Journal of Science and Technology
ISSN: 2456-5660 Volume 6, Issue 06 (November -December 2021)
www.jst.org.in *DOI:https://doi.org/10.46243/jst.2021.v6.i06.pp135 - 138*

A Survey on Semantic Web Search Engine

Md Ayub Khan, Ch.Krishna Prasad, P.Sandeep Reddy
Assistant Professor[1], Associate Professor[2,3]
Dept. of CSE,
mail-id:ayubkhanmd32@gmail.com, mail-id:krishnaprasadcse@gmail.com
mail-id:sandeepreddycse@anurag.ac.in
Anurag Engineering College,Anatagiri(V&M),Suryapet(Dt),Telangana-508206

To Cite this Article
Md Ayub Khan, Ch.Krishna Prasad, P.Sandeep Reddy, **A Survey on Semantic Web Search Engine "** *Journal of Science and Technology, Vol. 06, Issue 06,-November– December 2021, pp135-138*
Article Info
Received: 21-10-2021 Revised: 19-10-2021 Accepted: 10-11-2021 Published: 27-11-2021

Abstract. Physics, biology, and medicine have well-refined public explanations of their research processes. Even in simplified form, these provide guidance about what counts as "good research" both inside and outside the field. Soft- ware engineering has not yet explicitly identified and explained either our re- search processes or the ways we recognize excellent work.
Science and engineering research fields can be characterized in terms of the kinds of questions they find worth investigating, the research methods they adopt, and the criteria by which they evaluate their results. I will present such a characterization for software engineering, showing the diversity of research strategies and the way they shift as ideas mature. Understanding these strategies should help software engineers design research plans and report the results clearly; it should also help explain the character of software engineering re- search to computer science at large and to other scientists.

Introduction

Many sciences have good explanations of their research strategies. These explanations include not only detailed guidance for researchers but also simplified views for the public and other observers. Acceptance of their results relies on the process of obtain- ing the results as well as analysis of the results themselves. Schoolchildren learn the experimental model of physics: hypothesis, controlled experiment, analysis, and pos- sible refutation. The public understands large-scale double-blind medical studies well enough to discuss the risks of experimental treatment, the ethics of withholding prom- ising treatment from the control group, and the conflicts of interest that are addressed by the blinding process.
Software engineering does not have this sort of well-understood guidance. Software engineering researchers rarely write explicitly about their paradigms of research and their standards for judging quality of results. A number of attempts to characterize software engineering research have contributed elements of the answer, but they do not yet paint a comprehensive picture. In 1980, I [7] examined the relation of engi- neering disciplines to their underlying craft and technology and laid out expectations for an engineering discipline for software. In 1984-5, Redwine, Riddle, and others [5,6] proposed a model for the way software engineering technology evolves from research ideas to widespread practice. More recently, software engineering researchers
have criticized common practice in the field for failing to collect, analyze, and report experimental measurements in research reports [9,10,11,12]. In 2001 I [8] presented preliminary sketches of some of the successful paradigms for software engineering research, drawing heavily on examples from software architecture.
Scientific and engineering research fields can be characterized by identifying what they value:
What kinds of questions are "interesting"?
What kinds of results help to answer these questions, and what research methods can produce these results?
What kinds of evidence can demonstrate the validity of a result, and how are good results distinguished from bad ones?
In this paper I attempt to make generally accepted research strategies in software en- gineering explicit by examining research in the area to identify what is widely ac- cepted in practice.

Software Technology Maturation

Redwine and Riddle [5,6] reviewed a number of software technologies to see how they develop and propagate. They

found that it typically takes 15-20 years for a tech- nology to evolve from concept formulation to the point where it's ready for populari- zation. They identify six typical phases:
Basic research. Investigate basic ideas and concepts, put initial structure on the problem, frame critical research questions.
Concept formulation. Circulate ideas informally, develop a research commu-nity, converge on a compatible set of ideas, publish solutions to specific sub-problems.
Development and extension. Make preliminary use of the technology, clarify underlying ideas, generalize the approach.
Internal enhancement and exploration. Extend approach to another domain, use technology for real problems, stabilize technology, develop training mate-rials, show value in results.
External enhancement and exploration. Similar to internal, but involving a broader community of people who weren't developers, show substantial evi-dence of value and applicability.
Popularization. Develop production-quality, supported versions of the tech- nology, commercialize and market technology, expand user community.
Redwine and Riddle presented timelines for several software technologies as they progressed through these phases up until the mid-1980s. I presented a similar analysisfor the maturation of software architecture in the 1990s [8].
Our interest here is in the first three phases, the research phases. Software engineer- ing research is intended to help improve the practice of software development, so research planning should make provisions for the transition. The Redwine-Riddle data suggests that around 10 of the 15-20 years of evolution are spent in concept formationand in development and extension (still more time is spent in basic research, but it is very difficult to identify the beginning of this phase). As a result, full understanding ofresearch strategy must account for the accumulation of evidence over time as well as for the form and content of individual projects and papers.
The IMPACT project [3] is tracing the path from research into practice. The objec-tives of the project include identifying the kinds of contributions that have substantialimpact and the types of research that are successful. Preliminary results are now beingdiscussed at conferences.

Prior Reflections on Software Engineering and Related Research

Software engineering research includes, but is not limited to, experimental research. Further, it resembles in some respects research in human-computer interaction.

Critiques of Experimental Software Engineering
In 1993, Basili laid out experimental research paradigms appropriate for software engineering [1]. Later, Tichy [9,10] and colleagues criticized the lack of quantitative experimental validation reported in conference papers:
"Computer scientists publish relatively few papers with experimentally validated re-
sults … The low ratio of validated results appears to be a serious weakness in CS re- search. This weakness should be rectified." [9]They classified 246 papers in com-
puter science and, for comparison, 147 papers in two other disciplines, according to the type of contribution in the article. The majority of the papers (259 of 403) pro- duced design and modeling results. They then assess each paper's evaluation of its results on the basis of the fraction of the article's text devoted to evaluation. They found, for example, that hypothesis testing was rare in all samples, that a large fraction (43%) of computer science design and modeling papers lacked any experimental evaluation, and that software engineering samples were worse than computer science in general.
Zelkowitz and Wallace [11,12] built on Basili's description of experimental para- digms and evaluated over 600 computer science papers and over 100 papers from other disciplines published over a 10-year period. Again, they found that too many papers have no experimental validation or only informal validation, though they did notice some progress over the 10-year period covered by their study.
These critiques start from the premise that software engineering research should follow a classical experimental paradigm. Here I explore a different question: What are the characteristics of the software engineering research that the field recognizes asbeing of high quality?

Analyzing Research Approaches with Pro Forma Abstracts
Newman compared research in human-computer interaction (HCI) to research in engi-neering [4]. He characterized engineering practice, identified three main types of re- search contributions, and performed a preliminary survey of publications in five engi-neering fields. He found that over 90% of the contributions were of three kinds:
EM Enhanced analytical modeling techniques, based on relevant theory, that can be used to tell whether the design is practicable or to make performance predictions;
ES Enhanced solutions that overcome otherwise insoluble aspects of problems, or that are easier to analyze with existing modeling techniques;
ET Enhanced tools and methods for applying analytical models and for building functionalmodels or prototypes. [4]
Newman created pro forma abstracts -- templates for stylized abstracts what would capture the essence of the papers -- for each of these types of contributions. For exam-ple, the pro forma abstract for enhanced modeling techniques is
"Existing <model-type> models are deficient in dealing with <properties> of <solu- tion strategy>. An enhanced <model-type>

is described, capable of providing more accurate analyses / predictions of <properties> in <solution strategy> designs. The model has been tested by comparing analyses / predictions with empirically measured values of <properties>." [4]He found that in order to account for a comparable
fraction of the HCI literature, he needed two more templates, for "radical solutions" and for "experience and/or heuristics".
Newman reported that in addition to helping to identify the kind of research re- ported in a paper, the pro forma abstracts also helped him focus his attention while reading the paper. It seems reasonable to assume that if authors were more con- sciously aware of typical paper types, they would find it easier to write papers that presented their results and supporting evidence clearly. The approach of characteriz- ing papers through pro forma abstracts is also useful for software engineering, thougha more expressive descriptive model as described below provides better matches withthe papers.

Broad View of Research

Brooks reflected on the tension in human computer interaction research between
"narrow truths proved convincingly by statistically sound experiments, and
broad 'truths', generally applicable, but supported only by possibly unrepre- sentative observations"[2].
The former satisfy the gold standard of science, but are few and narrow compared to the decisions designers make daily. The latter provide pragmatic guidance, but at risk of over-generalization.
Brooks proposes to relieve the tension through a *certainty-shell* structure -- to rec- ognize three nested classes of results,
Findings: well-established scientific truths, judged by truthfulness and rigor;
Observations: reports on actual phenomena, judged by interestingness;
Rules of thumb: generalizations, signed by their author but perhaps incom- pletely supported by data, judged by usefulness
with freshness as a criterion for all three.
This tension is as real in software engineering as in human computer interaction. Observations and rules of thumb provide valuable guidance for practice when findingsare not available. They also help to understand and area and lay the groundwork for the research that will, in time, yield findings.

Questions, Results, and Validation in Software Engineering

Generally speaking, software engineering researchers seek better ways to develop and evaluate software. They are motivated by practical problems, and key objectives of the research are often quality, cost, and timeliness of software products.
This section presents a model that explains software engineering research papers by classifying the types of research questions they ask, the types of results they produce, and the character of the validation they provide. This model has evolved over several years. It refines the version I presented at ICSE 2001 [8] based on discussion of the model in a graduate class and review of abstracts submitted to ICSE 2002. Its status is, in Brooks' sense, a set of observations, perhaps becoming generalization.

Types of Research Questions

Research questions may be about methods for developing software, about methods for analyzing software, about the design, evaluation, or implementation of specific sys- tems, about generalizations over whole classes of systems, or about the sheer feasibil-ity of a task. Table 1 shows the types of research questions software engineers ask, together with some examples of specific typical questions.
Among ICSE submissions, the most common kind of paper reports an improved method or means of developing software. Also fairly common are papers about meth- ods for analysis, principally analysis of correctness (testing and verification).
Looking back over the history of software engineering, there is some indication thatthe types of questions have changed as the field matures. For example, generaliza- tions, especially in the form of more formal models, are becoming more common, andfeasibility papers seem to be becoming less common as the field matures.

Software engineering will benefit from a better understanding of the research strate- gies that have been most successful. The model presented here reflects the character ofthe discipline: it identifies the types of questions software engineers find interesting, the types of results we produce in answering those questions, and the types of evi- dence that we use to evaluate the results.
Research questions are of different kinds, and research strategies vary in response. The strategy of a research project should select a result, an approach to obtaining the result, and a validation strategy appropriate to the research question. More explicit awareness of these choices may help software engineers design research projects and report their results; it may also help readers read and evaluate the literature.
The questions of interest change as the field matures. One indication that ideas are maturing is a shift from qualitative and empirical understanding to precise and quanti-tative models.
This analysis has considered individual research reports, but major results that in- fluence practice rely on accumulation of evidence from many projects. Each individ- ual paper thus provides incremental knowledge, and collections of related research projects and reports provide both confirming and cumulative evidence.

Acknowledgement

The development of these ideas has benefited from discussion with colleagues at Car- negie Mellon, at open discussion sessions at the FSE Conference, and with the pro- gram committee of ICSE 2002. The work has been supported by the A. J. Perlis Chair at Carnegie Mellon.

References

Basili, Victor R. Methodology for developing software that emphasizes experimentation. Experimental Software Engineering Challenges: A Review and Proposed Solutions. Lec- ture Notes in Computer Science #706 by Springer-Verlag, 1993, edited by H. Dieter Rombach, Victor R. Basili, and Richard Selby.
Robert Frederick Brooks, Jr. Exploring Illusions to Understand Reality: The Role of Interactive Visualization in Scientific Research. Human Factors in Computing Systems: Proceedings of the 1988 ACM SIGCHI Conference, CHI '88, pages 1-11.
The Goal of the Impact Project is to Analyze the Real-World Effects of Academic Software Engineering Studies. Summarized panel discussion from the Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), 2001

W. Newman. An first evaluation of HCI research outputs based on template abstracts. Proceedings of the 1994 Annual ACM SIGCHI Symposium on Human Factors in Computing Systems (CHI '94), pages 278-284.
Authors Samuel Redwine et al. Defense Department Software Technology Needs, Current State, and Future Prospects. IDA Working Paper P-1788, June 1984.
By S. Redwine and W. Riddle. The evolution of software technology. In May 1985, pages 189–200 were published as part of the proceedings for the eighth international conference on software engineering.
Martha Shaw. The future of software engineering. The issue of IEEE Software from November 1990 has articles on pages 15–24.
Mary C. Shaw. Research in software architecture has reached maturity. Software Engineering (ICSE 2001), pages 656-664a. Proceedings.
A. Heinz, L. Prechelt, W. F. Tichy, and P. Lukowicz Quantitative analysis of experimental assessment in computer science. From pages 9–18 of the 1995 issue of Journal of Systems Software, Volume 28.
Should Computer Scientists Do More Experimenting?," Walter F. Tichy. These are 16 good reasons why you shouldn't experiment. In the May 1998 issue of IEEE Computer, page 31.
Delores Wallace and Marvin V. Zelkowitz. The importance of testing in software development. Retrieved from "Information and Software Technology," Vol. 39, No. 11, 1997, pp.
Delores Wallace and Marvin V. Zelkowitz. Technology validation experiments using experimental models.
Published in IEEE Computer, Issue 31 (May 1998), Pages 23–31.