On to code review: Lessons learned @ Microsoft

Michaela Greiler
Team: Tools for Software Engineers (TSE)

• Build
• Code review
• Testing
• Static analysis
• Branch health
• Software Development Practices
• Empirical Software Engineering
Tools for Software Engineers(*)

- TSE’s mission is to **shorten the development cycle without impacting quality**

- TSE’s **customers are** developers

(*) [http://research.microsoft.com/tse](http://research.microsoft.com/tse)
Code reviewing @Microsoft

• Do code reviews make a difference?
• How do we ensure an effective and efficient review process?
• Does each code change need a review?
• Can we automate parts of code reviews?

MSR Redmond/TSE: Michaela Greiler Jacek Czerwonka Wolfram Schulte Suresh Thummalapenta Kim Herzig

MSR Redmond: Christian Bird Nachi Nagappan Thomas Zimmermann

MSR Cambridge: Brendan Murphy
Code Review is Everywhere
Zoom in @ Microsoft
Code Reviews in Engineering Workflow

INNER LOOP

Pre-checkin Validation
- Private test env
- Functional tests
- Static analysis

Coding
- Version control
- Branch/Packages
- IDE & Dev Tools

Code Review
- Select reviewers
- Review
- Check-in

Check-in

OUTER LOOP

Rolling/Official Build
- Build verification
- Unit testing

E2E Validation
- Pre-prod deploy
- Scenario tests
- Build drop

Deploy to Production
- Deployment targets

Production Validation
- Watchdogs
- Flighting

Insights
Code review process
CodeFlow
Common code review steps:

- **Preparation** of the code to be reviewed
- **Selection of reviewers**
- **Notification** of the selected reviewers or stakeholders
- **Feedback** provided by reviewers
- **Iteration** involving communication between the author and reviewer
- **Signoff** by reviewers
- **Check-in** of the code change to the target system
<Build Include="$(SrcRoot)\Common\Database\Schemas\loader\Stored Procedures\GetInvalidConstraints.proc.sql">
</Build>

<Build Include="$(SrcRoot)\Common\Database\Schemas\loader\Tables\Constraints.table.sql">
</Build>

<Build Include="$(SrcRoot)\Common\Database\Schemas\loader\Tables\Keys\Constraints.PK_Constraints.pkey.sql">
</Build>

<Build Include="$(SrcRoot)\Common\Database\Schemas\loader\Tables\ProcessingStatus_QBuild.table.sql" />
</ItemGroup>
</Build>
</ItemGroup>

<Build Include="$(SrcRoot)\Common\Database\Schemas\Configuration\Stored Procedures\RebuildIndexes.proc.sql">
</Build>

<Build Include="$(SrcRoot)\Common\Database\Schemas\Maintenance\Tables\Updates.table.sql">
</Build>

<Build Include="$(SrcRoot)\Common\Database\Schemas\Maintenance\Tables\Keys\Updates.PK_Updates.pkey.sql">
</Build>

</ItemGroup>
</ItemGroup>

<None Include="Schema Comparisons\LiveChanges.scmp" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsMSG.Data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsWindows.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsWindowsPhone.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsWindowsServices.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsOffice.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsExchange.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsSync.data.sql" />
</None>

<None Include="Post-Deployment\Data\dbo.DepotsOfficeMac.data.sql" />
</None>
Viewing Pending Changes

Review Title
Adding support for new types of builds

Required Reviewers
Recommended reviewers were added as required reviewers

Work Items/Bugs
Michaela Greiter

Optional Reviewers

Description
New feature enabling saving data from new types of builds

305 <Build Include="$(SrcRoot)\Common\Database\Schemas\Loader\Stored Procedures\GetInvalidConstraints.proc.sql"/>
306 <Link>Schemas\Loader\Stored Procedures\GetInvalidConstraints.proc.sql</Link>
307 <Build Include="$(SrcRoot)\Common\Database\Schemas\Loader\Tables\Constraints.table.sql"/>
308 <Link>Schemas\Loader\Tables\Constraints.table.sql</Link>
309 <Build Include="$(SrcRoot)\Common\Database\Schemas\Loader\Tables\Keys\ConstraintsPK_Constraints.pkey.sql"/>
310 <Link>Schemas\Loader\Tables\Keys\ConstraintsPK_Constraints.pkey.sql</Link>
311 <Build Include="$(SrcRoot)\Common\Database\Schemas\Loader\Tables\ProcessingStatus_QBuild.table.sql"/>
312 <Link>Schemas\Loader\Tables\ProcessingStatus_QBuild.table.sql</Link>
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
This seems to refer to the wrong location. Shouldn't it be "Schemas\Loader\Tables\QBuildsTable.sql"?
Code Review at Microsoft is Big

• Great tool experience
  • Viral adoption
  • High user satisfaction
  • Natural tool consolidation
  • Data consolidation
• 5.5M+ Code Reviews by all products since 2010
• Engineers spend more than 30 minutes a day in code reviews
Code Reviews Are Not Free

Potential Benefits
• More maintainable designs
• More consistent code base
• Knowledge sharing
• Better awareness of changes
• Additional defects found

Potential Costs
• Time spent by reviewers
• Time spent by author addressing feedback
• Time spent by change waiting in process
Monitoring and improvement

• Several questions arise:
  • Is the time reviewing wise spend?
  • Do we find bugs in code reviews?
  • What is useful code review feedback?
  • Which issues are found during code review?
  • How long do engineers spend on a code review?
  • Which circumstances lead to more effective code reviews?
  • …

• Drive tool improvement
  • New features
  • Tool integration
  • …
Seeking answers

• Interview engineers
• Perform ethnographical observations
• Survey the engineers
• Look at the data traces left by engineers
Mixed Method Research

Is a research approach or methodology

• for questions that call for real-life contextual understandings;

• employing **rigorous quantitative research** assessing magnitude and frequency of constructs and

• **rigorous qualitative research** exploring the meaning and understanding of constructs;

*All methods are inherently flawed!*
Data (trace) collection
What is CodeMine? What data does CodeMine have?
Established in June 2012 as “Garage Project”
- 200GB Raw (compressed)
- 500GB Curated (compressed)
- > 5000 Projects
- > 5M Reviews by 50K Distinct Authors
- > 8M Iterations
- > 8M Review Participants
- > 15M Comment Threads

- Running in Microsoft Azure
- Won 3 internal engineering awards
- Funded by Tools for Software Engineers as part of Codemine

More information? See article: Lessons Learned from Building and Deploying a Code Review Analytics Platform
(Christian Bird, Trevor Carnahan, Michaela Greiler)
CodeFlow Analytics Project Leaderboard
A few studies on code reviews
Characteristics of Useful Code Reviews: An Empirical Study at Microsoft

Amiangshu Bosu, Michaela Greiler, Christian Bird
“Was this an impactful review, a useful comment? You know, not just a comment, but did it result in a change that wouldn’t have been there before.”

- Development Team Manager
Study Goals

• Identify factors that impact feedback usefulness
  • Question: What makes a code review comment useful?

• Recommendations for effectiveness improvements
  • What factors are related to usefulness feedback?
Methodology: Three Phases

- Interview Developers
- Build Comment Classifier
- Quantitatively Investigate Factors
Phase 1: Interview Developers

- Asked author what makes code review comments useful
- Asked code review author to rate 20-25 comments
Which comments are useful?

**Useful:**
- Pointing out defects
- Validation issues
- Info about execution environments
- API/tool suggestions
- Team conventions
- Designs to follow

**Somewhat Useful:**
- Code formatting
- Typos
- Questions that make code more comprehensible

**Not Useful:**
- Comments not in the changed code
- Praise
- Future work/Not immediately actionable
Phase 2: Build Comment Classifier

Positive sentiment?

Only one comment in thread?

Change within one line?

Is status won’t fix?

Is status won’t fix?

Third iteration?

Is status closed?

Not Useful

Useful

Not Useful

Useful

Not Useful

Useful

85% Recall

89% Precision
Phase 3: Investigated Factors of review effectiveness

• Classified 1.5 million reviews

• Related comment usefulness to:
  • Characteristics of the reviewer
  • Characteristics of the team
  • Characteristics of the change under review
Phase 3: Investigated Factors of review effectiveness

• After four reviews of a file, a developer’s knowledge reaches mastery and plateaus
• It takes between six months and a year to become a high quality reviewer at MS
• Quality of feedback degrades when there are more twenty files in a change

More results in the paper
Recommendations

• Include new developers for training
• Include experienced developers for useful feedback
• Select reviewers based on their past history with the changed files
• Avoid submitting large changes for review
Why do Code Review?

“The human eye has an almost infinite capacity for **not seeing what it does not want to see** [...] Programmers, if left to their own devices will ignore the most glaring errors in the output -- errors that **anyone else can see in an instant.”**

Weinberg
Code Reviews Do Not Find Bugs:
How the Current Code Review Best Practice Slows Us Down
Jacek Czerwonka, Michaela Greiler, Jack Tilford
Why Code Review?

<table>
<thead>
<tr>
<th>Reason for Code Reviewing</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code improvement</td>
<td>1</td>
</tr>
<tr>
<td>Find defects</td>
<td>2</td>
</tr>
<tr>
<td>Increase knowledge transfer</td>
<td>3</td>
</tr>
<tr>
<td>Find alternative solutions</td>
<td>4</td>
</tr>
<tr>
<td>Improve the development process</td>
<td>5</td>
</tr>
<tr>
<td>Avoid breaking builds</td>
<td>6</td>
</tr>
<tr>
<td>Build team awareness</td>
<td>7</td>
</tr>
<tr>
<td>Shared code ownership</td>
<td>8</td>
</tr>
<tr>
<td>Team assessment</td>
<td>9</td>
</tr>
</tbody>
</table>
What do we achieve in practice?
## Study: Comments Classification

<table>
<thead>
<tr>
<th>Category</th>
<th>Types of issues included</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Comments, naming, style</td>
<td>22.3%</td>
</tr>
<tr>
<td>Organization of code</td>
<td>Modularity, location of artifacts, new class, duplicate code, size of methods</td>
<td>15.9%</td>
</tr>
<tr>
<td>Solution approach</td>
<td>Alternate algorithm or data structure</td>
<td>8.5%</td>
</tr>
<tr>
<td>Validation</td>
<td>Lack of or improper validation</td>
<td>7.2%</td>
</tr>
<tr>
<td>Visual representation</td>
<td>Beautification, indentation, blank linkees</td>
<td>6.4%</td>
</tr>
<tr>
<td>False positive</td>
<td>Not an real issue</td>
<td>4.6%</td>
</tr>
<tr>
<td>Defect</td>
<td>Incorrect implementation or missing functionality</td>
<td>2.6%</td>
</tr>
<tr>
<td>Logical</td>
<td>Control flow or logic issues</td>
<td>2.3%</td>
</tr>
<tr>
<td>Support</td>
<td>Configuration support systems or libraries</td>
<td>2.0%</td>
</tr>
<tr>
<td>Interface</td>
<td>Interactions with other components</td>
<td>1.5%</td>
</tr>
<tr>
<td>Resources</td>
<td>Resource initialization, manipulation, and release</td>
<td>1.3%</td>
</tr>
<tr>
<td>Timing</td>
<td>Thread synchronization, races</td>
<td>0.3%</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>25.1%</td>
</tr>
</tbody>
</table>


By: Amiangshu Bosu (U of Alabama), Michaela Greiler (TSE), Christian Bird (Microsoft Research Redmond), Characteristics of Useful Code Reviews: An Empirical Study at Microsoft (MSR 2015)
## Study: Code Review Usefulness

<table>
<thead>
<tr>
<th>Category</th>
<th>Types of issues included</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td>Comments, naming, style</td>
<td>22.3%</td>
</tr>
<tr>
<td>Organization of code</td>
<td>Modularity, location of artifacts, new class, duplicate code, size of methods</td>
<td>15.9%</td>
</tr>
<tr>
<td>Solution approach</td>
<td>Alternate algorithm or data structure</td>
<td>8.5%</td>
</tr>
<tr>
<td>Validation</td>
<td>Lack of or improper validation</td>
<td>7.2%</td>
</tr>
<tr>
<td>Visual representation</td>
<td>Beautification, indentation, blank linkes</td>
<td>6.4%</td>
</tr>
<tr>
<td>False positive</td>
<td>Not an real issue</td>
<td>4.6%</td>
</tr>
<tr>
<td>Defect</td>
<td>Incorrect implementation or missing functionality</td>
<td>2.6%</td>
</tr>
<tr>
<td>Logical</td>
<td>Control flow or logic issues</td>
<td>2.3%</td>
</tr>
<tr>
<td>Support</td>
<td>Configuration support systems or libraries</td>
<td>2.0%</td>
</tr>
<tr>
<td>Interface</td>
<td>Interactions with other components</td>
<td>1.5%</td>
</tr>
<tr>
<td>Resources</td>
<td>Resource initialization, manipulation, and release</td>
<td>1.3%</td>
</tr>
<tr>
<td>Timing</td>
<td>Thread synchronization, races</td>
<td>0.3%</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>25.1%</td>
</tr>
</tbody>
</table>


Which challenges do developer face?
Observations, Interviews and Survey
Observations, Interviews and Survey

Observations & Interviews

• 4 teams
• 18 develops/managers
• Interviews during or shortly after code review (situated insights)
• Witness interactions not visible in the trace data
• Highlighting cultural and social issues
Observations, Interviews and Survey

**Observations & Interviews**
- 4 teams
- 18 develops/managers
- Interviews during or shortly after code review (situated insights)
- Witness interactions not visible in the trace data
- Highlighting cultural and social issues

**Follow-up Survey**
- Total Respondents: 911/4,300
- Years of Experience in the Software Industry:
  - 2+: 87%
  - 6+: 70%
  - 10+: 40%
- Time Employed at Microsoft
  - 2+ years: 72%
  - 6+ years: 43%
  - 10+ years: 17%
- Experience Code Reviewing
  - 2-5 years: 80.3%
  - 10+: 22%
Challenges
Getting feedback in a timely manner

“Usually you write up some code and then you send it out for review, and then about a day later you ping them to remind them... and then about half a day later you go to their office and knock on their door.” (Participant 7)
Large reviews & understanding motivation and purpose of code

“It's just this big incomprehensible mess... then you can't add any value because they are just going to explain it to you and you're going to parrot back what they say." (Participant 13)
<table>
<thead>
<tr>
<th>Challenge during Code Reviewing</th>
<th>Overall Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving feedback in a timely manner</td>
<td>1</td>
</tr>
<tr>
<td>Review size</td>
<td>2</td>
</tr>
<tr>
<td>Managing time constraints</td>
<td>3</td>
</tr>
<tr>
<td>Understanding the code's purpose</td>
<td>4</td>
</tr>
<tr>
<td>Understanding the motivations for the change</td>
<td>5</td>
</tr>
<tr>
<td>Obtaining insightful feedback</td>
<td>6</td>
</tr>
<tr>
<td>Bikeshedding (disputing minor issues)</td>
<td>7</td>
</tr>
<tr>
<td>Understanding how the change was implemented</td>
<td>8</td>
</tr>
<tr>
<td>Maintaining code quality</td>
<td>9</td>
</tr>
<tr>
<td>Reaching consensus</td>
<td>10</td>
</tr>
<tr>
<td>Finding relevant documentation</td>
<td>11</td>
</tr>
<tr>
<td>Managing communication channels</td>
<td>12</td>
</tr>
<tr>
<td>Identifying who to talk to</td>
<td>13</td>
</tr>
</tbody>
</table>
Best practices
Receive feedback in a timely manner

“Usually I try to get the person who I'm going to have review the thing to actually sit down and talk with them before I put out the code review” (Participant 7)
Receiving Feedback in a timely manner & Managing time constraints

• Author:
  • Aim for small, incremental changes
  • Allow reviewers to volunteer to perform a review
  • Notify potential reviewers in advance
  • Know when to skip a review
  • Run tests and analysis before

• Reviewer:
  • Set dedicated but bounded time aside for the review
  • Review frequently, fewer changes but more often
  • Provide feedback as soon as possible
Better understanding. Better feedback.

“Typically [a good code review] has a good description of what the problem was, what the solution is, and if it's a big change, it has [documentation explaining] what it's doing and how it's integrated with everything else.”  
(Participant 4)

“A bad reviewer tries to force their preference on you. A good code reviewer makes your code conform to certain principles, but not opinion"  
(Participant 4)
Better understanding. Better feedback.

• **Author:**
  - Aim for small, incremental changes
  - Cluster related changes and submit changes with context
  - Document the motivation
  - Select reviewers with the right expertise
  - Show gratitude & carefully consider feedback

• **Reviewer:**
  - Use richer communication channels, e.g. face-to-face meetings
  - Use tools that provide traceability for non-contentious or sensitive issues
  - Give constructive and respectful feedback
  - Justify and explain the reasons for rejecting a change
  - Focus on core issues first
As an organization, establish a review policy that promotes:

• Build a positive review culture that sets the tone for feedback
• Ensure time spent reviewing is “counted" and “expected"
• Watch for negative impacts of employee assessment or incentives
• Ensure appropriate tools are used
• Make sure tools integrate well
• Ensure sufficient training (e.g. junior working alongside senior)
• Develop, reflect on, and revise code reviewing policies and checklists.
• Resolve bottlenecks
• Identifying inappropriate or aggressive communication
Tool improvements

- Enforce a reviewing workflow
- Support finding right experts
- Show test coverage of code under review
- Automatic clustering of changes into groups of related entities
- Automate feedback
- Traceability support
- Dashboards for metrics and pending reviews
- Management for notifications
- Integration with other communication tools
Tool integration

• Trigger static analysis / inspect analysis output
• Trigger testing / inspect test output
• Overlay code with test coverage
• Trigger builds / inspect build output
• Automatic reviewer selection
• Check-in code
Code review video (3:40)
<table>
<thead>
<tr>
<th>Challenge during Code Reviewing</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving feedback in a timely manner</td>
<td>1</td>
</tr>
<tr>
<td>Review size</td>
<td>2</td>
</tr>
<tr>
<td>Managing time constraints</td>
<td>3</td>
</tr>
<tr>
<td>Understanding the code's purpose</td>
<td>4</td>
</tr>
</tbody>
</table>

Michaela Greiler
@mgreiler
www.michaelagreiler.com

http://research.microsoft.com/tse