Software Reviews

Software Engineering II
WS 2017/18

Christoph Matthies (christoph.matthies@hpi.de)
Tobias Dürschmid (tobias.duerschmid@student.hpi.de)

Enterprise Platform and Integration Concepts
Review Definition

“[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]

- Generate comments on software
- Several sets of eyes check
- People instead of using tools
Reviews Motivation

- Improve code
- Discuss alternative solutions
- Transfer knowledge
- Find defects

[Bacchelli ‘13]
Involved Roles

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

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
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
Types of Reviews [IEEE1028-97]

Management Review
- Monitor progress and status of plans, confirm requirements
- Evaluate effectiveness of management approaches / corrective actions

Technical Review
- Evaluate entire software on suitability for intended use
- Provide evidence whether software product meets specifications
Types of Reviews [IEEE1028-97]

**Inspections**
- Identify software product anomalies, invented at IBM in the 1970’s
- **Formal process**, can involve hard copies of the code and documents
- Review team members check important artifacts independently, consolidation meeting with developers
- Preparation time for team members, shorter meetings

**Walk-through**
- Evaluate software, focus on educating an audience
- Organized by developer for reviewing own work
- Bigger audience can participate, little preparation for team members
### What to Review?

<table>
<thead>
<tr>
<th>Should be reviewed</th>
<th>Might not have to be reviewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts with complicated algorithms</td>
<td>Trivial parts where no complications are expected</td>
</tr>
<tr>
<td>Critical parts where faults lead to system failure</td>
<td>Parts which won’t break the functionality if faults occur</td>
</tr>
<tr>
<td>Parts using new technologies / environment / ...</td>
<td>Parts which are similar to those previously reviewed</td>
</tr>
<tr>
<td>Parts constructed by inexperienced team members</td>
<td>Reused or redundant parts</td>
</tr>
</tbody>
</table>

[Galin2004]
## Comparison of Review Types

<table>
<thead>
<tr>
<th>Eigenschaft</th>
<th>Formaler technischer Review</th>
<th>Inspektion</th>
<th>Walkthrough</th>
<th>Persönlicher Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vortreffen</td>
<td>Nein</td>
<td>Ja</td>
<td>Nein</td>
<td>Nein</td>
</tr>
<tr>
<td>Vorbereitung der Teammitglieder</td>
<td>Ja - sehr gründlich</td>
<td>Ja - gründlich</td>
<td>Ja - oberflächlich</td>
<td>Nein</td>
</tr>
<tr>
<td>Sitzung</td>
<td>Ja</td>
<td>Ja</td>
<td>Ja</td>
<td>Nein</td>
</tr>
<tr>
<td>Nachfolgende Aktivitäten</td>
<td>Ja</td>
<td>Ja</td>
<td>Nein</td>
<td>Nein</td>
</tr>
<tr>
<td>Formales Training der Teilnehmer</td>
<td>Nein</td>
<td>Ja</td>
<td>Nein</td>
<td>Nein</td>
</tr>
<tr>
<td>Checklisten</td>
<td>Nein</td>
<td>Ja</td>
<td>Nein</td>
<td>Nein</td>
</tr>
<tr>
<td>Systematische Erfassung von Fehlern</td>
<td>Nicht formal benötigt</td>
<td>Formal benötigt</td>
<td>Nicht formal benötigt</td>
<td>Nicht formal benötigt</td>
</tr>
<tr>
<td>Reviewdokument</td>
<td>Formal design review report</td>
<td>1) Bericht zu den Ergebnissen der Sitzung</td>
<td>2) Zusammenfassung der Sitzung</td>
<td></td>
</tr>
</tbody>
</table>
Modern Code Reviews

- Follows more lightweight, flexible process
- Change sizes are smaller
- Performed regularly and quickly,
  mainly just before code committed to main branch

- Shift from defect finding to group problem solving activity
- Prefer discussion and fixing code over reporting defects

[Rigby’13]
[Bacchelli’13]
Recent Research

- Code review coverage and review participation share significant link with **software quality**
- Most comments concern code improvements, understandability, social communication
- Only ~15% of comments indicate possible defect
- Developers spend approximately five hours per week (10-15% of their time) in code reviews
Recent Research

Expectations

Outcomes

Maintainability and code improvements identified as most important aspects of modern code reviews

[Bosu’17]
Challenges of the Review Process

- **Delay** the use of implemented features
- **Forces** the reviewers to **switch context** away from their current work
- **Offer little feedback** for **legacy code**

- **Overloading** (too many files), developers create large patches

- **Overcrowding** (too many reviewers), assigning too many reviewers may lower review quality
Post-commit Code Review

- Review after committing to VCS (e.g. using pull requests)
- Used by most of projects on GitHub and BitBucket

+ Developers can commit continuously
- Other team members see code changes and can adapt their work
- Flexible definition of the code to be reviewed (set of commits, whole branch, some files)

- Chance of unreviewed code in main repository
- Requires branches to work effectively
- May take a while for developers to come back to the code and improvement ideas

https://www.devart.com/review-assistant/learnmore/pre-commit-vs-post-commit.html
Pre-commit Code Review

- Review before committing to the version control (e.g. using mailing lists / Gerrit, Crucible tools)
- Used by Linux Kernel, Git, Microsoft, Google, and Facebook

+ Ensures review was performed
- Code quality standards met before commit
- No repository access for reviews
- Other developers not affected by found bugs

- Decreased productivity due to overhead
  - Further work on submitted code not possible until review done
- Review and commit are not tightly coupled
Reviewer Assignment

- Usually, two reviewers find an optimal number of defects.
- People who contributed changes (find defects)
- New developers (transfer knowledge)
- Team members with a small review queue
- Reviewers with different fields of expertise
- Let reviewers know what they should look out for

[Source: Rigby'13]
Maximize Usefulness

- "Ask a programmer to review 10 lines of code, he'll find 10 issues. Ask him to do 500 lines and he'll say it looks good." - Giray Özil

- Semantically coherent set of changes easier to review than interleaved concerns
Code Review In Industry

**Microsoft**
- Median completion times: 14.7h (Bing), 18.9h (Office), 19.8h (SQL Server)
- Median number of reviewers: 3-4
- Developers spend 4-6 hours per week on reviews

**Google**
- Mandatory review of every change
- Median completion times: 15.7h (Chrome), 20.8h (Android)
- Median patch size: 78 lines (Chrome), 44 lines (Android)
- Median number of reviewers: 2

[Rigby’13]
Code Review Tools

**Gerrit** ([https://code.google.com/p/gerrit/](https://code.google.com/p/gerrit/))
- Integrated with Github: [http://gerrithub.io](http://gerrithub.io)
- Used by, e.g., Chromium, Eclipse, Qt, Typo3, Wikimedia, etc.
- Plug-ins available (e.g. EGerrit for Eclipse)

**Review Ninja** ([http://review.ninja](http://review.ninja))
- Github integration

**FishEye** ([https://www.atlassian.com/software/fisheye/overview](https://www.atlassian.com/software/fisheye/overview))
- Visualize, Review, and organize code changes
Tools

- Testing checks code function via dynamic analysis
- Code reviews manually check code quality via static analysis

Automated static analysis (linters) can help as well

- SimpleCov (code coverage, https://github.com/colszowka/simplecov)
- Reek (code smells, https://github.com/troessner/reek)
- Cane (code quality, https://github.com/square/cane)
- Rails_best_practices (Rails specific, https://github.com/flyerhzm/rails_best_practices)
Reviews are not a new thing, good reasons to do them

Different types of review techniques
- Management Review
- Technical Review
- Inspection
- Walk-through
- Modern / contemporary code reviews

Method to find faults and improvement opportunities early in the process
```python
def self.human_attribute_name(*args):
    if args[0].to_s == "start_date"
        return "Anfangs-Datum"
    elif args[0].to_s == "end_date"
        return "End-Datum"
end

# NOTE: In our quest for 100% code coverage we can't have this line.
# If anyone is to add a new attribute that uses the default label,
# reenable this line.
# super
end```

Code Examples
Should `super` be there or not?
- If yes, test it!

Better
- Don’t override Rails core methods
- Use proper i18n
describe "POST #create" do
  context "with valid params" do
    it "creates a new Profile" do
      sign_in FactoryGirl.create(:user)
      expect {
        post :create, profile: valid_attributes, session: valid_session
      }.to change(Profile, :count).by(1)
      end
      
    it "assigns a newly created profile as @profile" do
      sign_in FactoryGirl.create(:user)
      post :create, profile: valid_attributes, session: valid_session
      expect(assigns(:profile)).to be_a(Profile)
      expect(assigns(:profile)).to be_persisted
      end
      
    it "redirects to the created profile" do
      sign_in FactoryGirl.create(:user)
      post :create, profile: valid_attributes, session: valid_session
      expect(response).to redirect_to(Profile.last)
      end
    end
  end
  
  context "with invalid params" do
    it "assigns a newly created but unsaved profile as @profile" do
      sign_in FactoryGirl.create(:user)
      post :create, profile: invalid_attributes, session: valid_session
      expect(assigns(:profile)).to be_a(Profile)
      end
      
    it "re-renders the 'new' template" do
      sign_in FactoryGirl.create(:user)
      post :create, profile: invalid_attributes, session: valid_session
      expect(response).to render_template("new")
    end
  end
end
before(:each)
Code Examples

# POST /chair_wimi
# POST /chair_wimi.json

def create
    @chair_wimi = ChairWimi.new
    @chair_wimi.chair_id = params[:chair]
    @chair_wimi.user_id = params[:user]

    @chairapp = ChairApplication.find_by!(:user_id => params[:user], :chair_id => params[:chair])
    @chairapp.status = "accepted"
    @chairapp.save

    @user = User.find(params[:user])
    @user.role = "wimi"
    @user.save
Problem?

Parameters don’t match **params**

Business logic vs controller logic
- chair.add_wimi
- chair_application.accept!
Code Examples

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

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]+\+\-\+\d)/;([A-Za-z0-9]+\+\-\+\d)]([A-Za-z0-9]+\+\-\+\d)/
    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
Hard to test

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

Solution:

- `belongs_to :seller`
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)`
```python
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
Violates ruby coding conventions
- Camelcase methods
- Indentations
- Superfluous instance variable assignments

Solution:
- Test with uncommon values (“D”)
- Suggestion: Move it somewhere else -> Customer?
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.nil? ? "" : s
it "should belong to a customer" do
  customer = Factory.build(:customer)
  @campaign_response.customer = customer
  @campaign_response.customer.should == customer
end
Problems?

Solution:
- Do something with the customer
References


Image Sources

- "ScientificReview" by Center for Scientific Review
  Licensed under Public Domain via Wikimedia Commons
  http://commons.wikimedia.org/wiki/File:ScientificReview.jpg

- "WTF per Minute" by Glen Lipka
  http://commadot.com/wtf-per-minute/

- "The Dark Side of Infrastructure as Code" by Lori Macvittie

- Geek & Poke