Empirical Software Engineering Research

*Open Science**

Norman Peitek, Annabelle Bergum, Lina Lampel, Sven Apel

**and some related topics*

Learning Goals

- Understand some of the issues of science and how the goals of open science try to address them
- Recognize the steps of the scientific process regarding publication including peer review
- Gain awareness of the (dis)advantages of some open science ideas

Open Science

- Open science encapsulates several elements towards making scientific research
  - widely accessible
  - more transparent
  - more reproducible
  - (increase impact)

”Traditional” Science

- In the past, researchers traditionally conducted a study and then reported (only) on it in a published paper
- However, this process relies on the authors’ honesty
  - Science is a competitive field (e.g., more publications = better chances at jobs, grants, …)
- The paper itself does not provide all the information
  - For example, it lacks details that are important and necessary to replicate a study
- p-hacking, HARKing, … are hard to avoid

Peer Review

- Peer review is the assessment of manuscripts by researchers with relevant expertise
  - A paper is peer reviewed when it has been scrutinized and approved by expert researchers
  - In our field of software engineering, typically three reviewers
- But peer review does not guarantee the correctness of the content of a paper
  - It still is considered a credible source of information (and better than a random blog, website, …)

Peer Review Process (in Software Engineering)

For journals:

Authors submit paper → Editor checks for general fit and then finds and assigns reviewers → Typically 3 reviewers provide an evaluation → Editor synthesizes reviews and responds to authors

~ 6 months

- Decision can be: Accept, minor revision, major revision, reject
  - A revision requires the authors to make changes and/or justify the current version in a *response letter*
  - Minor revision: only the editor checks the changes
  - Major revision: all peer reviewers check the changes

Peer Review Process (in Software Engineering)

For conferences:

Authors submit paper

Program chair checks for general fit and then finds and assigns reviewers

Typically 3 reviewers provide an evaluation

Program chair synthesizes reviews and responds to authors

~ 3 months

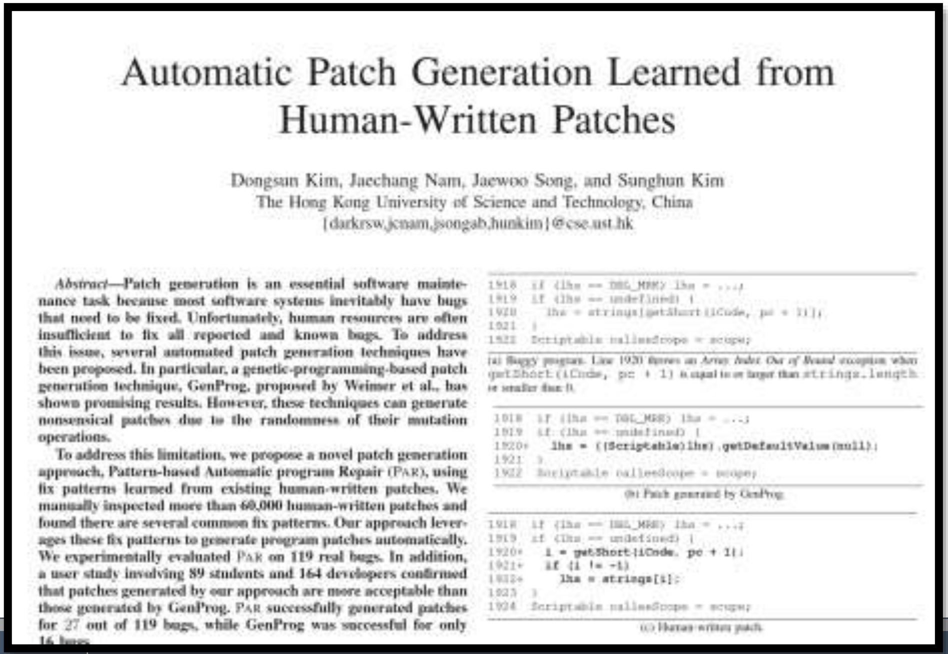
- Decision is typically: accept, conditionally accept, reject
  - Some conferences have a rebuttal period where authors can respond to questions
  - There are exceptions. This year’s FSE conference also allows major revision (for the first time)

Types of Peer Review

- Regular peer review
  - Academic theses (e.g., dissertations)
- “single-blind” (or single-anonymous) review
  - Reviewers know the authors’ identity, but authors’ do not know the reviewers’ identity
- “double-blind” (or double-anonymous) review
  - Neither reviewers nor authors know each others’ identities
- “triple-blind” (or triple-anonymous) review
  - Neither reviewers nor authors know each others’ identities
  - The editor/program chair also does not know either of the identities

Post-Publication Peer Review

- After a paper has been peer reviewed and published, future readers continue to provide some quality control
  - If readers disagree with a paper, they can either contact the editor of the paper or publish their own paper presenting their viewpoint

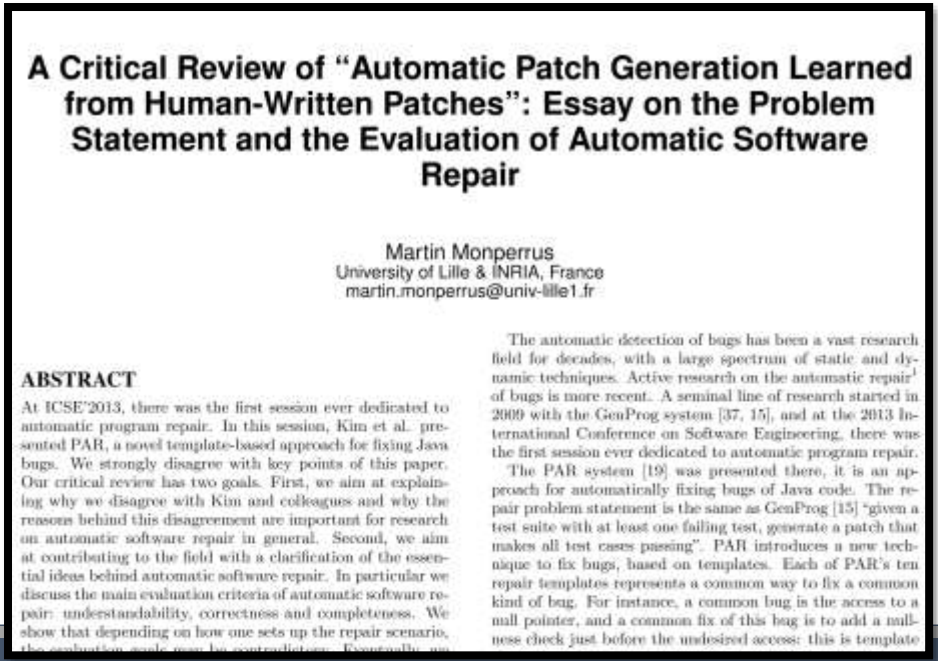


Automatic Patch Generation Learned from Human-Written Patches

Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim
The Hong Kong University of Science and Technology, China
{darkrsw,jcnam,jsongab,hunkim}@cse.ust.hk

Abstract—Patch generation is an essential software maintenance task because most software systems inevitably have bugs that need to be fixed. Unfortunately, human resources are often insufficient to fix all reported and known bugs. To address this issue, several automated patch generation techniques have been proposed. In particular, a genetic-programming-based patch generation technique, GenProg, proposed by Weimer et al., has shown promising results. However, these techniques can generate nonsensical patches due to the randomness of their mutation operations.

To address this limitation, we propose a novel patch generation approach, Pattern-based Automatic program Repair (PAR), using fix patterns learned from existing human-written patches. We manually inspected more than 60,000 human-written patches and found there are several common fix patterns. Our approach leverages these fix patterns to generate program patches automatically. We experimentally evaluated PAR on 119 real bugs. In addition, a user study involving 89 students and 164 developers confirmed that patches generated by our approach are more acceptable than those generated by GenProg. PAR successfully generated patches for 27 out of 119 bugs, while GenProg was successful for only 16 bugs.



A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair

Martin Monperrus
University of Lille & INRIA, France
martin.monperrus@univ-lille1.fr

ABSTRACT

At ICSE'2013, there was the first session ever dedicated to automatic program repair. In this session, Kim et al. presented PAR, a novel template-based approach for fixing Java bugs. We strongly disagree with key points of this paper. Our critical review has two goals. First, we aim at explaining why we disagree with Kim and colleagues and why the reasons behind this disagreement are important for research on automatic software repair in general. Second, we aim at contributing to the field with a clarification of the essential ideas behind automatic software repair. In particular we discuss the main evaluation criteria of automatic software repair: understandability, correctness and completeness. We show that depending on how one sets up the repair scenario,

The automatic detection of bugs has been a vast research field for decades, with a large spectrum of static and dynamic techniques. Active research on the automatic repair[1] of bugs is more recent. A seminal line of research started in 2009 with the GenProg system [37, 15], and at the 2013 International Conference on Software Engineering, there was the first session ever dedicated to automatic program repair.

The PAR system [19] was presented there, it is an approach for automatically fixing bugs of Java code. The repair problem statement is the same as GenProg [15] "given a test suite with at least one failing test, generate a patch that makes all test cases passing". PAR introduces a new technique to fix bugs, based on templates. Each of PAR's ten repair templates represents a common way to fix a common kind of bug. For instance, a common bug is the access to a null pointer, and a common fix of this bug is to add a nullness check just before the undesired access: this is template

Peer Review: Potential Issues

- Peer review cannot catch all problems or guarantee quality
- Potential homophily
  - Reviewers generally rate work similar to their own higher than outside their normal approaches
- High variance in decisions (“luck”)
  - In 2014, the NeurIPS conference split the program committee. Some papers were reviewed independently by two groups of reviewers
  - From the 166 papers that were reviewed twice, the set of accepted papers overlapped by 43%
  - More than half of the papers were accepted by one group were rejected by the other group!

Peer Review: One of My Experiences

- My third large fMRI study on complexity metrics felt like my best study as part of my PhD journey
  - Submitted to ICSE 2020
    - Strong accept, accept, strong reject
    - → rejected
  - Submitted to FSE 2021
    - accept, weak reject, weak reject
    - → rejected
  - Submitted to ICSE 2021
    - Strong accept, strong accept, strong accept
    - ACM SIGSOFT Distinguished Paper Award

Program Comprehension and Code Complexity Metrics: An fMRI Study

Norman Peitek
Leibniz Institute for Neurobiology
Magdeburg, Germany

Sven Apel
Saarland University, Saarland Informatics Campus
Saarbrücken, Germany

Chris Parnin
NC State University
Raleigh, North Carolina, USA

André Brechmann
Leibniz Institute for Neurobiology
Magdeburg, Germany

Janet Siegmund
Chemnitz University of Technology
Chemnitz, Germany

Peer Review: Your Experiences

- Have you had experience with peer review?
- Is peer review comparable to code review? If so, can we adapt ideas in either direction?

Peer Review: New Trends

- Newer more “open science-y” forms of peer review address some issues
- Open peer review
  - publish paper along with the reviews (and the reviewers' identities) and the response letter
- Registered reports
  - First, peer review the experiment design (and after approval authors conduct the study)
  - Second, peer review the paper reporting on the experiment (focusing on presentation)

Registered Reports

Focus review on research questions and proposed experiment design

Focus on quality of presentation only.
The outcome of the results are irrelevant.

Registered Reports

- Eliminates reviews demanding a different experiment design
  - Reduces the chance to have to collect data twice
- Eliminates bias against negative/non-results and does not force authors to glamourize their results (or pick a specific positive result)
- But: more work for the reviewers and mortality can be frustrating
- But: extends the research timeline for authors

Preregistration

- Preregistration means researchers specified their experiment design including research questions and data analysis *before* collecting data
  - Or, if you work on an existing data set, before analyzing the data
- Separates hypothesis-generating (exploratory) and hypotheses-testing (confirmatory) research
  - With preregistration, it eliminates the chance that authors (even by accident) mix these and conduct p-hacking or HARKing
  - Difference to registered reports: no peer review of experiment design

Publishing & Open Access

Traditional Publishing

- After concluding a study and writing up the paper, authors submit it to a publishing vendor (conference/journal)
- Journals are highly profitable
- For example, Elsevier owns around 17% of all published science material
    - In 2017, Elsevier had a revenue of ~3.2 billion USD with extremely high profit margins of 30-40%

Authors Submit Paper → Peer Review → Authors Publish Paper → Publication → Access

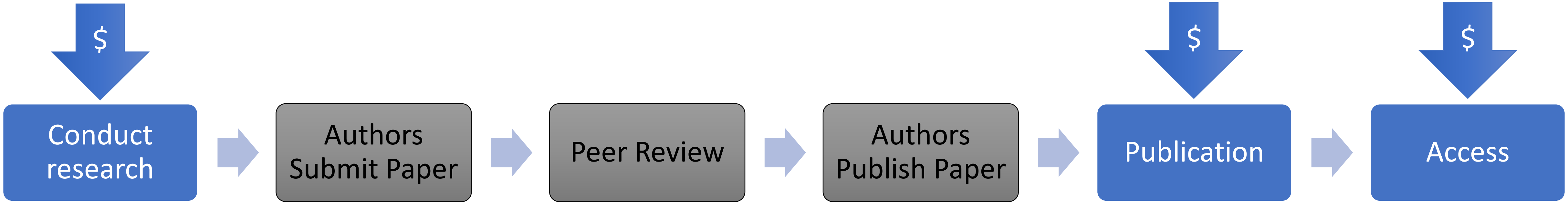
Authors Submit Paper	Peer Review	Authors Publish Paper	Publication	Access
	~~$~~	$$$	$	$$$

Traditional Publishing

- The commercialization can be extremely problematic for science
- In my case: submitted paper to Transaction of Software Engineering (TSE) at IEEE (a non-profit organization)
  - After submission, but before publication, the publication rules changed and papers above 12 pages in length had to pay $200/page extra
  - → $2500 USD for one paper
  - Also puts reviewers in a difficult spot: should you request more (useful) content knowing it costs authors a lot of extra money?
- Many researchers from lower-income countries cannot afford to access or publish in international journals because of the high costs

Traditional Publishing

- Public (e.g., Germany) has to pay to access its own research



- Grants (DFG, ERC) required self-archival in the past (e.g., on your own website)
- Nowadays, they typically demand open access

Self-Archival of Publications

- In addition to official publication, many researchers "self-archive" the author accepted manuscript (AAM)
  - Official publication may take years to publish (my TSE paper: accepted 2017, published in 2020)
  - Self-archival is typically immediately on (or before) acceptance on the authors' personal website
    - Or on an institutional repository (Saarland University publication server)
    - Or on a general-purpose repository (arxiv.org, …)
    - Sometimes self-archival is forbidden or restricted by the publisher
  - If you do not have access to a journal/digital library, check the authors' personal websites and search additional repositories
    - Use tools such as OpenAire or Google scholar
    - If they are not available, email the first/contact author. If they are still in science, they almost always love to send their paper as a pdf

Finding (Self-Archived) Papers

- Use tools such as [OpenAire]() or Google scholar

Program comprehension and code complexity metrics: An fmri study
N Peitek, S Apel, C Parnin... - 2021 IEEE/ACM ..., 2021 - ieeexplore.ieee.org
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and tempting to use code metrics for this purpose, their validity is debated, since they rely on simple code properties and rarely consider particularities of human cognition. Aims: We investigate whether and how code complexity metrics reflect difficulty of program comprehension. Method: We have conducted a functional magnetic resonance imaging ...
☆ Save 99 Cite Cited by 24 Related articles All 6 versions

[PDF] ieee.org

Program comprehension and code complexity metrics: An fmri study
N Peitek, S Apel, C Parnin... - 2021 IEEE/ACM ..., 2021 - ieeexplore.ieee.org
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
☆ Save 99 Cite Cited by 24 Related articles

Program Comprehension and Code Complexity Metrics: An fMRI Study
N Peitek, S Apel, C Parnin, A Brechmann... - Proceedings of the 43rd ..., 2021 - dl.acm.org
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
99 Cite

Program Comprehension and Code Complexity Metrics: An fMRI Study
N Peitek, S Apel, C Parnin, A Brechmann... - 2021 IEEE/ACM 43rd ..., 2021 - computer.org
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
99 Cite

[PDF] Program Comprehension and Code Complexity Metrics: An fMRI Study
N Peitek, S Apel, C Parnin, A Brechmann, J Siegmund - se.cs.uni-saarland.de
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
99 Cite

[PDF] uni-saarland.de

Program Comprehension and Code Complexity Metrics: An fMRI Study
N Peitek, S Apel, C Parnin, A Brechmann... - 2021 IEEE/ACM ..., 2021 - store.computer.org
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
99 Cite

[PDF] Program Comprehension and Code Complexity Metrics: An fMRI Study
N Peitek, S Apel, C Parnin, A Brechmann, J Siegmund - tu-chemnitz.de
Background: Researchers and practitioners have been using code complexity metrics for decades to predict how developers comprehend a program. While it is plausible and ...
99 Cite

[PDF] tu-chemnitz.de

Traditional Publishing → "Open Access"

- Example: "Empirical Software Engineering" Journal by Springer
  - One of the most relevant journals in our field
- In the past, it was a traditional publishing journal
  - Authors' paid to publish their paper
  - Readers' paid to read them
- Nowadays, Empirical Software Engineering offers traditional and open access as option ("hybrid journal")
  - With the goal towards becoming a pure open access journal ("transformative journal")

Open Access

- Furthest developed element of open science
- Two main types of open access
  - Gold open access = publisher provides free access
  - Green open access = author provides free access through self-archival
- Financing of publication costs typically through the authors
  - Article processing charges (APC)
    - For example: traditional publishing $1000, open access $3000
  - Some funding conflicts remain

Advantages of Open Access

- From an author perspective
  - Maximize research impact
  - Increase exposure by wider access
  - No copyright transfer necessary, publication under Creative Commons licenses

- Generally, improved scientific process

Open Data

Open Data

- Extension of open access and makes the underlying data available under the same principles (available for free and immediately)
- In addition, open data underlie the FAIR principle
  - Findable
  - Accessible
  - Interoperable
  - Reusable
- More recently, also mandated by many grants (EU/DFG)

Open Data: Technical Requirements

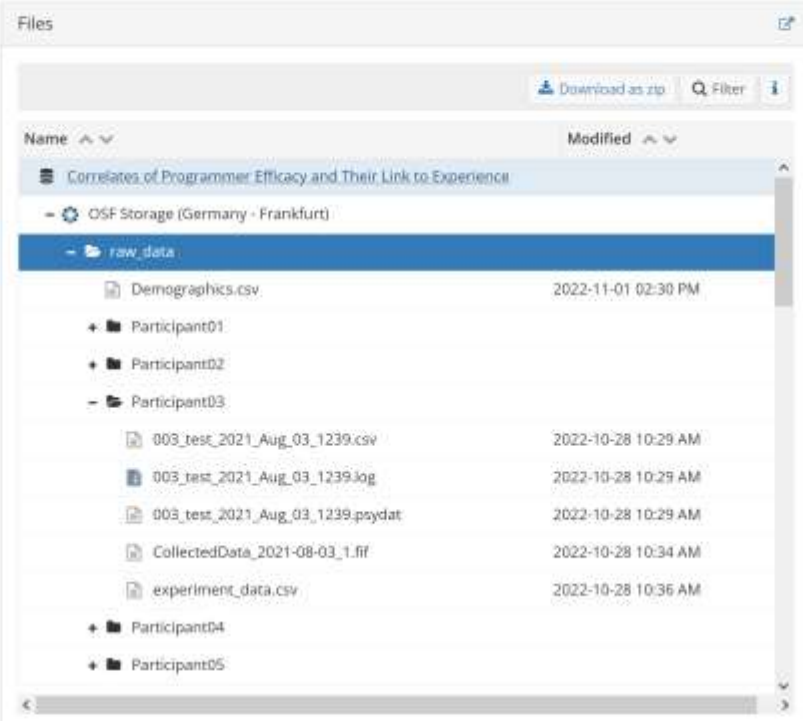
- Meta data is crucial to understand the raw data
- License can be freely chosen, but impacts future uses (ideally Creative Commons)
- Data must be stored in a suitable long-term archive underlying suitable data management plans (backups, …)
  - Typically institutional (e.g., university) or organizational (e.g., OSF, Zenodo)

Open Data: Advantages

- Increases potential for collaborations
- Increased *reproducibility*
- Decreases hurdles for lower-income countries
    - I have received emails from researchers before that cannot afford EEG/fMRI devices, but are interested in this research direction
    - → open data enables them to test their hypotheses on existing data

Open Data: Problems

- Privacy and ethical considerations
  - Will discuss in the next lecture
- Large data sets can occur data-storage costs
  - Our last EEG and eye-tracking data set: ~35 GB
  - Common neuroimaging (EEG) repositories are not suitable for eye-tracking data
  - General open-science repositories typically cap the data (e.g., to 10 GB)
  - Open Science Framework (OSF) allows 50 GB
    - Our data set: https://osf.io/4hjbd/



Open Data & Open Source

- Do open data and open source projects face similar challenges?

- How are they different?

Further Reading & Materials

- https://cacm.acm.org/blogs/blog-cacm/248824-how-objective-is-peer-review/fulltext
- Open Access Explained! https://www.youtube.com/watch?v=L5rVH1KGBCY