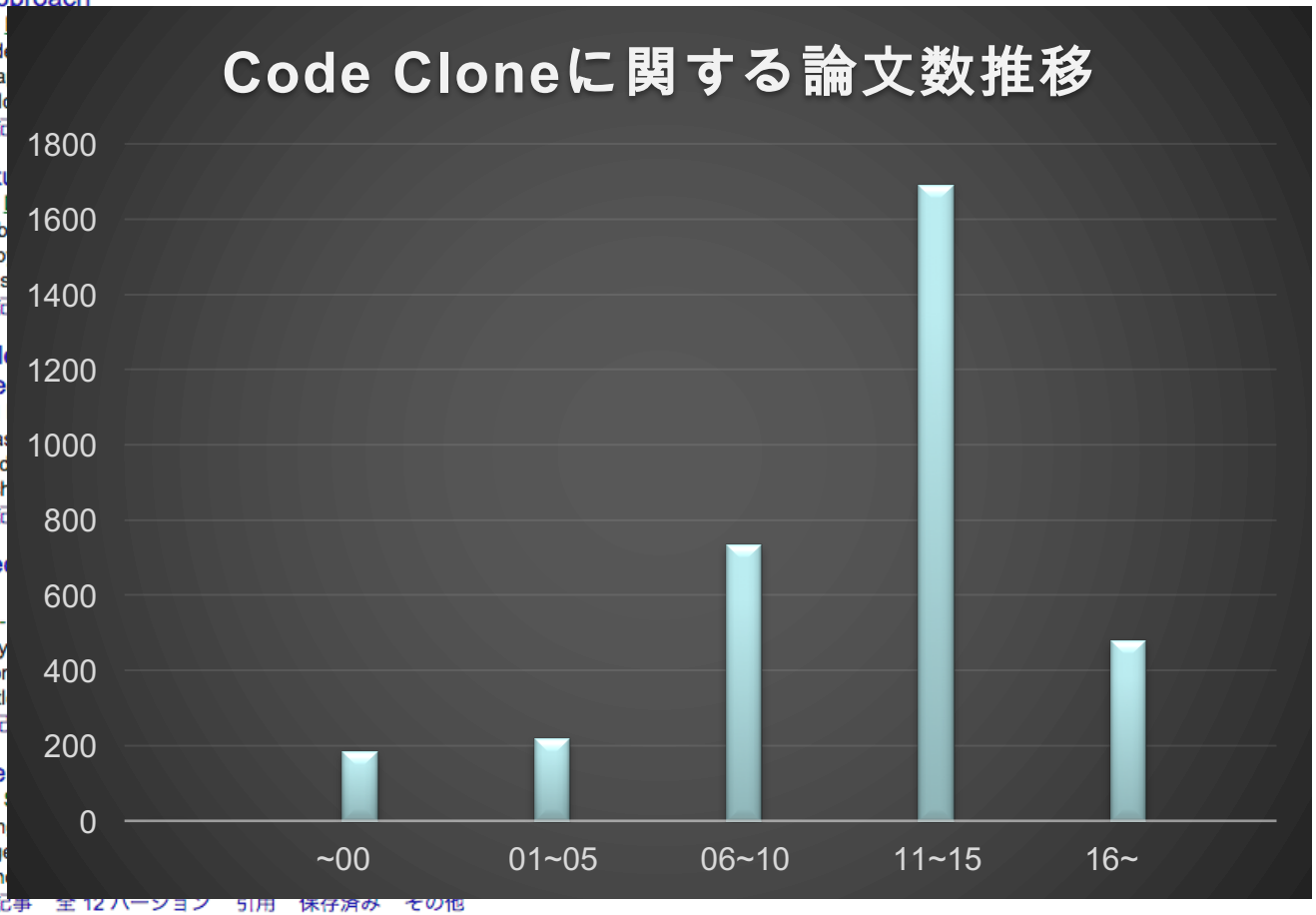
Very-large scal
using distribute
S Livieri, Y Higo, M
Abstract The increas
explore a distributed
distributed approach
引用元 153 関連記

A mutation/inje
detection tools
CK Roy, JR Cordy -
Abstract: In recent y
proposed. While sor
these tools, very littl
引用元 115 関連記

Gemini: Mainte
Y Ueda, T Kamiya,
Abstract: Maintainin
scale becomes large
software maintenan
引用元 102 関連記事 全 12 バージョン 引用 保存済み その他

Index-based **code clone** detection: incremental, distributed, scalable
B Hummel, E Juergens, L Heinemann, ... - ... (ICSM), 2010 IEEE ..., 2010 - ieeexplore.ieee.org

[PDF] semanticscholar.org



ペンペン草も生えないまでに…

Multi-Project Software Engineering

Katsuro Inoue [†,*], Pankaj K. Garg [‡],
Hajimu Iida[††], Kenichi Matsumoto [††,*], Koji Torii [††,*]
[†] Graduate School of Information Science and Technology, Osaka University
1[illegible]yama, Toyonaka, Osak[illegible] Jap[illegible]
[illegible], 168[illegible] Nigh[illegible]ngale Avenue, Suite 201, [illegible], CA 94[illegible]7, USA
[††] Nara Institute of Scien[illegible] and Technology, Nara 630-0192, Japan
[*] EASE (Empirical Approach t[illegible] [illegible]ware Engineering) Project, Senri, Osaka, [illegible]
in[illegible]osaka-u.ac.jp, garg@zeesource.net
[illegible]a, matumoto, torii}@is.aist-nara.ac.jp

ICSE'04 Rejected

FSE'04 Rejected

Abstract

In various fields of computer science, rapidly growing hardware power, such as high-speed network, high-performance CPU, huge disk capacity, and large memory space, has been fruitfully harnessed. Examples of such usage are large scale data and web mining, grid computing, and multimedia environments. We propose that such rich hardware can also catapult software engineering to the next level. Huge amounts of software engineering data can be systematically collected and organized from tens of thousands of projects inside organizations, or from outside an organization through the Internet. The collected data

useful for software engineering. A unique feature of software products, however, is that the end product has virtually no physical manifestation. Hence, composing or taking apart a software product has virtually no cost implications. As a result, software component reuse is a common practice for code sharing among multiple projects.

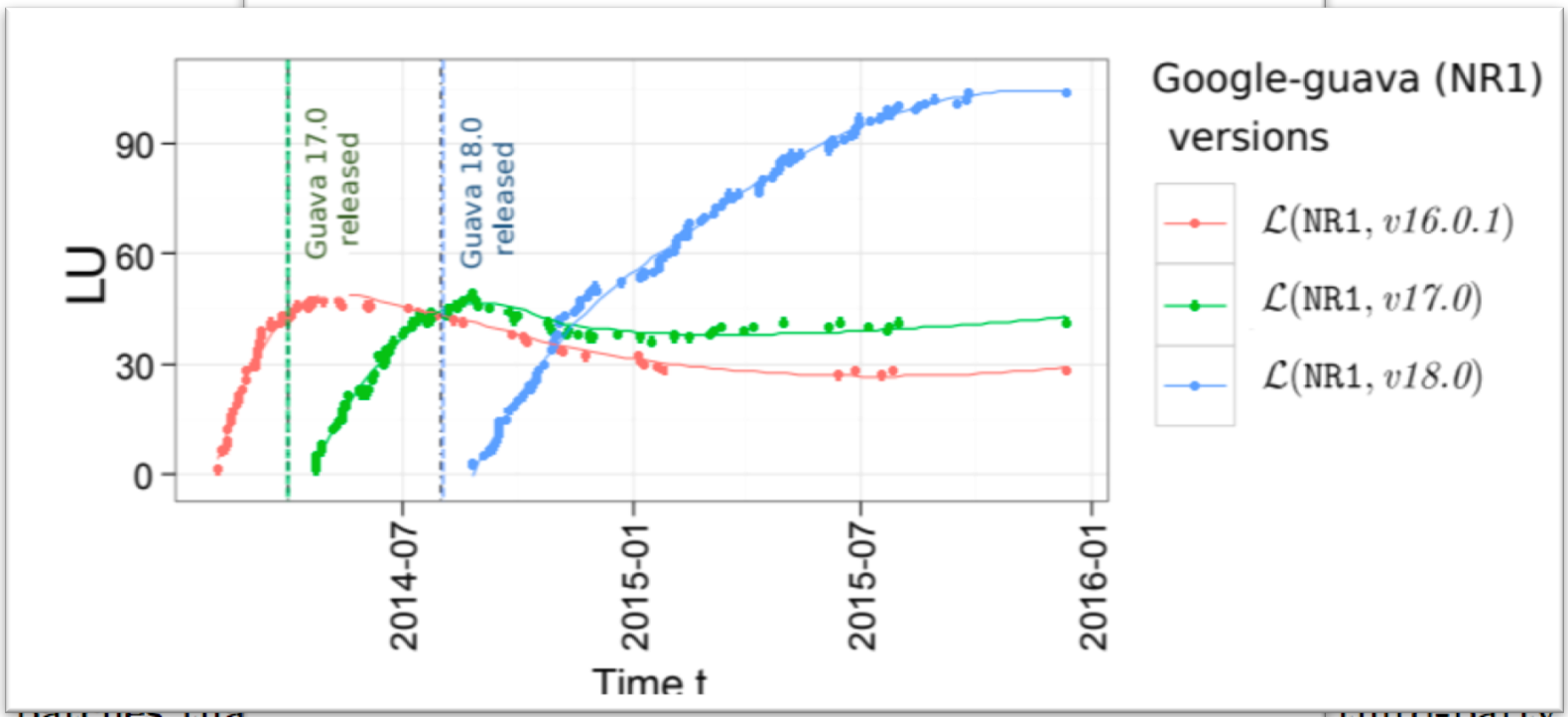
We posit that "sharing" among software projects can be extended beyond code or component sharing to more and varied kinds of "knowledge" sharing. Such sharing can be achieved using what we call *multi-project software engineering*. Instead of narrowly engineering a product, or a product family, an organization can undertake the responsibility and benefits of engineering a large number of projects

FSE'14 TSE'14 ICSE'15 J. EMSE'17

Rejected Rejected Rejected Accepted

Do Developers Update Their Library Dependencies?

An Empirical Study on the Impact of Security Advisories on Library Migration



A sentence-matching method for automatic license identification of source code files

ICSE 2010 Paper Notification [30] 受信トレイ x

Prem Devanbu and Sebastian Uchitel <icse2010-papers-chairs@borbala.com>
To dmg, y-manabe, inoue, icse2010-paper.

Dear Daniel, Yuki and Katsuro

Thanks for your submission to ICSE 2010. W regret t have to inform you that your paper,

"A sentence-matching method for automatic license identification of source code files"

ha not been accepted for inclusion in the conference program. The

AB
The
nen
leng
has
cen
of i
leng
is n
algo
bili
We
forr
spe
source code files of Debian that highlight interesting facts about the manner in which licenses are used by FOSS.

1. INTRODUCTION

Free and Open Source Software (FOSS) has become an important source of reusable code [19]. To be able to reuse a FOSS component an application (proprietary or FOSS) should satisfy all the requirements and conditions that the

One of the major challenges of intellectual property clearance is to identify the license under which a FOSS component, and each of its files, is made available. This is due to several factors: 1) there is a vast number of open source licenses, some approved by the Open Source Initiative (currently 65), and many more that are not—Table 1 shows some frequent FOSS licenses and their abbreviations, as used in this paper; 2) a FOSS product might be made available under several licenses, 3) different versions of a FOSS com-

イチロー	衣笠
0.323	0.270

$$0.269 < \text{真の打率} < 0.474$$

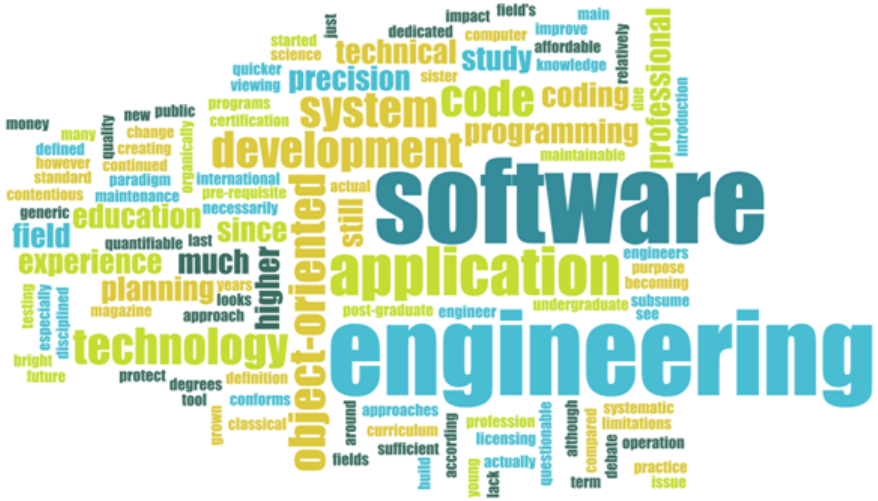
RQ1： 光るソフトウェア工学研究とは？

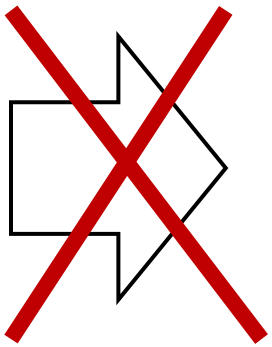
光る研究？

・当然、人によって考えはいろいろ

- お金
- 昇進
- 論文
- 産学連携

SE研究は金が儲かるか？





お金

Computer Science > Digital Libraries

Publish or impoverish: An investigation of the monetary reward system of science in China (1999–2016)

Table 5 Comparison of Average Amount of Cash Awards* for a Paper Published in Selected Journals (2008-2016)

	2008	2009	2010	2011	2012	2013	2014	2015	2016
Nature, Science	$26,212	$26,006	$25,781	$25,365	$33,990	$36,658	$38,908	$43,783	$43,783
PNAS	$3,156	$3,025	$3,353	$3,443	$3,664	$3,619	$3,751	$3,513	$3,513
PLOS One	$1,096	$1,086	$1,035	$994	$991	$915	$941	$984	$984
MIS Quarterly	$2,613	$2,570	$2,553	$2,654	$2,876	$2,861	$2,992	$2,938	$2,938
JASIST	$1,737	$1,758	$1,741	$1,887	$2,066	$2,303	$2,435	$2,488	$2,488
Journal of Documentation	$1,082	$1,087	$			265	$1,329	$1,408	$1,408
Library Hi Tech	$781	$775				768	$795	$783	$783
LIBRI	$650	$644				509	$517	$484	$484

* All the amounts are full amount (in USD)

1. $100？
2. $1,000？
3. $10,000？
4. それ以上？

ヒント：教授の平均年収$8,600

Submission history

From: Fei Shu [view email]
[v1] Tue, 4 Jul 2017 21:46:35 GMT (1131kb)

SE系のベンチャーの例

Gail Murphy

2007年設立

Prof. of UBC　&　Chief Scientist and Co-Founder of Tasktop

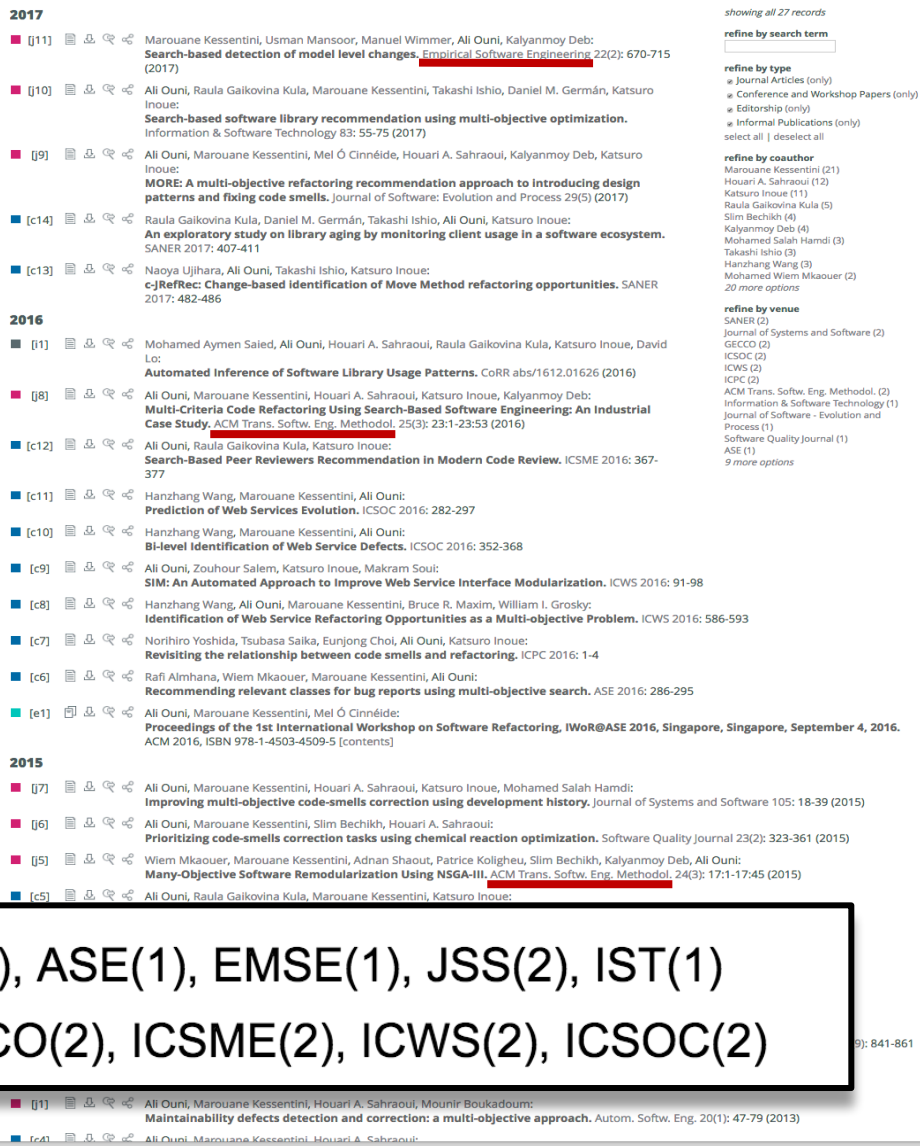
SE研究で昇進できるか？

- 情報科学分野の中でも地味な分野
  - インターネット、人工知能など、明らかなブームはなし
- しかし無くならない分野
  - 企業で必要とされている
  - 学生の多くは、SEとして就職
  - 教育を考えれば無くせないはず
- 欧米では積極的な移動でプロモーション獲得

A Case

アリ オウニ（チュニジア出身、SBSE分野）

- 2014年11月 モントリオール大修了
- 2014年12月 阪大、特任助教
- 2016年8月 Assistant Prof. UAE Univ.
- 2017年8月 Assistant Prof. Univ. Quebec in Montreal



2017

[j11] Marouane Kessentini, Usman Mansoor, Manuel Wimmer, **Ali Ouni**, Kalyanmoy Deb: **Search-based detection of model level changes.** Empirical Software Engineering 22(2): 670-715 (2017)

[j10] **Ali Ouni**, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M. Germán, Katsuro Inoue: **Search-based software library recommendation using multi-objective optimization.** Information & Software Technology 83: 55-75 (2017)

[j9] **Ali Ouni**, Marouane Kessentini, Mel Ó Cinnéide, Houari A. Sahraoui, Kalyanmoy Deb, Katsuro Inoue: **MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells.** Journal of Software: Evolution and Process 29(5) (2017)

[c14] Raula Gaikovina Kula, Daniel M. Germán, Takashi Ishio, **Ali Ouni**, Katsuro Inoue: **An exploratory study on library aging by monitoring client usage in a software ecosystem.** SANER 2017: 407-411

[c13] Naoya Ujihara, **Ali Ouni**, Takashi Ishio, Katsuro Inoue: **c-JRefRec: Change-based identification of Move Method refactoring opportunities.** SANER 2017: 482-486

2016

[i1] Mohamed Aymen Saied, **Ali Ouni**, Houari A. Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, David Lo: **Automated Inference of Software Library Usage Patterns.** CoRR abs/1612.01626 (2016)

[j8] **Ali Ouni**, Marouane Kessentini, Houari A. Sahraoui, Katsuro Inoue, Kalyanmoy Deb: **Multi-Criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study.** ACM Trans. Softw. Eng. Methodol. 25(3): 23:1-23:53 (2016)

[c12] **Ali Ouni**, Raula Gaikovina Kula, Katsuro Inoue: **Search-Based Peer Reviewers Recommendation in Modern Code Review.** ICSME 2016: 367-377

[c11] Hanzhang Wang, Marouane Kessentini, **Ali Ouni**: **Prediction of Web Services Evolution.** ICSOC 2016: 282-297

[c10] Hanzhang Wang, Marouane Kessentini, **Ali Ouni**: **Bi-level Identification of Web Service Defects.** ICSOC 2016: 352-368

[c9] **Ali Ouni**, Zouhour Salem, Katsuro Inoue, Makram Soui: **SIM: An Automated Approach to Improve Web Service Interface Modularization.** ICWS 2016: 91-98

[c8] Hanzhang Wang, **Ali Ouni**, Marouane Kessentini, Bruce R. Maxim, William I. Grosky: **Identification of Web Service Refactoring Opportunities as a Multi-objective Problem.** ICWS 2016: 586-593

[c7] Norihiro Yoshida, Tsubasa Saika, Eunjong Choi, **Ali Ouni**, Katsuro Inoue: **Revisiting the relationship between code smells and refactoring.** ICPC 2016: 1-4

[c6] Rafi Almhana, Wiem Mkaouer, Marouane Kessentini, **Ali Ouni**: **Recommending relevant classes for bug reports using multi-objective search.** ASE 2016: 286-295

[e1] **Ali Ouni**, Marouane Kessentini, Mel Ó Cinnéide: **Proceedings of the 1st International Workshop on Software Refactoring, IWoR@ASE 2016, Singapore, Singapore, September 4, 2016.** ACM 2016, ISBN 978-1-4503-4509-5 [contents]

2015

[j7] **Ali Ouni**, Marouane Kessentini, Houari A. Sahraoui, Katsuro Inoue, Mohamed Salah Hamdi: **Improving multi-objective code-smells correction using development history.** Journal of Systems and Software 105: 18-39 (2015)

[j6] **Ali Ouni**, Marouane Kessentini, Slim Bechikh, Houari A. Sahraoui: **Prioritizing code-smells correction tasks using chemical reaction optimization.** Software Quality Journal 23(2): 323-361 (2015)

[j5] Wiem Mkaouer, Marouane Kessentini, Adnan Shaout, Patrice Koligheu, Slim Bechikh, Kalyanmoy Deb, **Ali Ouni**: **Many-Objective Software Remodularization Using NSGA-III.** ACM Trans. Softw. Eng. Methodol. 24(3): 17:1-17:45 (2015)

[c5] **Ali Ouni**, Raula Gaikovina Kula, Marouane Kessentini, Katsuro Inoue:

[j1] **Ali Ouni**, Marouane Kessentini, Houari A. Sahraoui, Mounir Boukadoum: **Maintainability defects detection and correction: a multi-objective approach.** Autom. Softw. Eng. 20(1): 47-79 (2013)

showing all 27 records

refine by search term

refine by type
- Journal Articles (only)
- Conference and Workshop Papers (only)
- Editorship (only)
- Informal Publications (only)

select all | deselect all

refine by coauthor
Marouane Kessentini (21)
Houari A. Sahraoui (12)
Katsuro Inoue (11)
Raula Gaikovina Kula (5)
Slim Bechikh (4)
Kalyanmoy Deb (4)
Mohamed Salah Hamdi (3)
Takashi Ishio (3)
Hanzhang Wang (3)
Mohamed Wiem Mkaouer (2)
20 more options

refine by venue
SANER (2)
Journal of Systems and Software (2)
GECCO (2)
ICSOC (2)
ICWS (2)
ICPC (2)
ACM Trans. Softw. Eng. Methodol. (2)
Information & Software Technology (1)
Journal of Software - Evolution and Process (1)
Software Quality Journal (1)
ASE (1)
9 more options

- **Journals:** TSE(1), TOSEM(2), TSC(1), ASE(1), EMSE(1), JSS(2), IST(1)
- **Conferences:** ASE(1), FSE(1), GECCO(2), ICSME(2), ICWS(2), ICSOC(2)

論文、論文、論文

- 大学や研究機関にとっては論文の発表は必須
  - 近年より良い場所に多数の論文発表を求められるプレッシャー増大
  - 昔はのんびりしてた…

- 良い場所
  ICSE > FSE > ASE > … > ASPEC > … ？
  TSE > TOSEM > EMSE > … > JSS > … ？
  Tire 1 : TSE, ICSE, …　Tire 2: … ？

- 論文数
  - 論文数 vs. 被引用数
  - H-index ？
    H-indexがxの人：x回以上参照されている論文がx本以上ある

Top Publications（English）

Categories > Engineering & Computer Science > **Software Systems**

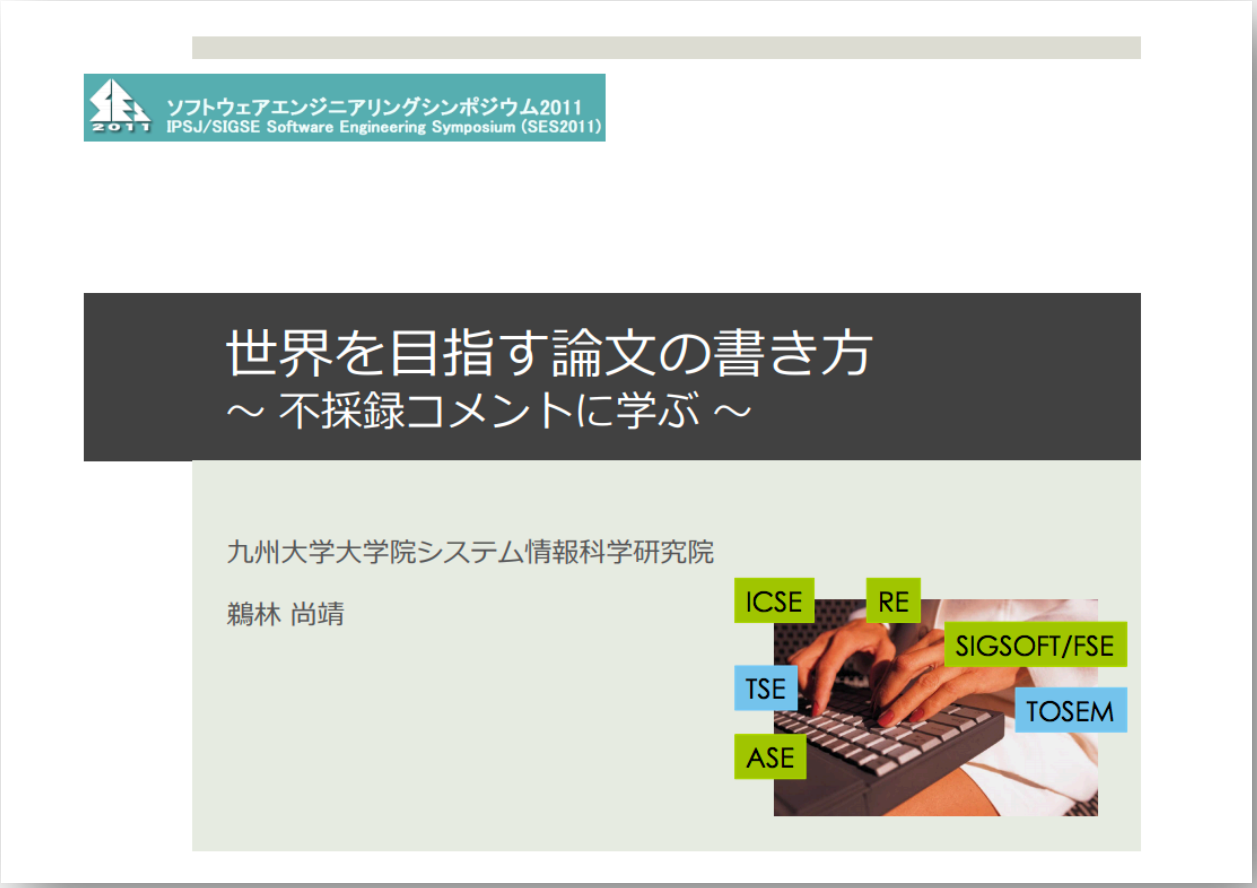
	Publication	h5-index	h5-median
1.	International Conference on Software Engineering	68	91
2.	IEEE Transactions on Software Engineering	52	80
3.	Journal of Systems and Software	51	73
4.	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	50	67
5.	ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	46	68
6.	Information and Software Technology	45	68
7.	ACM SIGSOFT International Symposium on Foundations of Software Engineering	43	64
8.	Mining Software Repositories	39	57
9.	IEEE Software	38	54
10.	ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)	37	55
11.	ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)	37	51
12.	Empirical Software Engineering	37	48
13.	International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	36	55
14.	arXiv Software Engineering (cs.SE)	36	50
15.	International Symposium on Software Testing and Analysis	31	50
16.	IEEE/ACM International Conference on Automated Software Engineering (ASE)	31	44
17.	Software & Systems Modeling	31	43
18.	IEEE International Conference on Software Maintenance	29	41
19.	IEEE International Conference on Software Testing, Verification and Validation (ICST)	29	41
20.	arXiv Programming Languages (cs.PL)	29	40

Top Publications（日本語）

Japanese

	Publication	h5-index	h5-median
1.	土木学会論文集 B2 (海岸工学)	14	16
2.	電気学会論文誌 B (電力・エネルギー部門誌)	12	15
3.	電子情報通信学会技術研究報告	11	16
4.	情報処理学会論文誌	11	15
5.	日本建築学会構造系論文集	10	12
18.	電子情報通信学会論文誌 D	8	9
19.	電気学会論文誌 D (産業応用部門誌)	8	9
20.	理学療法科学	8	8
21.	国際 P2M 学会誌	7	15
60.	コンピュータ ソフトウェア	6	7
61.	人工知能学会全国大会論文集	6	7
62.	保全生態学研究	6	7
63.	分析化学	6	7
64.	土木学会論文集 G (環境)	6	7
65.	地域学研究	6	7
66.	地盤工学ジャーナル	6	7
67.	情報処理学会論文誌 論文誌ジャーナル	6	7

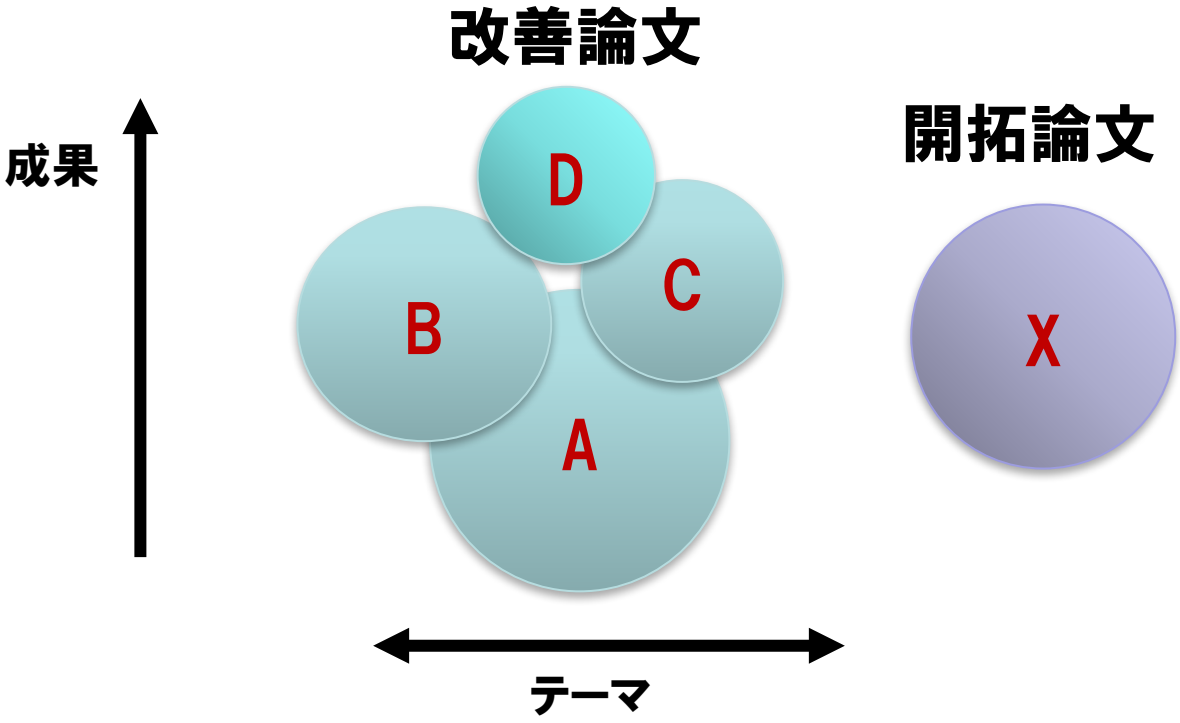
よい論文を書くには！



- **鵜林先生のSES-2011のチュートリアル**
- **良い論文書くのに必要なこと、全部述べられている！**

ちょっとだけおせっかいを

- いろいろなPCやエディターやってきた
  - ICSE (8), FSE/ESEC (3), MSR (9), ...
  - TSE Associate Editor, J. EMSE Editorial Board, ...
- 通りやすい論文の傾向



研究の目的とは...

?

ツール開発、OSS展開、産学連携

－しっかりした開発には手間暇かかる

－研究的な新規性、有用性がなかなか言いにくい

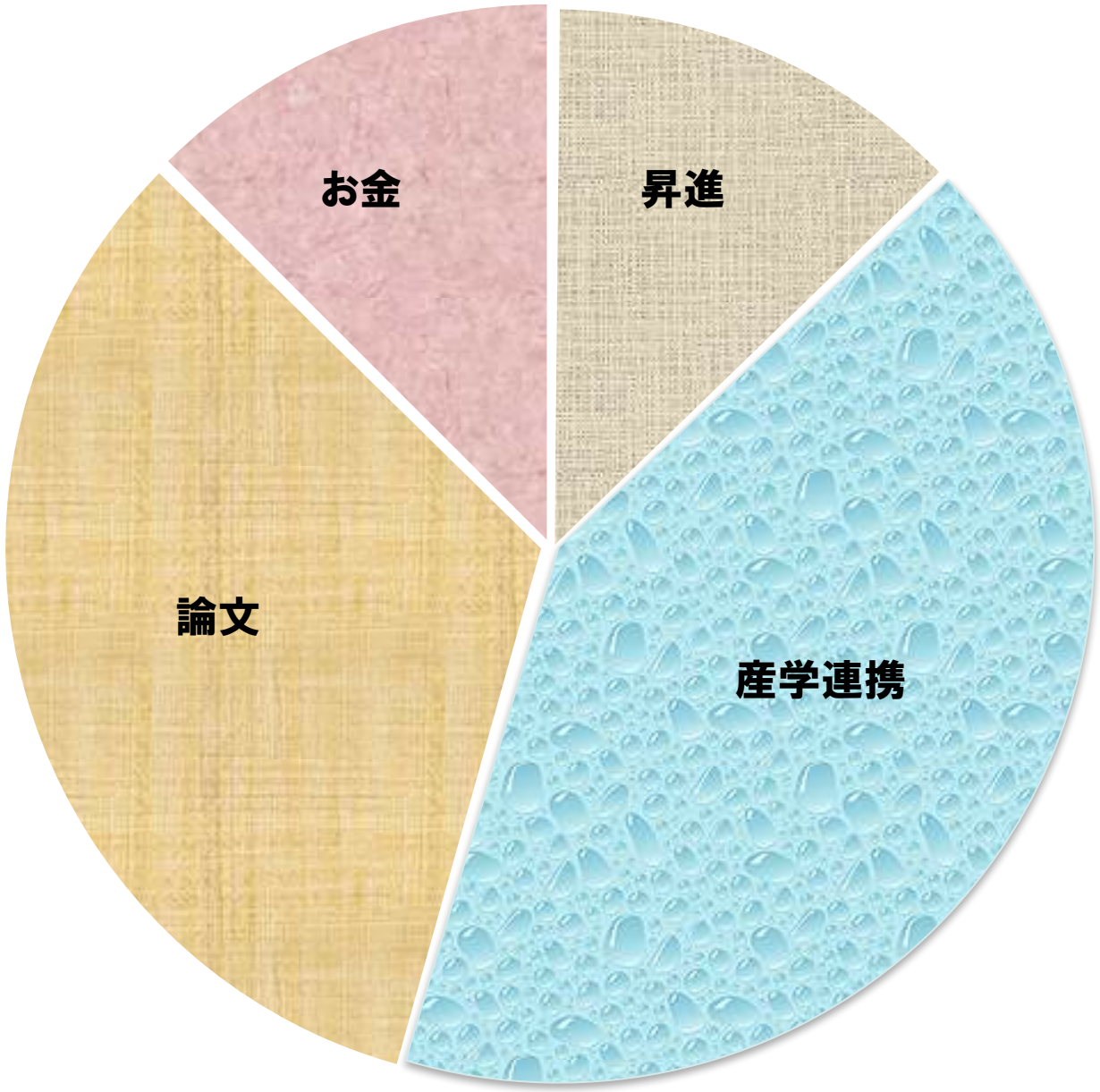
－使ってもらえるまでには、コミュニティーや会社との付合い、理解が必須。多大なコスト

＋いったん使われると、それを元に論文書きやすい

＋共同研究費などで資金的に潤う

＋OSSコミュニティでの認知度向上

＋新たな課題、要求などが見え、現実の課題解決に直結


お金
昇進
論文
産学連携

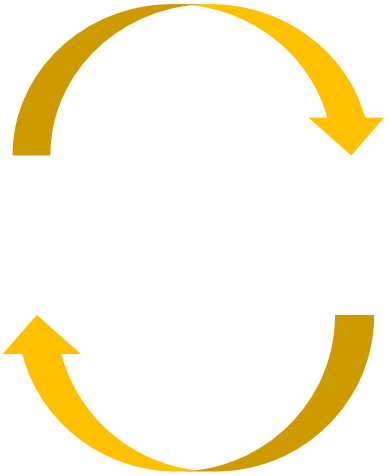
RQ2： どうすれば光るのか？

光るためのいくつかの論点

1. テーマ選び・論文戦略
2. 産学連携・ツール開発
3. 予算獲得
4. 研究と教育
5. 国際化（留学生、国際交流）
6. 大学を取り巻く環境

1.1 テーマ選び

テーマ



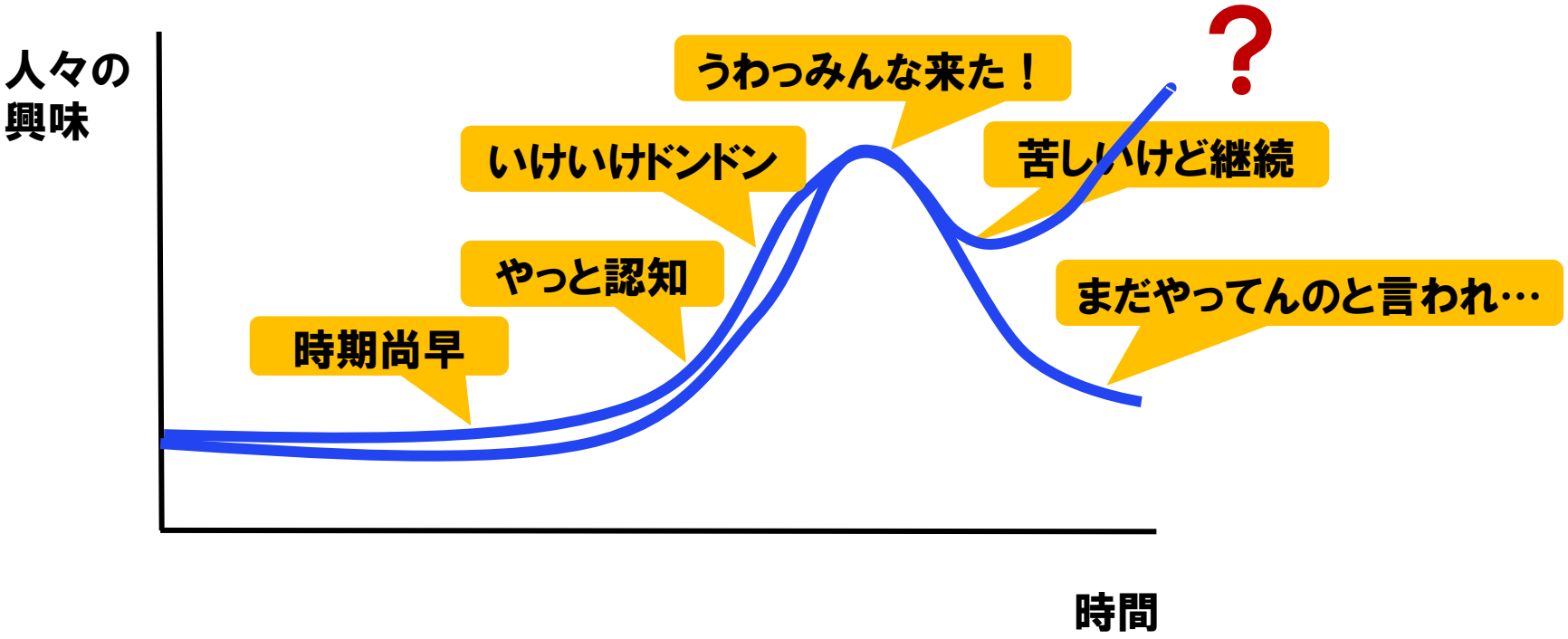
結果

1.2 テーマ選びの変遷

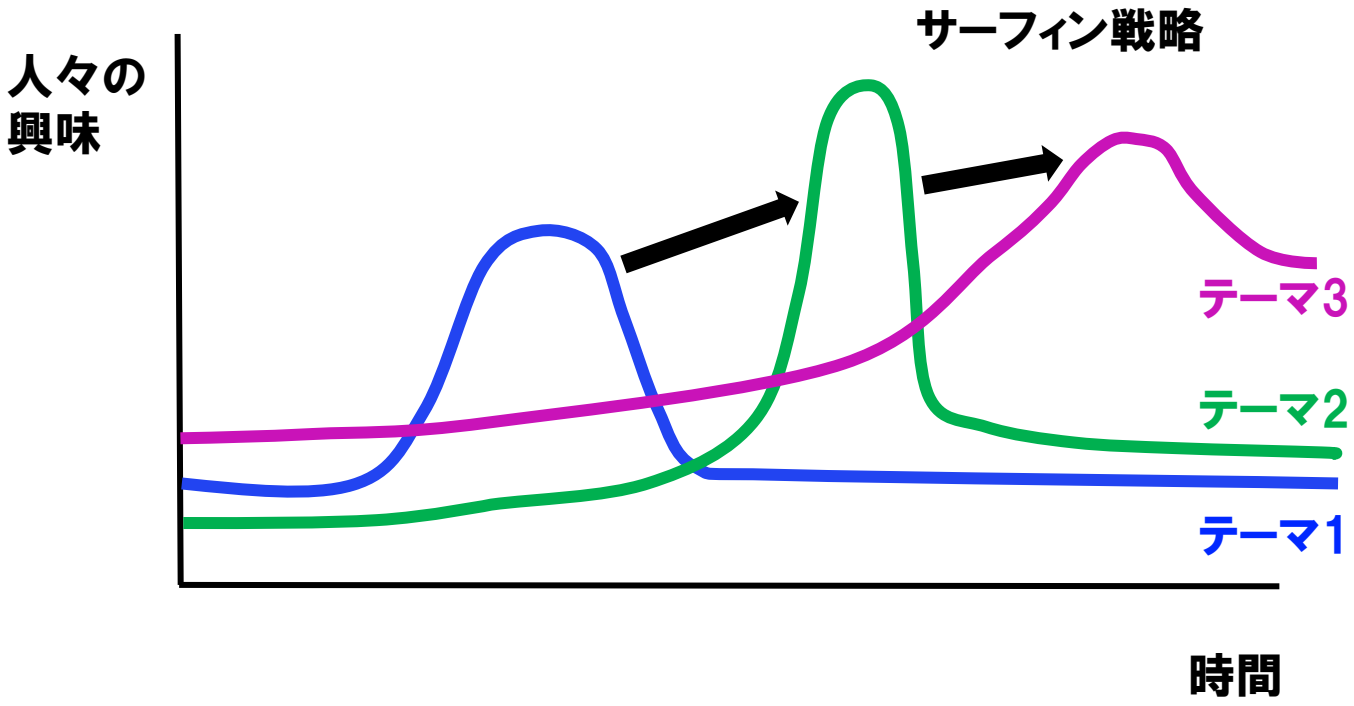
- 学生時代
  - 教員による指導　vs.　自分の選択　　- 形式的仕様　vs. インプリメント
  - その後のベース。　かなり重要　　- 形式的定義、アルゴリズム
- 自立に向けての時代
  - 研究室グループのテーマ　　- メトリクス、開発環境
  - 自分のベース研究の発展　　- プログラムスライス
  - 新たなトレンド、ニーズの勉強　　- プロセス、生物情報
- 自己責任の時代
  - グループを牽引できる魅力的なテーマ　　- マイニング、OSS関連
  - 学術的に競争力あるテーマ　　- コードクローン、コード検索
  - 社会にインパクトのあるテーマ　　- コードクローン技術応用、...

1.3 論文の発表戦略

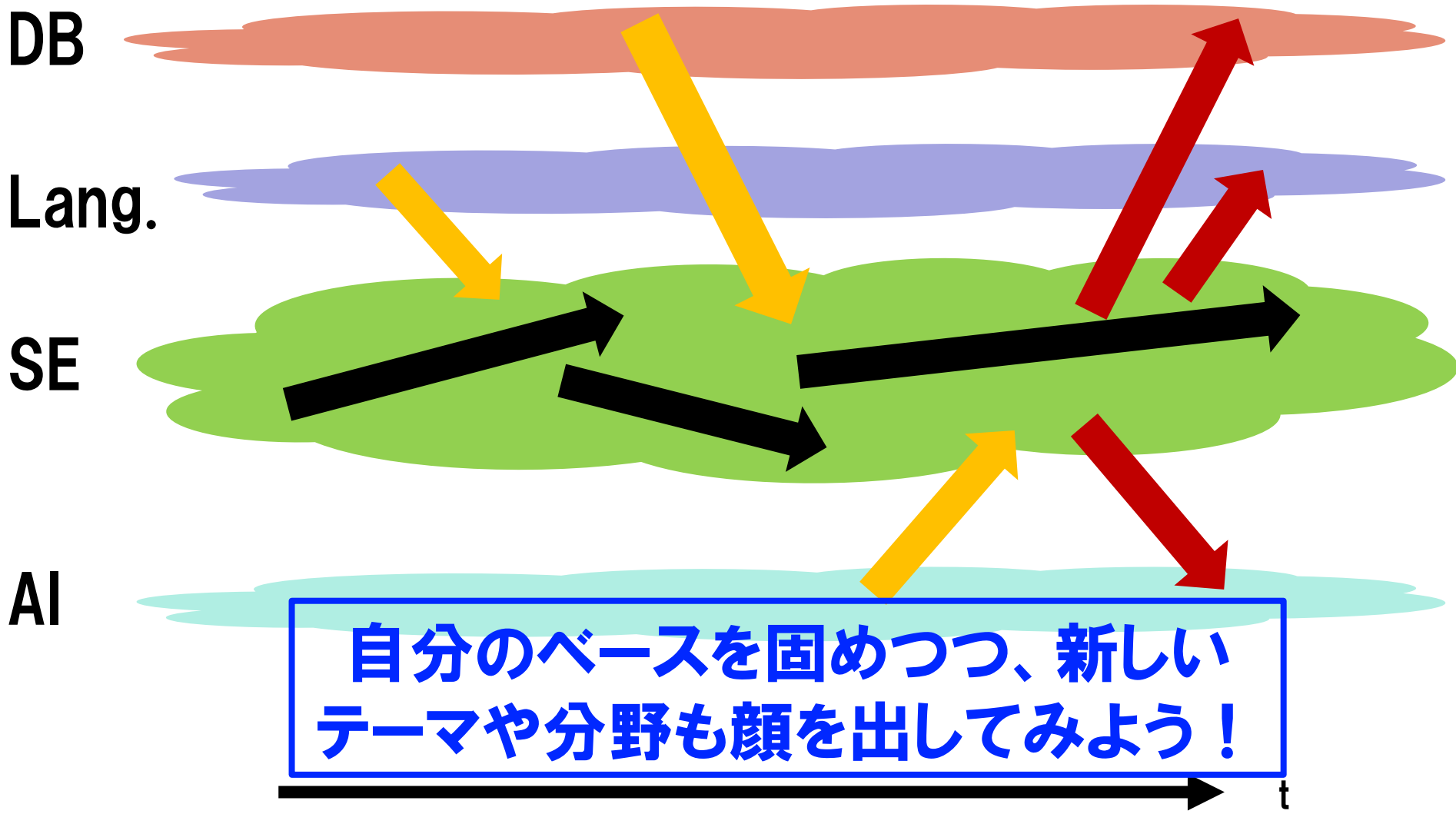
- 良いテーマ・結果をふさわしい時期のふさわしい場所に発表
  - いい場所には持てる力全部注ぐ。論文分割戦略は？
  - ふさわしい時期とは？



1.4 論文の発表戦略(その2)



1.5 他分野への参入

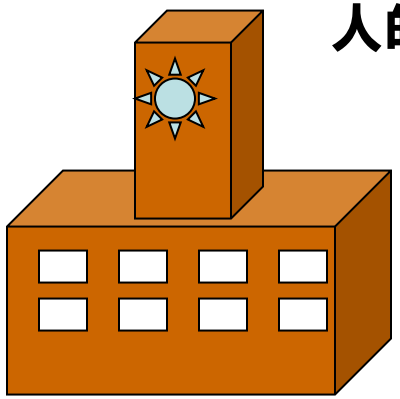


2.1 産学連携の重要性

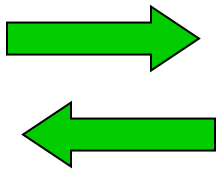
- ソフトウェア工学は、所詮、経済原則の上での科学？
  - 高いプロダクト品質、高いシステム性能、高い開発効率…
- 経済感覚がないと研究の目標が定まらない
- 産学連携を実施することで、感覚を磨く

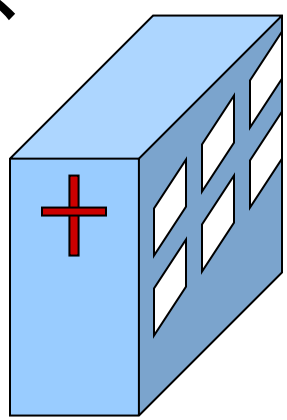
2.2 大学病院モデル

大学と社会との連携モデル

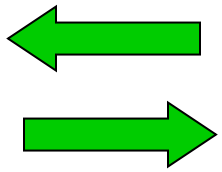


人的資源、シーズ





患者・研修医、お金、ニーズ

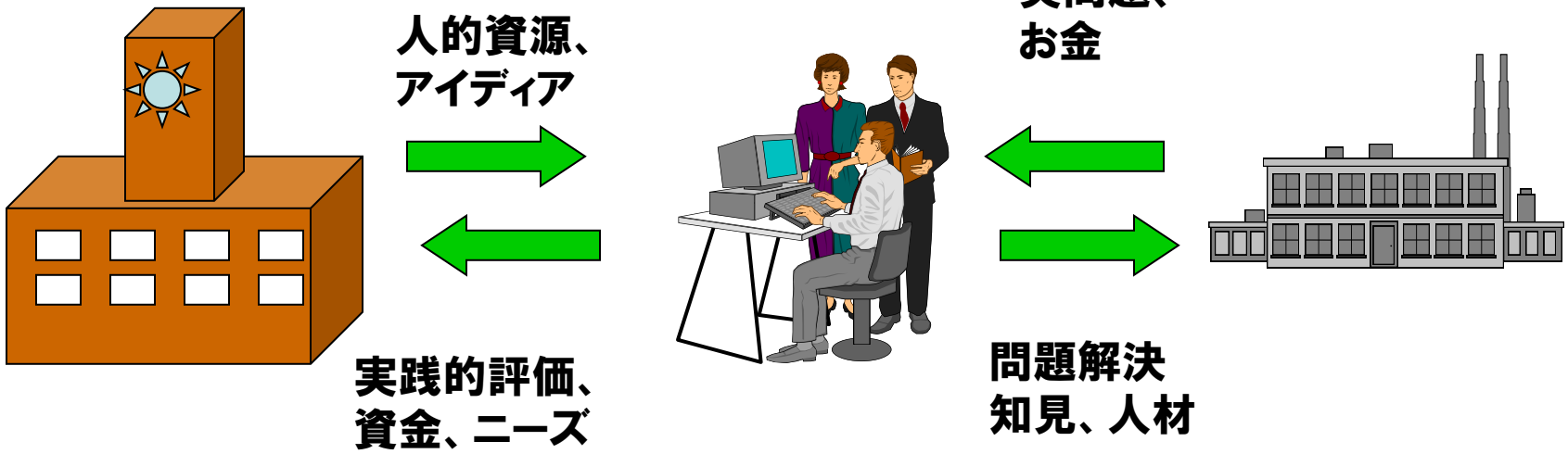


研究費、
実データ、
実践知識

治療、知識やエキスパート普及

大学　　大学病院　　社会

2.3 ソフトウェア工学の共同研究



2.4 産学連携のメリット

- 少々の手間を超えるメリット
  - 研究のネタ(ニーズ)の発見
  - 産業界の動き把握
  - 実践的な研究評価ができる
    - 論文の時の強みになる
- 少し付き合って、合わなそうなら早い目に撤退

SE研究は使われてこそ存在価値あり
とりあえず産学連携やってみよう！

似たような意見

Tao Xie, Planning and Executing Practice-Impactful Research, Chinese Computer Federation Software Engineering Special Interest Group Young Researcher Forum, Aug. 2017. https://www.slideshare.net/taoxiease/planning-and-executing-practiceimpactful-research

2.5 ツール開発、OSS貢献

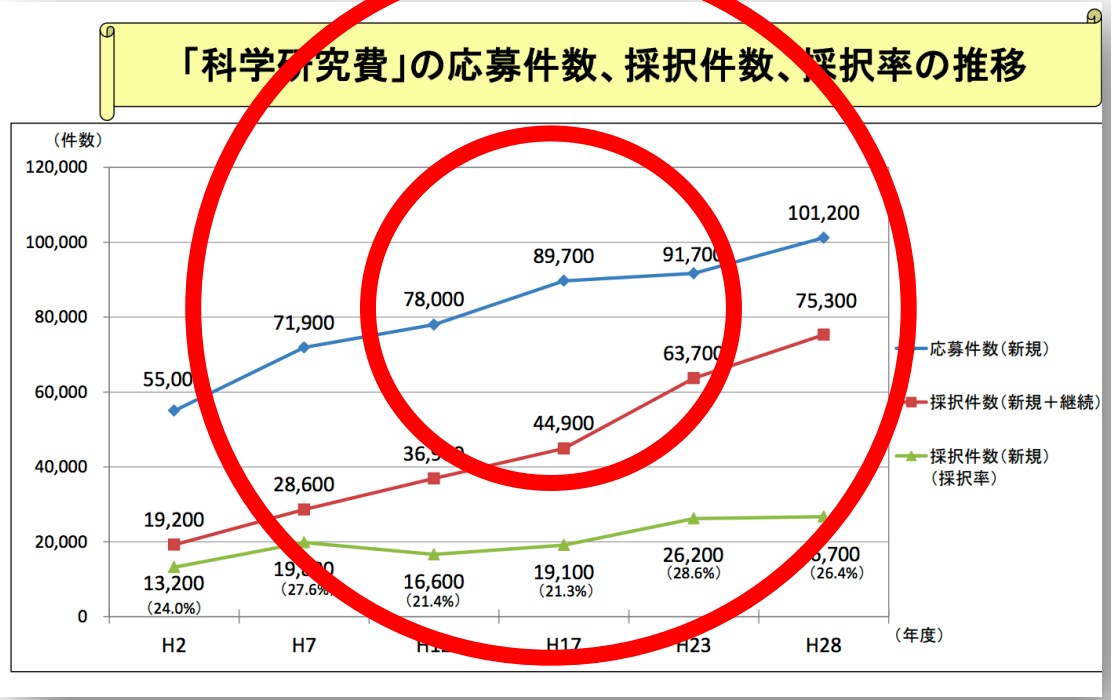
- 産学連携の一つの形態
- ツール開発は楽しい
  - コードが動き、アイディアが実現される
  - ユーザーがついて、実用化が進み、フィードバッグが励みになる
  - ユーザーが増えるとそのツールの論文の被参照数もあがる
- ツール開発は苦しい
  - 新規性、有用性を示すのは努力が必要
    - めちゃくちゃ時間をかけて論文一本。修了までに間に合わない
    - システム開発論文に走る(?)
  - 保守の要求やセキュリティ対応に時間が取られる
- 研究アイディアの重要な評価方法
- 近年、論文に付随していろいろなデータセット要求される

ぜひ日本からどんどんOSSやツールの貢献が増えてほしい！

3.1 プロジェクト予算獲得

科研費

- 出せるものは何でも出せ、という圧力（大学、研究機関、民間）
- 採択率は下がってる？

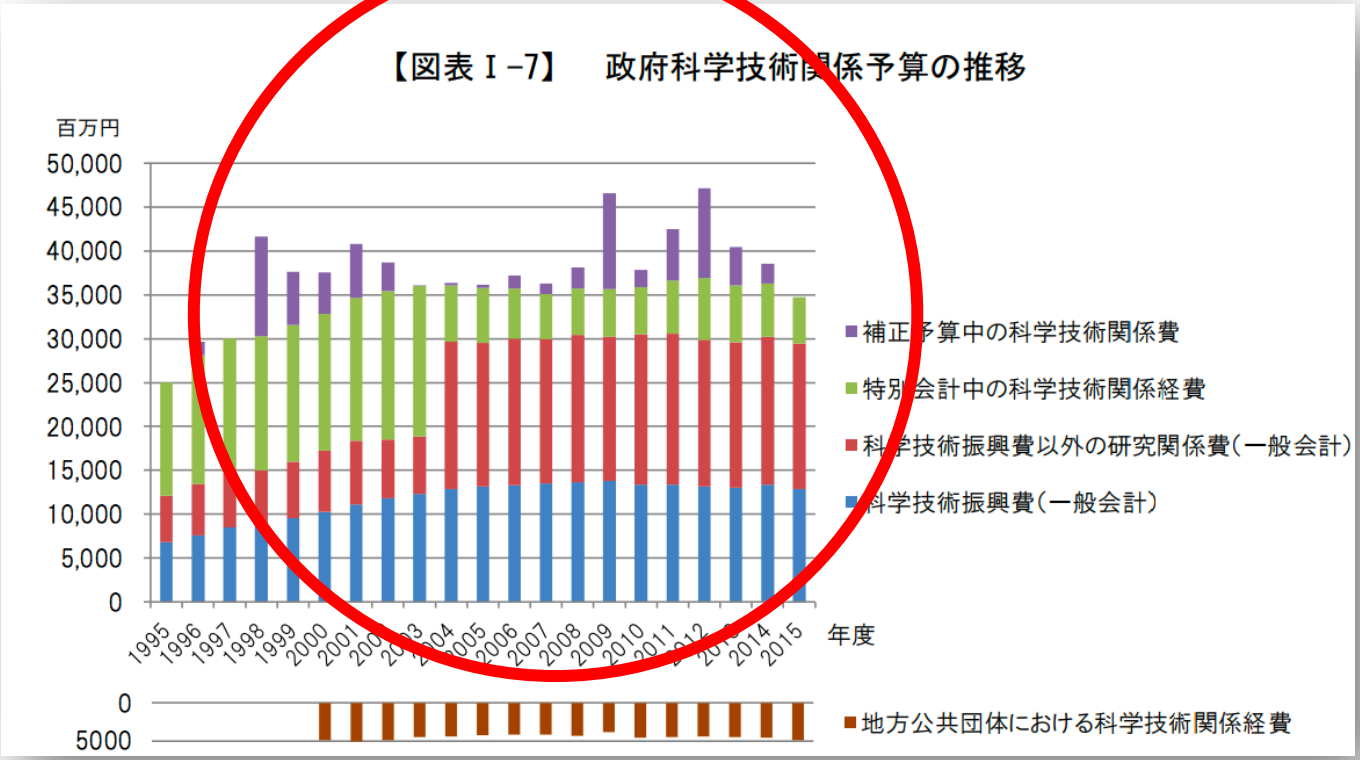


- 予算の使い勝手はよい
- 比較的自由な研究プロセス、評価も軽い

3.2 プロジェクト予算獲得(その2)

国のプロジェクト(含む文科省)

- いろいろプロセス、予算使途に制約が多い、評価が厳しい
- 忙しい、途中で目標など変えにくい
- 個々の予算規模大きい、予算総額は横ばい



3.3 プロジェクト予算獲得(その3)

獲得した予算を何に使うか？

- 設備費
  - 教員、学生の計算機
- 旅費
  - 教員、学生の発表旅費
- 人件費
  - 博士課程学生へのサポート
  - 国際公募で優秀なポスドク雇用

4 研究と教育

- 教育を充実するようにとプレッシャー増大
  - 授業日数の確保
  - 演習の充実、教育の実践化
  - 研究室の学生も手間がかかる。昔のように上級生が下級生を面倒みない…
- 割けるエフォートは有限で手を抜きたくなる
- しかしSEの目的(ソフトウェアの品質、生産性をあげる)に対して、技術者を教育しレベルアップをはかる、というのは、最も効率的な手段　→　David Notkin先生の意見
- 教育の準備をすることによって研究を整理できる(特に大学院)。欧米の大学院教育は、研究に密接に関連

5.1 国際化(留学生)

- 研究に意欲を持つ日本人学生の減少、特に博士課程学生の減少
- 留学希望学生の増加

- 必然的に留学生が増加　(井上研約1/4)
  - → 悪いことではないが…
  - 留学生指導に手間暇かかる
  - 意欲の高い素晴らしい学生もいる
  - そうでない学生もいる、しばらくは分からない
  - 日本人学生への波及効果を期待
- 何が自分、研究室、大学、日本の目標なのかよく考える…
- 指導体制やスペースのキャパを見ながら、慎重に受入れ拡大

5.2 国際化(国際共同研究)

- できるだけ交流を増やしたい
  - 研究のアクティビティ向上
  - 大学の希望(共著論文が多い方が、大学ランキング向上)
  - 学生等の刺激
    - しかし、刺激を避けたがる学生も多くて困りものだが
- 新規の相手と短期間ではなかなか成果でにくい
  - 年単位の付き合い、必須
- 科研費等の予算を国際交流(行く、呼ぶ)に使う
- 呼ぶためにはいろいろな手続き、環境設定大変
  - 秘書、ボランティア学生などの力も必要
  - 教員自身も保証人等になったり
  - サポート欲しい(大学でも一部手伝ってくれるがまだまだ)
- 密度の濃い研究打合せ
  - ビジターは時間があるが、こっちは日常業務に追われてる

6. その他現在の大学を取り巻く環境

- 日本の大学、特に国立大学は厳しい状況
  - 運営費交付金の毎年の削減によるポスト削減、予算のカット
  - 文科省等の各種プロジェクトの受託とその実施
  - 入試の複数回実施、入試ミスへの厳しい対応などによる負荷上昇
  - 各種コンプライアンスの強化
  - 大学ランキング向上、競争的プロジェクトの獲得圧力
  - 優秀な高校生獲得のためのいろいろな施作実施（改組、イベント等）
  - 大学改革のための学内会議、調整
  - プロパー教員の役職へのアサイン

    …
- こういう環境で落着いて研究できるか？
  - せざるを得ない！海外は落着いてるように見えるが中では大変？
  - できるだけ若いうちに集中的に研究
  - 夏休みなど、一定期間、確保する習慣

RQのまとめ

- RQ1: 光るソフトウェア工学研究とは？
  - 人によって価値観が違うが、SEは産業界目線大事
  - 産業界と積極的に交流して価値観を鍛えよう

- RQ2: どうすれば光るのか？
  - No Silver Bullet！
  - 企業とのコラボを大事にして、社会的インパクトの大きそうな競争力のあるテーマなどで、お金を獲得するために申請書書きをがんばり、獲得した予算で優秀な外国人ポスドクを雇い、教育にも研究との関連性を意識しながら手抜かず大学改革の諸事業に参画する
  - 自分の価値観を磨いて、自分のリソースを賢く分配し投入しよう

Reject!

よく言われてることと同じだ！新規性なし。
１例のみで一般化できない！定量的評価なし。

将来に向けて(その1)

中長期的なビジョンや夢を持とう

- ますます短期的な成果を重視される世の中の傾向
  - ある程度、短いサイクルの研究も必要(プロの研究者として)
  - しかしそれだけでは自転車操業…
- 将来の環境の変化を常にチェック
  - ハードウェア、ネットワーク
  - 周辺技術(例えばAI、GPU、クラウド、OS、…)
- 今はスケールしない問題でも将来的には実用化でき、大きなブレークスルーが期待できるテーマに**夢を持つ**。たとえば
  - プログラムの完全履歴追跡
  - 人類が作った全プログラムの収集
  - バグの完全自動修正

…

チャンス到来と思ったら全リソースを投入しよう!

将来に向けて（その2）

ソフトウェア工学をベースに「ソフトウェア」のプロに！

- 従来のソフトウェア工学の考え方
  - 品質の高いソフトウェアを効率よく開発する技術
  - まず仕様ありき。仕様に対して、安くてうまいソフトウェアを提供するという、**仕様の奴隷**
- ソフトウェアは今やモノ・サービスの価値を差別化する強力な手段
  - 魅力ある製品になることが必須。スピード重視。品質は二の次
  - 仕様はない、ビジネスやシステムレベルの新たなアイディアを、ソフトウェアに落とし込む力
    - ソフトウェアに関するビジネス分析力、企画力、実行力…
- 「私の専門はソフトウェア工学です」
  - 自分で自分の可能性を縛ってないか？

ソフトウェアのプロとしてビジネスや
システムのイノベーションに関わろう！

ありがとうございました