



Software Reviews

Software Engineering II
WS 2016/17

Arian Treffer
arian.treffer@hpi.de

Prof. Plattner, Dr. Uflacker
Enterprise Platform and Integration Concepts

Review Techniques



“[Formal or informal] meeting during which a software product is [examined by] project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval”
[IEEE1028]

- People-intensive approach instead of using tools

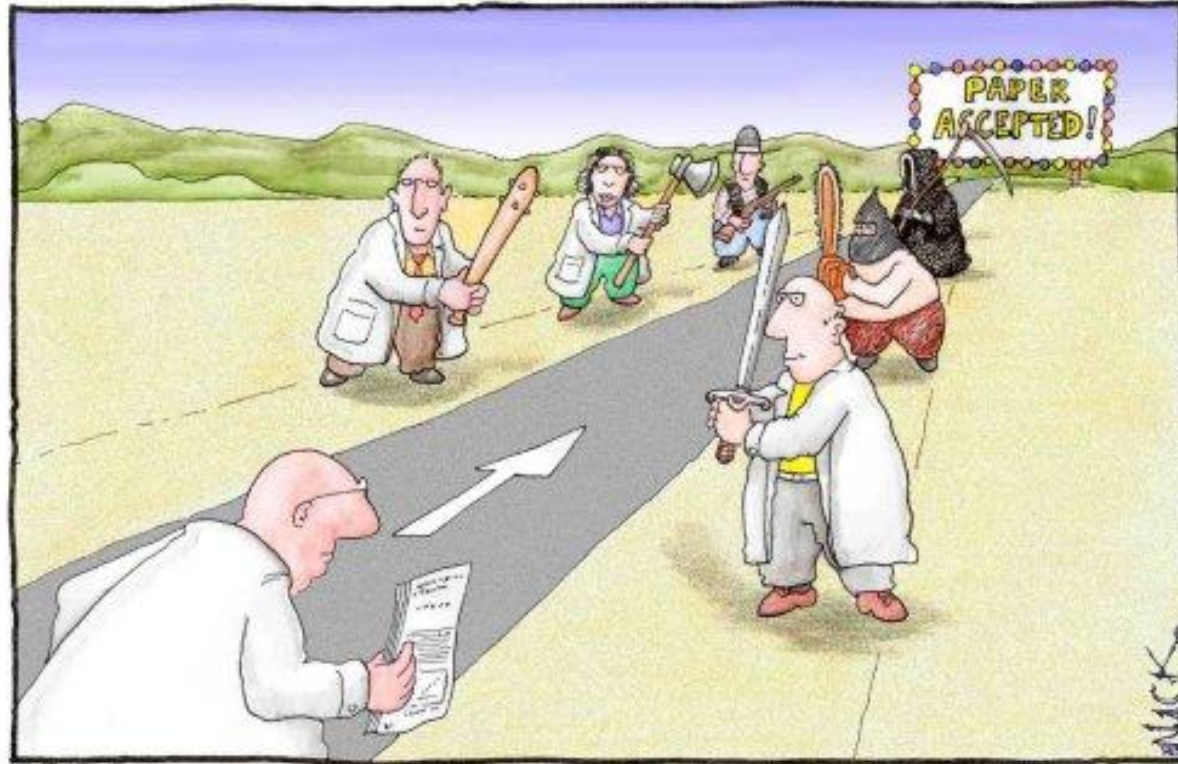
Why Reviews?



[Giese]

- Assure that software fulfills the requirements
- Faults are covered as early as possible
- Projects gets more manageable by identifying new risks
- Improvement of communication
- Further education of participants
- Software gets more visible

Involved Roles



[<http://community.acs.org/journals/acbcct/cs/Portals/0/wiki/PeerReview.jpg>]

Involved Roles



[Giese]



Manager

- Assessment is an important task for manager
- But: Lack of technical understanding
- But: Assessment of a product vs. assessment of a person
- è Outsider in review process, but should support with resources (time, staff, rooms, ...)

Developer

- Should not justify but only explain their results
- **No boss** should take part at review



Review team

[Giese]

Team leader

- Responsible for quality of review
- Technical, personal and administrative competence
- Moderation of review meetings



Reviewer

- Study the material before first meeting
- Don't try to achieve personal targets!
- Gives positive *and* negative comments on review artifacts
 - Not on the author!



Recorder

- Any reviewer, can rotate even in review meeting
- Protocol as basis for final review document



Task of Review Team



[Giese]

Deliver a good review

- “Don’t shoot the messenger”
- Find problems, but don’t try to solve them

Artifact of interest should be assessed

- Accepted, partly accepted, needs corrections, rejected
- Acceptance only possible if no participant speaks against it

Problems should be clearly identified/ extracted

Review Team Building



[Giese]

Team members: In general staff with personal interest in a good result

Review as basis for management decisions

Potential members

- Representative of team which build artifact (not the author!)
- Representative of customer
- Representative of team which will use the artifact
- Representative of QA unit
- Experienced staff or outsiders to ensure objectivity

3-6 members (with some extra viewers)

Management Reviews



“The purpose of a management review is to monitor progress, determine the status of plans and schedules, confirm requirements and their system allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose” [IEEE1028-97]

- Support decisions about changes and corrective actions that are required during a software project
- Determine the adequacy of plans, schedules, and requirements and monitor their progress or inconsistencies

Technical Reviews



“The purpose of a technical review is to evaluate a software product to determine its suitability for its intended use. The objective is to identify discrepancies from approved specifications and standards. The results should provide management with evidence confirming (or not) that the product meets the specifications and adheres to standards, and that changes are controlled” [IEEE1028-97]

- Roles: a decision-maker, a review leader, a recorder, and technical staff to support the review activities
- Inputs: Statement of objectives, a specific software product, the specific project management plan, the issues list associated with this product, the technical review procedure

Inspections

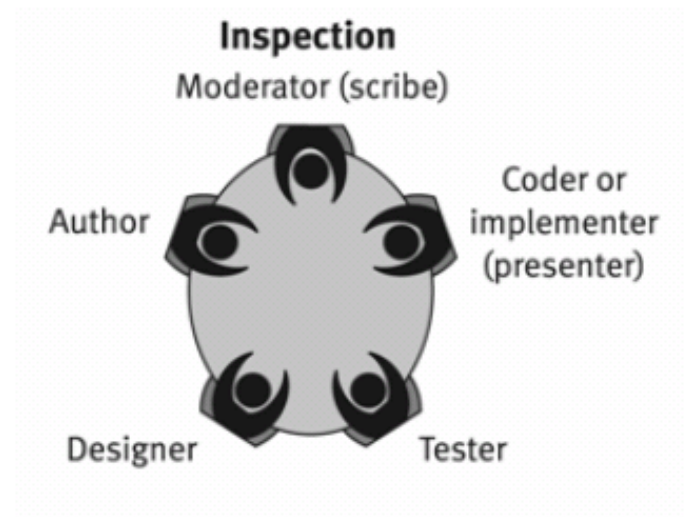
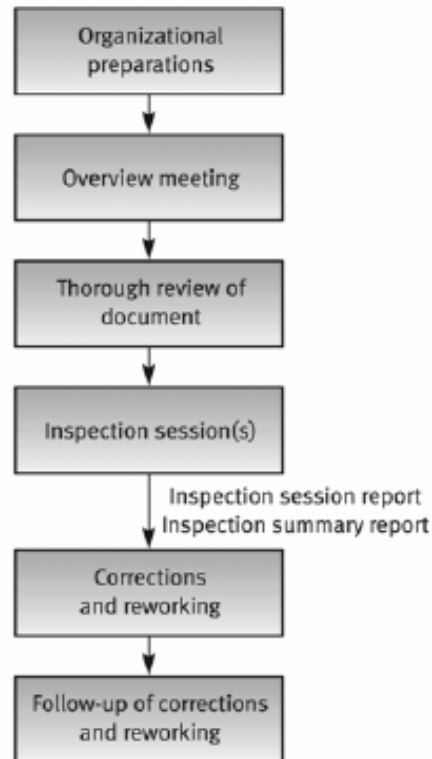


“The purpose of an inspection is to detect and identify software product anomalies” [IEEE1028-97]

- Team members checks the material/ artifacts independently
- Consolidation of results in meeting of team members and developer
- Focus on important parts of software
- Meetings gets more structured/ shorter, but much preparation time for each team member

Inspections - Process and Roles

[Galin2004]



Walk-Throughs

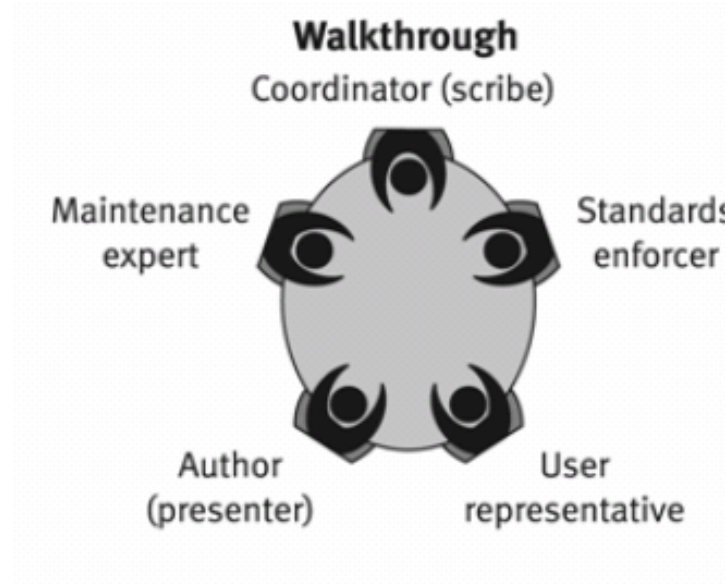
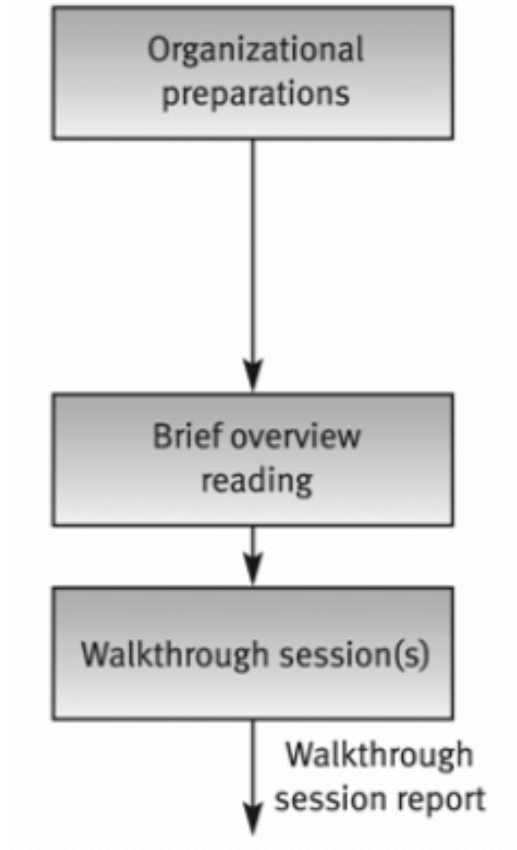


“The purpose of a walk-through is to evaluate a software product. A walk-through may be conducted for the purpose of educating an audience regarding a software product.” [IEEE1028-97]

- Similar to inspection but typically less formally
- Organized by developer/ software engineer for reviewing his own work
- Bigger audience can participate at meeting (e.g. for education purposes)
- Few preparation for team members

Walk-Throughs

[Galvin2004]



What to Review?



[Galín2004]

Should be reviewed	Don't have to be reviewed
Parts with complicated algorithms	Trivial parts where no complications are expected
Critical parts where faults could have bad effects	Parts which won't break the functionality if faults occur
Parts using new technologies/ environment/ ...	Parts which are similar to some which has been reviewed in previous meetings
Parts which has been constructed by inexperienced team members	Reused or redundant parts

Comparison of Review and Audit Types

[Giese, 2012]

Eigenschaft	Formaler technischer Review	Inspektion	Walkthrough	Persönlicher Review
Vortreffen	Nein	Ja	Nein	Nein
Vorbereitung der Teammitglieder	Ja - sehr gründlich	Ja - gründlich	Ja - oberflächlich	Nein
Sitzung	Ja	Ja	Ja	Nein
Nachfolgende Aktivitäten	Ja	Ja	Nein	Nein
Formales Training der Teilnehmer	Nein	Ja	Nein	Nein
Checklisten	Nein	Ja	Nein	Nein
Systematische Erfassung von Fehlern	Nicht formal benötigt	Formal benötigt	Nicht formal benötigt	Nicht formal benötigt
Reviewdokument	Formal design review report	<ol style="list-style-type: none"> 1) Bericht zu den Ergebnissen der Sitzung 2) Zusammenfassung der Sitzung 		

Code Review Tools



Gerrit: <https://code.google.com/p/gerrit/>

- Integrated with Github: <http://gerrithub.io>
- Used by, e.g., Chromium, Eclipse, Qt, Typo3, Wikimedia, etc.

Review Ninja: <http://review.ninja>

- Github integration

FishEye: <https://www.atlassian.com/software/fisheye/overview>

- Visualize, Review, and organize code changes

Conclusion



- Reviews are very effective and efficient techniques
- “Low tech” (without tools)
- Unfortunately, in practice, these techniques aren’t widely-used!

Tools that might help...



- Measured code complexity with Flog
- <http://ruby.sadi.st/Flog.html>

“Flog shows you the most torturous code you wrote. The more painful the code, the higher the score.”

- Example input class and report

```
class Test
  def blah
    a = eval "1+1"
    if a == 2 then
      puts "yay"
    end
  end
end
```

```
Test#blah: (11.2)
 6.0: eval
 1.2: branch
 1.2: ==
 1.2: puts
 1.2: assignment
 0.4: lit_fixnum
```

Tools that might help (2/2)



Find painful parts:

- Flay (structural similarities, <https://rubygems.org/gems/flay>)
- Reek (code smells, <https://github.com/troessner/reek>)
- Cane (code quality, <https://github.com/square/cane>)
- ...
- Metric_fu (combines the above, https://github.com/metricfu/metric_fu/)
- Rails_best_practices (Rails specific, https://github.com/flyerhzm/rails_best_practices)

Find slow parts of your code/tests:

- `rake spec SPEC_OPTS="--profile"`
- Show 10 slowest examples from your test suite

Further Reading



<http://guides.rubyonrails.org>

<http://rails-bestpractices.com/>