Software Systems Verification and Validation

Vescan Andreea, PhD, Assoc. Prof. Habil.

Faculty of Mathematics and Computer Science
Babeș-Bolyai University

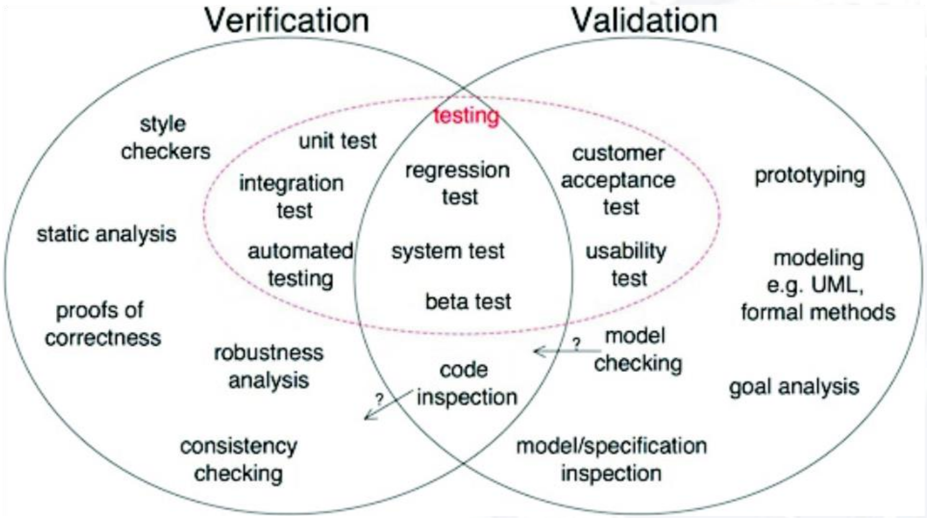
2023-2024

Software Systems Verification and Validation

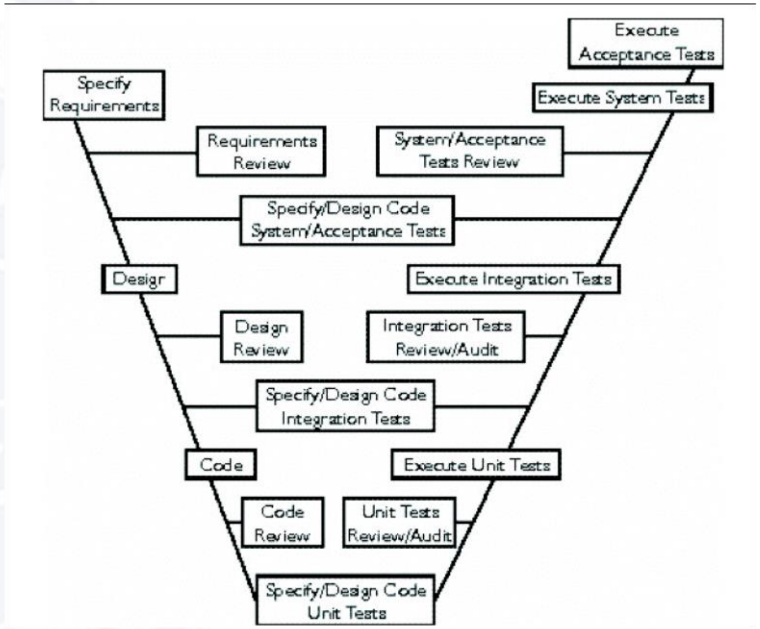
"Tell me and I forget, teach me and I may remember, involve me and I learn."

(Benjamin Franklin)

What we will learn!



SDLC – V Model

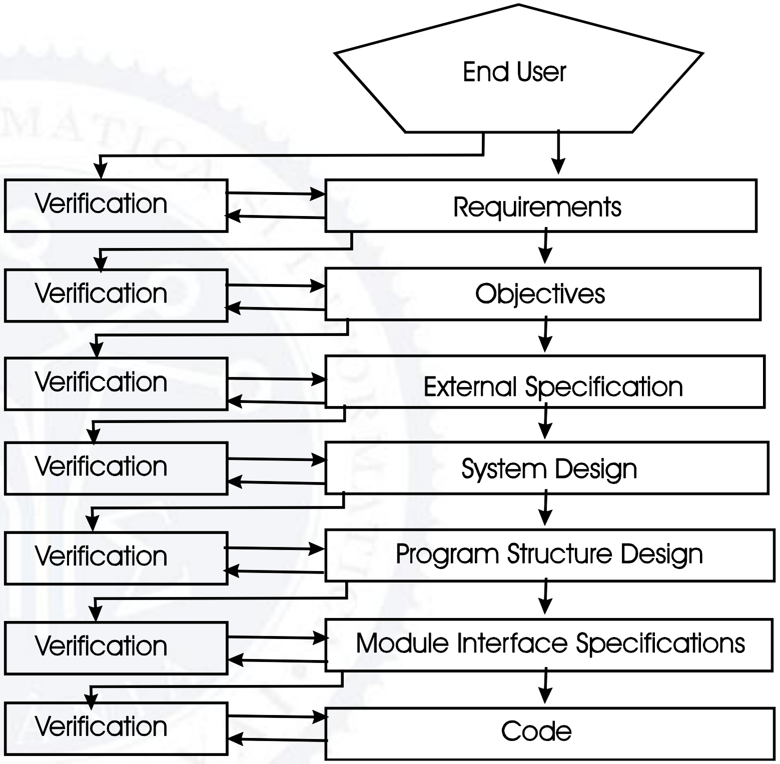

Execute Acceptance Tests
Specify Requirements
Execute System Tests
Requirements Review
System/Acceptance Tests Review
Specify/Design Code System/Acceptance Tests
Design
Execute Integration Tests
Design Review
Integration Tests Review/Audit
Specify/Design Code Integration Tests
Code
Execute Unit Tests
Code Review
Unit Tests Review/Audit
Specify/Design Code Unit Tests

Outline

- Human testing
- Human testing methods
  - Inspections
  - Walkthroughs
  - Pair-programing

Human testing

- Prevent errors
  - introduction of a verification step at the end of each process.

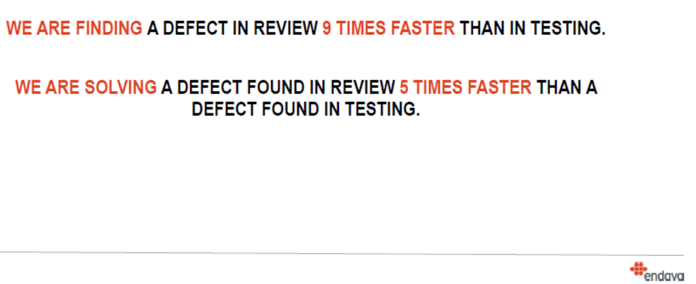


Outline

- Human testing
- Human testing methods
  - Inspections
  - Walkthroughs
  - Pair-programing

Human testing methods

- Is it useful? they contribute to productivity and reliability:
  - The earlier errors are found, the lower the cost of correcting the errors.
  - Psychological change of programmers when computer-based testing commences.
- Human testing methods are:
  - Inspections
  - Walkthroughs
  - Pair-programing



WE ARE FINDING A DEFECT IN REVIEW 9 TIMES FASTER THAN IN TESTING.

WE ARE SOLVING A DEFECT FOUND IN REVIEW 5 TIMES FASTER THAN A DEFECT FOUND IN TESTING.

- Objective - to find errors but not to find solutions to the errors.
- Advantage - when an error is found it is usually located.
  - **Finds from 30% to 70% of the logic-design/coding errors in programs (?).**
- Inspection and computer-based testing are complementary.

[Mye04] (chapter 3), [PY08] (chapter 18), [Fre10] (chapter 4)

Outline

- Human testing
- Human testing methods
  - Inspections
  - Walkthroughs
  - Pair-programing

Inspection

- **Inspection** - process of trying to find defects in development documents during various phases of the software development process.
- Fagan Inspection team ([4 members])
  - Moderator - duties
    - Distributing materials and scheduling the inspection session.
    - Leading the session
    - Ensuring that the errors are subsequently corrected.
  - Author of the product (analyst, designer, programmer)
  - Secretary
  - Reader
- Checklists
- Time - 90-120 minutes

Inspection activities

- **Planning**
  - the moderator selects the team members
  - distribution of the materials to the members; task assignment
- **Presentation/Overview - not compulsory**
  - used to present details to the members of the inspection team
- **Individual preparation**
  - reading and understanding the received documentation
- **Inspection meeting**
  - critical observations of each individual inspectors - discussed
  - conclusions of the inspection - documented
- **Rework**
  - the author makes the required changes and correct the errors
- **Follow-up**
  - to verify if the modification did eliminate the errors
  - may be only between the author and the moderator

Inspection **checklists**

- Inspection scope - to find errors
- Depending on the analyzed document - special kind of errors
- **Specification Document**
  - Does the specification conform to the user’s needs?
  - Are there ambiguities in the specification?
  - Do the input/output date are clearly stated? What about input/output conditions?
  - Are there requirements that are not present in the specification?
  - Are there performance conditions? What precise computation conditions?
- **Analysis Document**
  - Does the design conform to the specification?
  - Are all the functionalities from the specification specified?
- Is there an analysis documentation about the made decisions? Inspection scope - to find errors

- Depending on the analyzed document - special kind of errors
- **Specification Document**
  - Does the specification conform to the user’s needs?
  - Are there ambiguities in the specification?
  - Do the input/output date are clearly stated? What about input/output conditions?
  - Are there requirements that are not present in the specification?
  - Are there performance conditions? What precise computation conditions?
- **Analysis Document**
  - Does the design conform to the specification?
  - Are all the functionalities from the specification specified?
  - Is there an analysis documentation about the made decisions?

- **Code**
  - Does the code conform to the design?
  - Are all the methods are called?
  - Are all the variables initialized?
  - Problems with: infinite cycles, out of bound indexes, improper allocation of memory.
- **Test Document**
  - The test cases are well documented?
  - The test cases are well chosen?
  - Are the test data sufficient to coverage criterion?
  - For the integration testing, the order of integration is clear?
  - At regression testing is the testing continued?

Inspection advantages [CB03]

- Early error discovery
- Reduce product development time and cost
- Group method
- Mean to education
- The source of error is known (locating defect)
- Eliminates the debugging stress if few day remains until product release

- Inspection – more efficient than testing [CB03]
  - detecting, locating, repairing defect
  - a two-pass approach (individuals first and by the group)
  - checklist – calls attention to specific defect prone areas

Outline

- Human testing
- Human testing methods
    - Inspections
    - Walkthroughs
    - Pair-programing

Walkthrough

- **Walkthrough** [You79], [CB03] - process of trying to find defects in development documents during various phases of the software development process.
- Similar to Inspection
- **Team members** ([3-5] members)
  - Moderator ([CB03]- moderator = the producer of the reviewed material
    - ➔ a larger amount of material can be processed by the group)
  - Secretary
  - Tester
- **Procedures** are slightly different
  - Planning
  - Meeting - the participants “play computer” (no checklist)
  - No Individual preparation [CB03]
  - Rework [You79]
  - Follow-up
- Different error-detection technique
- Time - 90-120 minutes

Outline

- Human testing
- Human testing methods
    - Inspections
    - Walkthroughs
    - Pair-programing

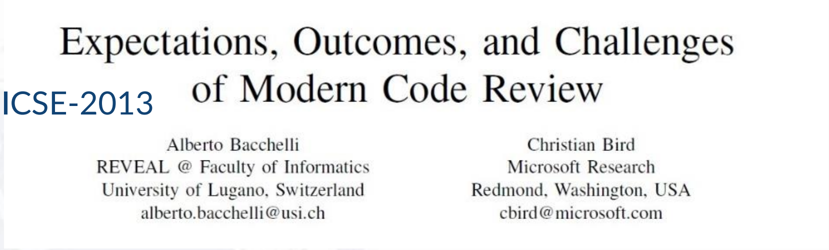
Pair-Programming

- Variation of program inspection.
- Merges coding and inspection activities.
- The inspection activities
  - are not driven by checklists
  - are based on shared programming practice and style
- Programmers frequently alternate roles
- Is carried out in normal workdays, without excessive overtime and without severe schedule pressure.
- No mediator, so responsibility for open and non-defensive discussion of decisions/alternatives falls to the programmers.

Outline

- Human testing
- Human testing methods
  - Inspections
  - Walkthroughs
  - Pair-programing

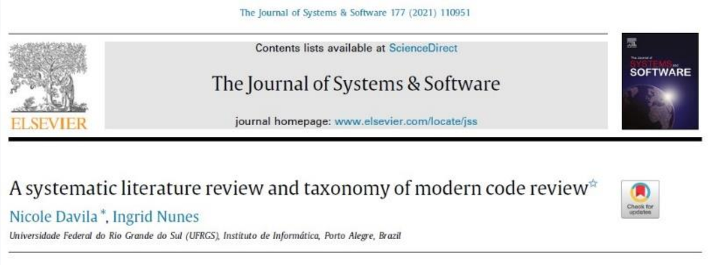
- Modern Code Review
  - ICSE-2013



Expectations, Outcomes, and Challenges of Modern Code Review

Alberto Bacchelli
REVEAL @ Faculty of Informatics
University of Lugano, Switzerland
alberto.bacchelli@usi.ch

Christian Bird
Microsoft Research
Redmond, Washington, USA
cbird@microsoft.com



The Journal of Systems & Software 177 (2021) 110951

Contents lists available at ScienceDirect

The Journal of Systems & Software

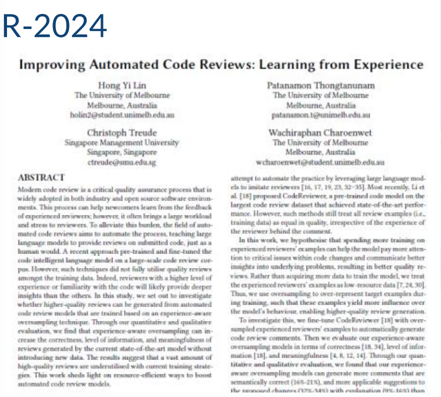
journal homepage: www.elsevier.com/locate/jss

ELSEVIER
SYSTEMS and SOFTWARE
Check for updates

A systematic literature review and taxonomy of modern code review☆

Nicole Davila*, Ingrid Nunes
Universidade Federal do Rio Grande do Sul (UFRGS), Instituto de Informática, Porto Alegre, Brazil

- MSR-2024



Improving Automated Code Reviews: Learning from Experience

Hong Yi Lin
The University of Melbourne
Melbourne, Australia
holin2@student.unimelb.edu.au

Patanamon Thongtanunam
The University of Melbourne
Melbourne, Australia
patanamon.t@unimelb.edu.au

Christoph Treude
Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Wachiraphan Charoenwet
The University of Melbourne
Melbourne, Australia
wcharoenwet@student.unimelb.edu.au

ABSTRACT

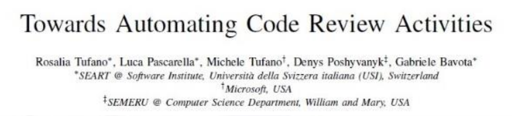
Modern code review is a critical quality assurance process that is widely adopted in both industry and open source software environments. This process can help newcomers learn from the feedback of experienced reviewers; however, it often brings a large workload and stress to reviewers. To alleviate this burden, the field of automated code reviews aims to automate the process, teaching large language models to provide reviews on submitted code, just as a human would. A recent approach pre-trained and fine-tuned the code intelligent language model on a large-scale code review corpus. However, such techniques did not fully utilise quality reviews amongst the training data. Indeed, reviewers with a higher level of experience or familiarity with the code will likely provide deeper insights than the others. In this study, we set out to investigate whether higher-quality reviews can be generated from automated code review models that are trained based on an experience-aware oversampling technique. Through our quantitative and qualitative evaluation, we find that experience-aware oversampling can increase the correctness, level of information, and meaningfulness of reviews generated by the current state-of-the-art model without introducing new data. The results suggest that a vast amount of high-quality reviews are underutilised with current training strategies. This work sheds light on resource-efficient ways to boost automated code review models.

attempt to automate the practice by leveraging large language models to imitate reviewers [16, 17, 19, 23, 32–35]. Most recently, Li et al. [18] proposed CodeReviewer, a pre-trained code model on the largest code review dataset that achieved state-of-the-art performance. However, such methods still treat all review examples (i.e., training data) as equal in quality, irrespective of the experience of the reviewer behind the comment.

In this work, we hypothesise that spending more training on experienced reviewers' examples can help the model pay more attention to critical issues within code changes and communicate better insights into underlying problems, resulting in better quality reviews. Rather than acquiring more data to train the model, we treat the experienced reviewers' examples as low-resource data [7, 24, 30]. Thus, we use oversampling to over-represent target examples during training, such that these examples yield more influence over the model's behaviour, enabling higher-quality review generation.

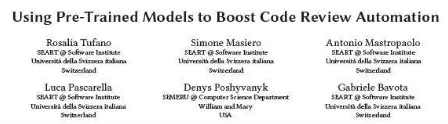
To investigate this, we fine-tune CodeReviewer [18] with oversampled experienced reviewers' examples to automatically generate code review comments. Then we evaluate our experience-aware oversampling models in terms of correctness [18, 34], level of information [18], and meaningfulness [4, 8, 12, 14]. Through our quantitative and qualitative evaluation, we found that our experience-aware oversampling models can generate more comments that are semantically correct (16%-21%), and more applicable suggestions to the proposed changes (12%-34%) with explanation (9%-16%) than

- ICSE-2021



Towards Automating Code Review Activities

Rosalia Tufano*, Luca Pascarella*, Michele Tufano†, Denys Poshyvanyk‡, Gabriele Bavota*
*SEART @ Software Institute, Università della Svizzera italiana (USI), Switzerland
†Microsoft, USA
‡SEMERU @ Computer Science Department, William and Mary, USA



Using Pre-Trained Models to Boost Code Review Automation

Rosalia Tufano
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Simone Masiero
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

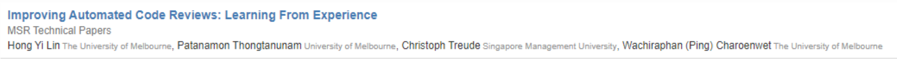
Antonio Mastropaolo
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Luca Pascarella
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

Denys Poshyvanyk
SEMERU @ Computer Science Department
William and Mary
USA

Gabriele Bavota
SEART @ Software Institute
Università della Svizzera italiana
Switzerland

- ICSE-2022



Improving Automated Code Reviews: Learning From Experience
MSR Technical Papers
Hong Yi Lin The University of Melbourne, Patanamon Thongtanunam University of Melbourne, Christoph Treude Singapore Management University, Wachiraphan (Ping) Charoenwet The University of Melbourne

Outline

- Human testing
- Human testing methods
  - Inspections
  - Walkthroughs
  - Pair-programing

Mentimeter

Join at menti.com | use code 3674 6554

Next Lecture

- Testing. Test planning.
- Test case design - Black-box testing

- Testing Management Tool
- Continuous integration - Jenkins

References

- [PY08] M. Pezzand and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. John Wiley and Sons, 2008.
- [Mye04] Glenford J. Myers, *The Art of Software Testing*, John Wiley & Sons, Inc., 2004
- [You79] E. Yourdon, Structured Walkthroughs, Prentice-Hall,Englewood Cliffs, NJ, 1979
- [CB03] Jean-Francois Collard and Ilene Burnstein. *Practical Software Testing*. Springer-Verlag New York, Inc., 2003.
- [Fre10] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010

Software Systems Verification and Validation

“Tell me and I forget, teach me and I may remember, involve me and I learn.”

(Benjamin Franklin)