HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Software Reviews

Software Engeneering II
WS 2014/15

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]

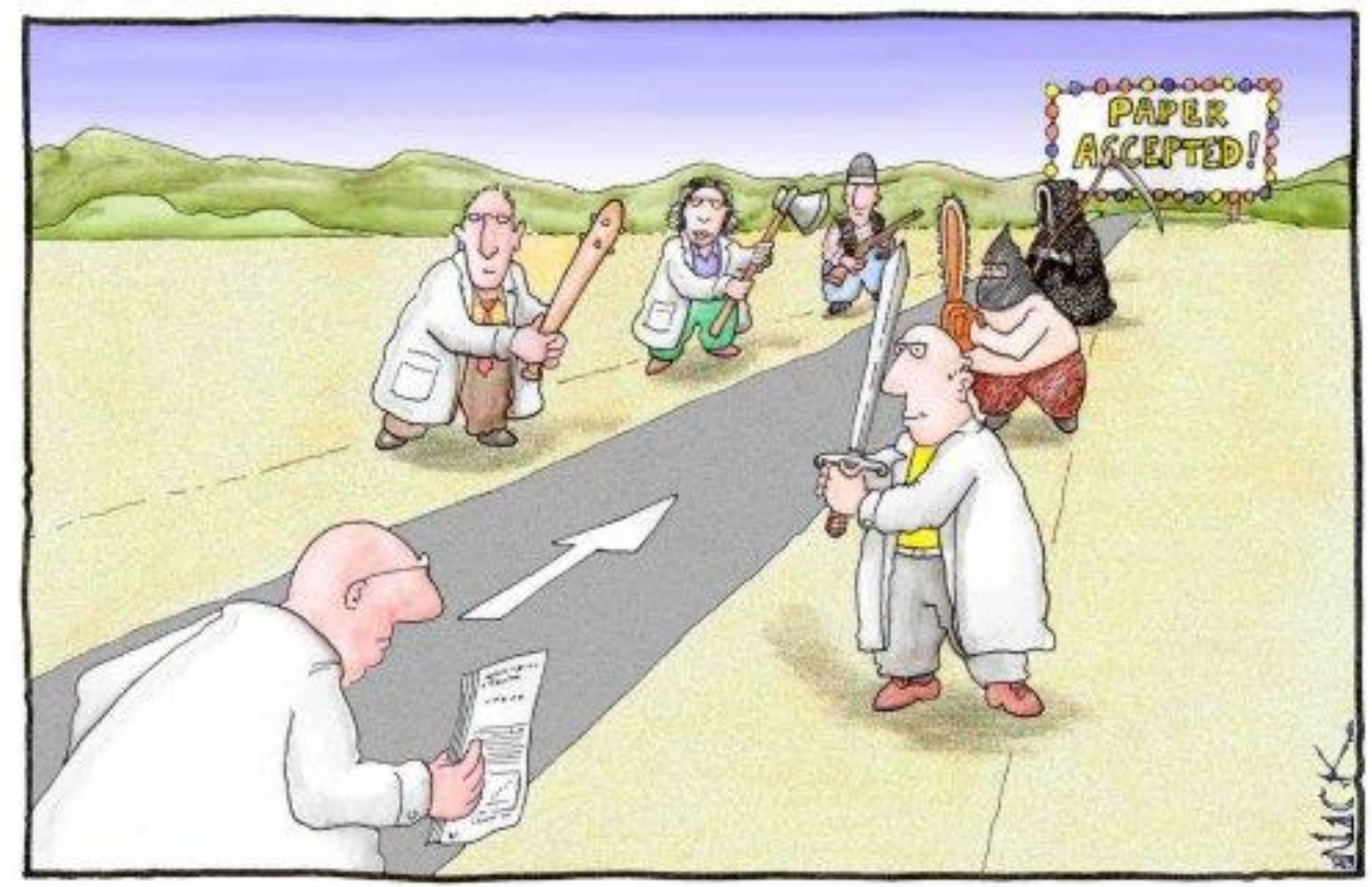
- People-intensive approach instead of using tools

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

➔ 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

Team leader

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

[Giese]

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

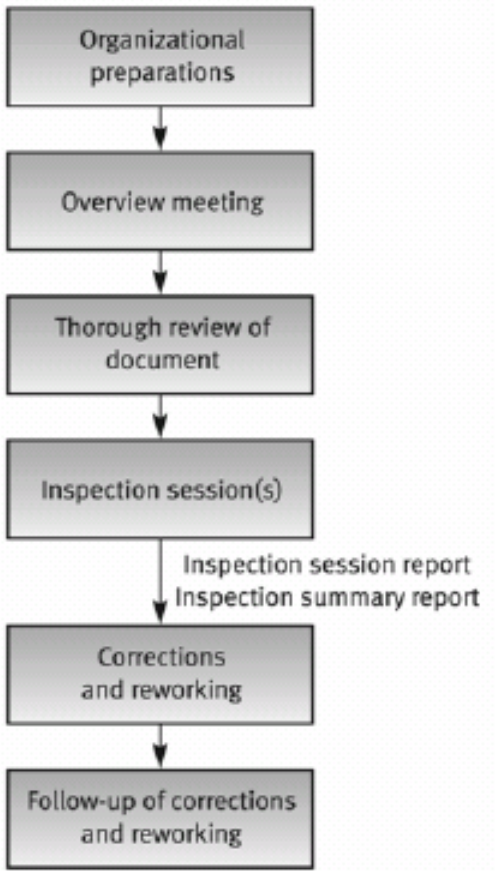
Inspections

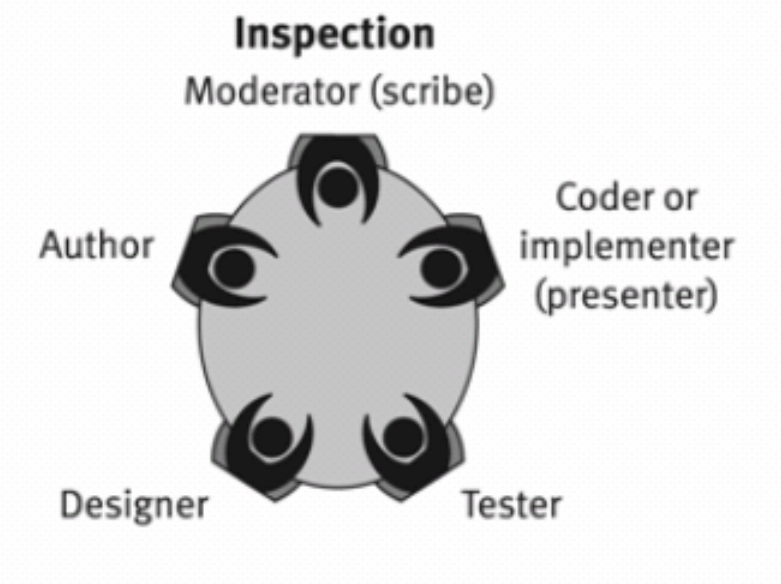
"The purpose of an inspection is to detect and identify software product anomalies" [IEEE1028-97]

- 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]



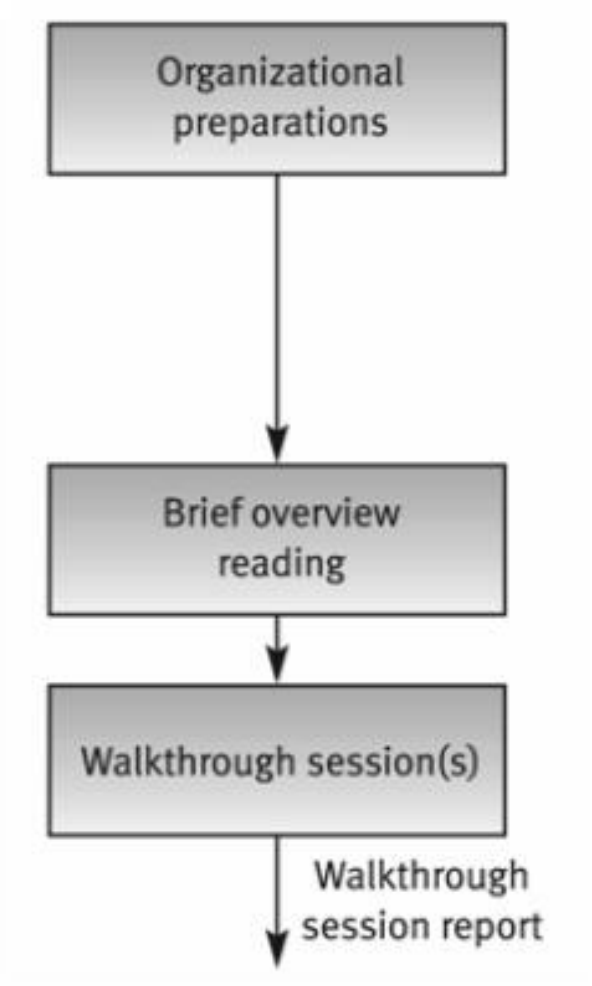


Walk-Throughs

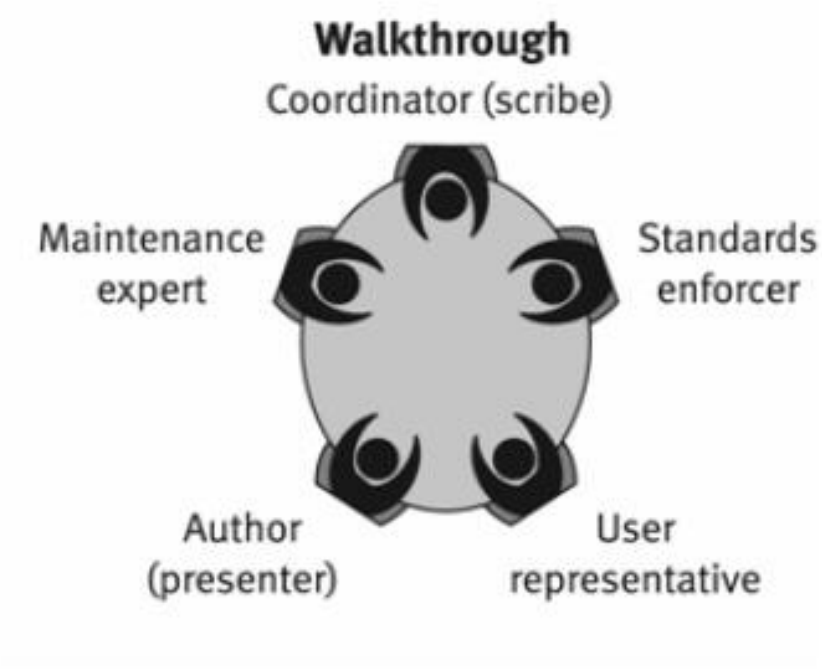
"The purpose of a walk-through is to evaluate a software product. A walk-through may be conducted for the purpose of educating an audience regarding a software product." [IEEE1028-97]

- 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



[Galin2004]



Reviews

[Galin2004]

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	**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

```ruby
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 (structual 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

Code Examples


```
validates_presence_of :last_name
validates_presence_of :source
validates_inclusion_of :potential, :in => 0..100, :message => " ist in % anzugeben und kan
validates_inclusion_of :status, :in => 1..4, :message => ": 1 - offen | 2 - benachrichtigt
validates_format_of :email, :with => /^(|(([A-Za-z0-9]+_+)|([A-Za-z0-9]+\-+)|([A-Za-z0-9]+

def self.newLead (first_name, last_name, source, potential, status, email, adr_street, adr

  if first_name == nil or last_name == nil or first_name == "" or last_name == ""
    return nil
  end
  if source == nil or source == ""
    return nil
  end
  if potential == nil or potential == "" or potential < 0 or potential > 100
    return nil
  end
  if status == nil or status == "" or status < 1 or status > 4
    return nil
  end

  if email != nil and email != "" and (email =~ /^(|(([A-Za-z0-9]+_+)|([A-Za-z0-9]+\-+)|([
    return nil
  end

  lead = Lead.create(:first_name => first_name, :last_name => last_name, :source => source

  return lead
end
```

Problem?

Re-implements Active Record Validation Logic

Acts different than the embodied method

Nightmare to test

Violates Ruby coding conventions

Solution:

- xyz = Lead.new({:first_name => first_name, :last_name => …})
- xyz.valid? => false

Code Examples


```ruby
def getSeller
  seller_list=[]
  for s in Seller.find_by_sql ["SELECT name FROM sellers where id = ?",self.seller_id]
    seller_list << Seller.find(s.attributes["name"])
  end
  return seller_list
end
```

Problem?

- Re-implements Active Record Association Logic
- Min. 2 SQL queries when you already have the desired object…

- Solution:
  - belongs_to :seller

Code Example


```ruby
def SupportTicket.selectClosedTickets
  result = Array.new
  all.each do |ticket|
    if ticket.closed?
      result << ticket
    end
  end
  return result
end
```

Problem?

- Re-implements Active Record Finder Logic
- Major performance issue
- Violates Ruby coding conventions

- Solution:
  - SupportTicket.find_all_by_closed(true)
  - SupportTicket.where(:closed => true)

Code Example


```ruby
class MockupProduct < ActiveRecord::Base
  attr_accessible :discount_range, :name, :price
  has_one :offer
end
```

Problem?

Cluttering the source code with “Mockup Classes” (what’s that anyway?)

Solution:

- Commit dependent classes very early
- fill them with content later
- Predefine interfaces
- Use ambassadors
- Stub the methods that you want to use, not the entire classes

Code Example


```ruby
def getActualDiscount
  @customer = self.opportunity.mockup_customer
  if @customer.discount_class == "A"
  @customer_discount = 30
  end
  if @customer.discount_class == "B"
  @customer_discount = 20
  end
  if @customer.discount_class == "C"
  @customer_discount = 10
  end
  return @customer_discount + self.discount
end
```

Problem?

Code is error prone

At the wrong place

Violates ruby coding conventions

- Camelcase methods
- 2 whitespaces indent per level

Solution:

- Test with uncommon values (“D”)
- Suggestion: Move it somewhere else -> Customer?

Code Example


```
def e_r_s (s)
  if s == nil
  return ""
  else
  return s
  end
end
```

Problem?

Self-explanatory method and variable names?

Indent?

Solution:

- Why not use ruby standard functionality
- return s || “”
- return s.nil? ? “” : s

Code Example


```ruby
it "should belong to a customer" do
  customer = Factory.build(:customer)
  @campaign_response.customer = customer
  @campaign_response.customer.should == customer
end
```

Problems?

…

Solution – At least do something with that customer…

Further Reading

http://guides.rubyonrails.org

http://rails-bestpractices.com/