The Secret Life of Bugs

Going Past the Errors and Omissions in Software Repositories

Jorge Aranda (University of Toronto)

jaranda@cs.toronto.edu

Gina Venolia (Microsoft Research)

ginav@microsoft.com

Two questions

- As researchers, can we trust software repositories?
  - …for mining purposes?
  - …as good-enough records of the history of a project?
- How do the stories of bugs of large software projects really look like?
  - How do people coordinate to solve them?
  - Which documents and artifacts do they use to diagnose and fix them?
  - How do issues of accountability, ownership, and structure play out?

Methodology

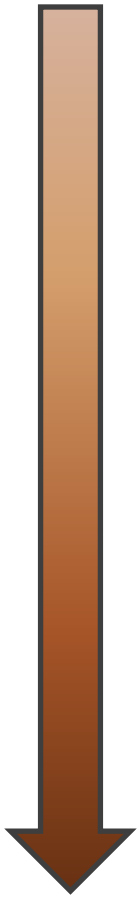
- We performed a field study of communication and coordination around bug fixing
  - Multiple-case case study
  - Survey of software professionals

Methodology – Case study (1)

- Ten in-depth cases
- Full investigation of the history of a bug
  - Randomly select a recently closed bug record
  - Obtain as much information as possible from electronic records
  - Tracing backwards, contact the people that updated or were referenced by the records
  - Interview them to correct and fill the holes in our most current story
    - Interventions of other people
    - Documents or artifacts that were not referenced
    - Misunderstandings and documentation errors
    - Tacit or delicate information
  - Repeat for each new person, document, or artifact in our story
    - (Need to use judgment to know when to stop)

Methodology – Case study (2)

- Constructing partial stories



Automated analysis of bug record data

Automated analysis of electronic conversations and other repositories

Human sense-making

Direct accounts of the history by its participants

Methodology – Survey

- Designed after most cases were completed or near completion
  - Its purpose: to confirm or refute the findings from the case study
  - It consisted of 54 questions
  - 1,500 Microsoft employees (developers, testers, program managers) were invited to participate; 110 replied (7.3% response rate)
- Questions focused on the *last* closed bug that the respondent worked on

Cases

Case	Type	How Found	Resolution	Direct Agents	Indirect Agents	Other Listeners	Lifespan (days)	Days w/Events	Events
C1	Documentation	Ad-hoc test	Fixed	6	4	179	320	12	19
C2	Code (security)	Ad-hoc test	Fixed	21	6	3	408	49	138
C3	Build test failure	Automated	Fixed	42	8	291	59	21	141
C4	Code (functionality)	Ad-hoc test	Fixed	6	1	0	7	5	16
C5	Code (install)	User (Beta)	By Design	2	3	0	2	2	12
C6	Code (functionality)	Automated	Fixed	2	4	11	29	6	20
C7	Build test failure	Automated	Fixed	6	7	197	14	6	34
C8	Code (functionality)	Dog food	Won't Fix	2	7	0	2	2	5
C9	Code (functionality)	Automated	Not Repro	5	2	1	2	2	12
C10	Code (functionality)	Escalation	Fixed	23	18	13	35	20	220

Errors and omissions (1)

- ALL of our cases' electronic repositories omitted important information
- MOST of them included erroneous information
- ALL of our cases were strongly dependent on social, organizational, and technical knowledge that cannot be solely extracted through automation

Errors and omissions (2)

People

Case	Level 1	Level 2	Level 3	Level 4
C1	7	9	9	10
C2	5	5	27	27
C3	12	38	38	50
C4	5	5	7	7
C5	4	5	2	5
C6	7	7	5	6
C7	7	14	12	13
C8	6	6	15	9
C9	6	7	7	7
C10	8	25	41	41

Events

Case	Level 1	Level 2	Level 3	Level 4
C1	8	16	17	19
C2	11	11	138	138
C3	19	119	119	141
C4	11	14	15	16
C5	8	11	11	12
C6	12	18	19	20
C7	6	33	34	34
C8	4	4	5	5
C9	7	11	12	12
C10	17	78	149	220

Level 1: Automated analysis of bug record
Level 2: Automated analysis of electronic conversations and repositories

Level 3: Human sense-making
Level 4: Direct accounts of the history by its participants

Errors and omissions (3)

- Erroneous data in bug record
  - The most basic data fields were sometimes incorrect
    - A “Code bug” that should have been a “Test bug”
    - A duplicate still marked only as “Resolved” when the issue and all other duplicates were already “Closed”
    - A “Won’t Fix” that should have been a “By Design”
  - Survey
    - 10% had an inaccurate resolution field

Errors and omissions (4)

- Missing data in bug record
  - Important bits of data often missing from the record
    - Links to corresponding source code change-sets
    - Links to duplicates
    - Links to bugs found in the process of resolving the original
    - Reproduction steps
    - Corrective actions and root causes
  - Survey:
    - 70% bugs required a source code commit; 23% of them had no link from the record to the change-set
    - Reproduction steps incomplete, inaccurate, or missing: 18%
    - Root cause incomplete, inaccurate, or missing: 26%
    - Corrective actions incomplete, inaccurate, or missing: 35%

Errors and omissions (5)

- People
  - A problem in almost every case
    - Key people not mentioned in bug record or in emails
    - Bug “owners” that had in fact nothing to do with the bug
    - Little relation between how much a person speaks and how much that person actually contributes
    - Geographic location wrong (at least) twice
  - Survey:
    - Bug owners drive the resolution of their bugs only 34% of the time (and they have nothing to do with them in 11%)
    - For 10% of the bugs, the primary people are hard to spot from the record; for an additional 10% they do not even appear on the record
    - All of the people in the bug’s history and fields are fully irrelevant in 7% of the cases

Errors and omissions (6)

- Events
  - It is unrealistic to expect that all events will be logged electronically; nevertheless…
    - Key events (troubleshooting sessions, high-level meetings) often left no trace
    - Most face-to-face communication events also left no trace
    - Many of the events actually logged are junk or noise
    - Some events were logged in an erroneous chronological sequence
- Groups and politics
  - Team- and division-level issues; “soft” information
    - Pockets of people with different culture and dynamics
    - Changes in dynamics depending on proximity to milestones
    - Struggles over bug ownership

Errors and omissions (7)

- Rationale
  - *Why* questions were usually the hardest to answer
    - Why did A choose B as a required reviewer, but C as an optional one?
    - Why was there no activity in this bug for two weeks after bouts of minute-by-minute updates?
    - Why are the Status or the Resolution fields wrong?

Example (1)

- Day 3, 10:00 AM – Opened by Gus (includes reproduction steps and screenshot)
- Day 3, 10:00 AM – Edited by Gus
- Day 4, 12:15 PM – Assigned by Brian to David
- Day 4, 12:15 PM – Edited by David
- Day 7, 9:30 AM – Edited by David
- Day 7, 9:30 AM – Edited by David
- Day 7, 12:00 PM – Edited by David (explanation, code review approved)
- Day 7, 12:00 PM – Resolved as Fixed by David
- Day 7, 2:00 PM – Closed by Igor

Example (2)

- On Day 1, Igor, a developer, notices an odd behavior in a feature of a colleague, David
- He creates a bug record. He doesn't include reproduction steps or any details in the record, but he discusses the issue with David face-to-face
- That evening, Claudia (a PM) assigns the bug to David
- On Day 3, after another face-to-face chat with Igor, David reports that he "understood the problem"
- In parallel, though, a tester named Gus stumbles upon the same problem through ad-hoc testing. He logs the bug, provides detailed reproduction steps, and a screenshot of the error
- On Day 4, Brian, another PM, assigns this second bug to David as well. A minute later, David marks the *first* bug as Resolved (Duplicate)

Example (3)

- After the weekend, on Day 7, David submits a fix, and requests a code review (as is required in his team) from two other developers, Pradesh and Alice. Pradesh approves the code in less than two hours.

- The fix consists of hundreds of lines of code spread across several files. How did David code it so quickly? He used code to address the same issue from his old company, which is now owned by Microsoft. Pradesh, being familiar with the problem and the old code (he, too, comes from the same old company), simply reviews the "stitches" and approves the change

- David marks the bug as Resolved (Fixed)

- An hour later, Igor contacts David to ask him what the *old* bug was a duplicate of. David gives him the reference to the new bug. It seems Igor has both bugs open in his screen, and mistakenly closes the *new* bug instead of his own, which remained open until we pointed it out during our questioning, on Day 9.

Errors and omissions (recap)

- ALL of our cases' electronic repositories omitted important information
- MOST of them included erroneous information
- ALL of our cases were strongly dependent on social, organizational, and technical knowledge that cannot be solely extracted through automation

Two questions (recap)

- As researchers, can we trust software repositories?
  - …for mining purposes?
    - Perhaps, depending on your research questions and constructs
    - You'll need extreme caution if you do
  - …as good-enough records of the history of a project?
    - No

- How do the stories of bugs of large software projects really look like?
  - How do people coordinate to solve them?
  - Which documents and artifacts do they use to diagnose and fix them?
  - How do issues of accountability, ownership, and structure play out?

Coordination dynamics

- No uniform process or lifecycle
  - Very rich stories for even the simplest cases
- We opted to describe the pieces rather than the whole
  - We created a list of coordination patterns
    - And used the survey to validate their existence and relevance

Coordination patterns (1)

Communication media	
Broadcasting emails	Sending a manual or automatic notification to a number of mailing lists to inform their members of an event.
Shotgun emails	Sending an email to a number of mailing lists and individuals in the hope that one of the recipients will have an answer to the current problem.
Snowballing threads	Adding people to an ever-increasing list of email recipients.
Probing for expertise	Sending emails to one or few people, not through the “shotgun” method, in the hope that they will either have the expertise to assist with a problem or can redirect to somebody that will.
Probing for ownership	Sending emails to one or few people, not through the “shotgun” method, requesting that they accept ownership of the bug or can redirect to somebody that will.
Infrequent, direct email	Emails sent privately and infrequently among a handful of people.
Rapid-fire email	Bursts of email activity in private among a few people in the process of troubleshooting the issue.
Rapid-fire email in public	Like the above, but with tens or hundreds of people copied as recipients of the email thread, most of them unconnected to the issue.
IM discussion	Using an instant messaging platform to pass along information, troubleshoot, or ping people.
Phone	Phone conversations used to pass along information, troubleshoot, or ping people.

Coordination patterns (2)

Bug database	
Close-reopen	A bug that is reopened because it had been incorrectly diagnosed or resolved, or because there is a disagreement on its resolution or on the team's ability to postpone addressing it.
Follow-up bugs filed	Other bugs were found and filed in the process of fixing this one, or a piece of this bug was filed in a different record as follow-up.
Forgotten	A bug record that goes unnoticed and unattended for long periods.

Meeting	
Drop by your office	Getting a piece of information, or bouncing some ideas regarding the issue, face to face informally with a co-worker in a nearby office.
Air time in status meeting	The issue was discussed in a regular group status meeting.
Huddle	The issue called for a team meeting exclusively to discuss it.
Summit	The issue called for a meeting among people from different divisions exclusively to discuss it.
Meeting w/ remote participants	Any meeting where at least one member is attending remotely.
Video conferences	Any meeting where video was used to communicate with at least one attendee.

Coordination patterns (3)

Working on code	
Code review	The fix for this bug was reviewed and approved by at least one peer.
Two birds with one stone	The fix for this bug also fixed other bugs that had been discovered and filed previously.
While we're there	The fix for this bug also fixed other bugs that had not been discovered previously.

Other patterns	
Ignored fix/diagnosis	A correct diagnosis or fix that was proposed early on and was temporarily ignored by the majority.
Ownership avoidance	Bouncing ownership of the bug or code.
Triaging	Discussing and deciding whether this is an issue worth addressing.
Referring to the spec	At least one concrete and specific reference to a spec, design document, scenario, or vision statement, to provide guidance to solve or settle the issue.
Unexpected contribution	New information or alternatives that come from people out of the group discussing the issue.
Deep collaboration	Two or more people working closely (face to face or electronically) and for a sustained period to unravel the issue.

Coordination patterns (4)

	Essential (E)	Occurred (O)	E / O
Probing for ownership	16%	30%	52%
Summit	1%	2%	50%
Probing for expertise	17%	34%	49%
Code review	30%	66%	46%
Triaging	32%	69%	46%
Rapid-fire email	13%	33%	38%
Infrequent, direct email	18%	49%	37%
Shotgun emails	7%	19%	37%
Drop by your office	23%	64%	36%
IM discussion	11%	32%	35%
Phone	7%	20%	33%
Meetings w/remote participants	4%	12%	33%
Huddle	2%	6%	33%
Air time in status meeting	8%	27%	29%
Close-reopen	4%	13%	29%
Broadcasting emails	10%	35%	27%
Referring to the spec	6%	24%	24%
While we're there	3%	14%	21%
Rapid-fire email (in public)	1%	5%	20%
Follow-up bugs filed	3%	18%	16%
Deep collaboration	3%	22%	13%
Snowballing threads	2%	16%	13%
Two birds with one stone	1%	17%	6%
Ownership avoidance	1%	28%	3%
Ignored fix/diagnosis	0%	9%	0%
Video conferences	0%	3%	0%
Unexpected contribution	0%	9%	0%

Two questions (recap)

- As researchers, can we trust software repositories?
  - …for mining purposes?
    - Perhaps, depending on your research questions and constructs
    - You'll need extreme caution if you do
  - …as good-enough records of the history of a project?
    - No
- How do the stories of bugs of large software projects really look like?
  - They are rich and varied
  - Some of their major elements may be described using patterns

But that's just the case for Microsoft, right?

- No

- Microsoft employees seem to be as careful as those of other large companies (or more) in keeping and using their electronic records appropriately
  - But important information is often tacit
  - Personal, social, and political factors are part of all organizations
  - Note that many of these errors don't matter for the organization itself
    - The goal of an electronic repository is to help develop a product, *not* to serve as an accurate record for historians
    - For them it's often more efficient to simply move on

What about open source projects?

- We don’t know
  - Records may match reality better (less face-to-face communication, better logging)
  - But tacit information will likely remain tacit
  - Our lists of goals and coordination patterns should remain valid

Thanks

- …to the Human Interactions of Programming (HIP) Group at Microsoft Research
- …to Steve Easterbrook, Greg Wilson, and Jeremy Handcock for thoughts and comments
- …to our study participants

- *Credits for photographs: John Cancalosi (fossil), André Karwath (yellow-winged darted dragonfly)*

Questions?

Jorge Aranda (jaranda@cs.toronto.edu) and Gina Venolia (ginav@microsoft.com)

- As researchers, can we trust software repositories?
  - …for mining purposes?
    - Perhaps, depending on your research questions and constructs
    - You'll need extreme caution if you do
  - …as good-enough records of the history of a project?
    - No

- How do the stories of bugs of large software projects really look like?
  - They are rich and varied
  - Some of their major elements may be described using patterns