Software Reviews

Software Engineering II
WS 2018/19

Enterprise Platform and Integration Concepts

Christoph Matthies
christoph.matthies@hpi.de

"Puzzle piecing a software suite" Juhan Sonin (CC BY 2.0), https://www.flickr.com/photos/juhansonin/12659667364/
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]

Code Quality Measurement: WTFs/Minute

http://commedot.com
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
Task of Review Team

**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 2) Zusammenfassung der Sitzung</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Giese, 2012]
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]
Code Review Hierarchy of Needs

- Findings bugs vs. understanding code
- Building a shared mental model
- Ensuring sane design
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

[Bosu’17] [McIntosh’14] [Bacchelli ‘13]
Recent Research

Expectations

- Finding defects
- Code Improvement
- Alternative Solutions
- Knowledge Transfer
- Team Awareness
- Improving Dev Process
- Share Code Ownership
- Avoid Build Breaks
- Track Rationale
- Team Assessment

Outcomes

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

Expectations 4 years later

- Maintainability
- Knowledge sharing
- Functional defects
- Community building
- Minor Errors, Typos
- Other

Percentage of respondents

Comments in each Category

- Code Improvement
- Understanding
- Social Communication
- Defects
- External Impact
- Testing
- Review Tool
- Knowledge Transfer
- Misc

Percentage of Comments

[Bacchelli ‘13]

[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 (pull requests are one way of doing this)
- Used by most projects on GitHub and BitBucket

- Developers can commit continuously
- Other team members see code changes in VCS 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
  - Need to / can set restrictions
- Requires branches or similar 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 version control system (e.g. using mailing lists / Gerrit, Crucible tools)
- Used by e.g. Linux Kernel, Git, Google

+ No code enters unreviewed
- Code quality standards met before commit, no 'fixes'
+ No repository access needed for reviews
- Other developers definitely not affected by bugs in reviewed code

- Reviewing all code takes time
  - Deciding what needs a review takes time too
- Possibly another complex system to handle
  - Might not want to work on submitted code until review done (e.g. mailing list)
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

_How to Make a Good Code Review_

1. Rule 1: Try to find at least something positive
2. At least we don’t need to obfuscate it before shipping

[ 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)
Summary

- 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
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
Problems?

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 bc_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 bc_a_new(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")
Problems?

before(:each)
# POST /chair_wimis
# POST /chair_wimis.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!
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

Solution:
- lead = Lead.new({ first_name: first_name, last_name: ... })
- lead.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)`
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
- Ternary operator
- 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:
- Do something with the customer
```ruby
# GET /events/1/ranking

def ranking
  # Array of RankingEntry Structs that gets sorted when filled completely
  @ranking_entries = []

  # Leaves the Array of RankingEntry Structs empty when no teams participate in the event
  @event.teams.each do |team|
    ranking_entry = RankingEntry.new(nil, team.name, 0, 0, 0, 0, 0, 0, 0, 0)
    event_matches = @event.matches
    # Considers only the team's home matches that belong to the event
    home_matches_in_event = team.home_matches & event_matches
    parse_matches_data_into_ranking_entry team, ranking_entry, home_matches_in_event, :parse_match_details_for_home

    # Considers only the team's away matches that belong to the event
    away_matches_in_event = team.away_matches & event_matches
    parse_matches_data_into_ranking_entry team, ranking_entry, away_matches_in_event, :parse_match_details_for_away

    ranking_entry.goals_difference = ranking_entry.goals - ranking_entry.goals_against
    @ranking_entries.push ranking_entry
  end

  # Sorts the RankingEntries in the following order:
  # 1. DESCENDING by points
  # 2. DESCENDING by goals
  # 3. ASCENDING by name
  @ranking_entries = @ranking_entries.sort_by { | ranking_entry | [-ranking_entry.points, -ranking_entry.goals, ranking_entry.name] }

  # Adds a rank to each RankingEntry based on its position in the Array
  @ranking_entries.each_with_index do |ranking_entry, index|
    ranking_entry.rank = index + 1
  end
end
```
Problem?

Looks complicated

- Slim controller?
- Small methods!
- Custom Route (No REST)

Solution:

- Create a PORO (Plain old ruby object)
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