The Secret Life of Bugs

Going Past the Errors and Omissions in Software Repositories

- Jorge Aranda, University of Toronto
- Gina Venolia, Microsoft Research

Presented By: Shivjot Baidwan

Outline

- Introduction
- Main Focus
- Related Work
- Study, Design and Execution
  - Multiple-case Case Study
  - Survey
- Errors and Omissions
  - Levels of Data Collection and Analysis
  - Levels in actual practice
- Coordination Patterns
- Results
- Discussion

Introduction

- A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result.
- Bugs are typically found in almost all the software thus making Software Testing one of the most critical and costly phases in Software Development.
- With the increasing size and complexity of software, the number of bugs found in a software also increase.
- The modern large-scale software development demands managing a huge quantity of bugs on a day-to-day basis thus fixing them is one of the most common and time consuming task for developers.
- It is highly impractical to remember details about bugs.

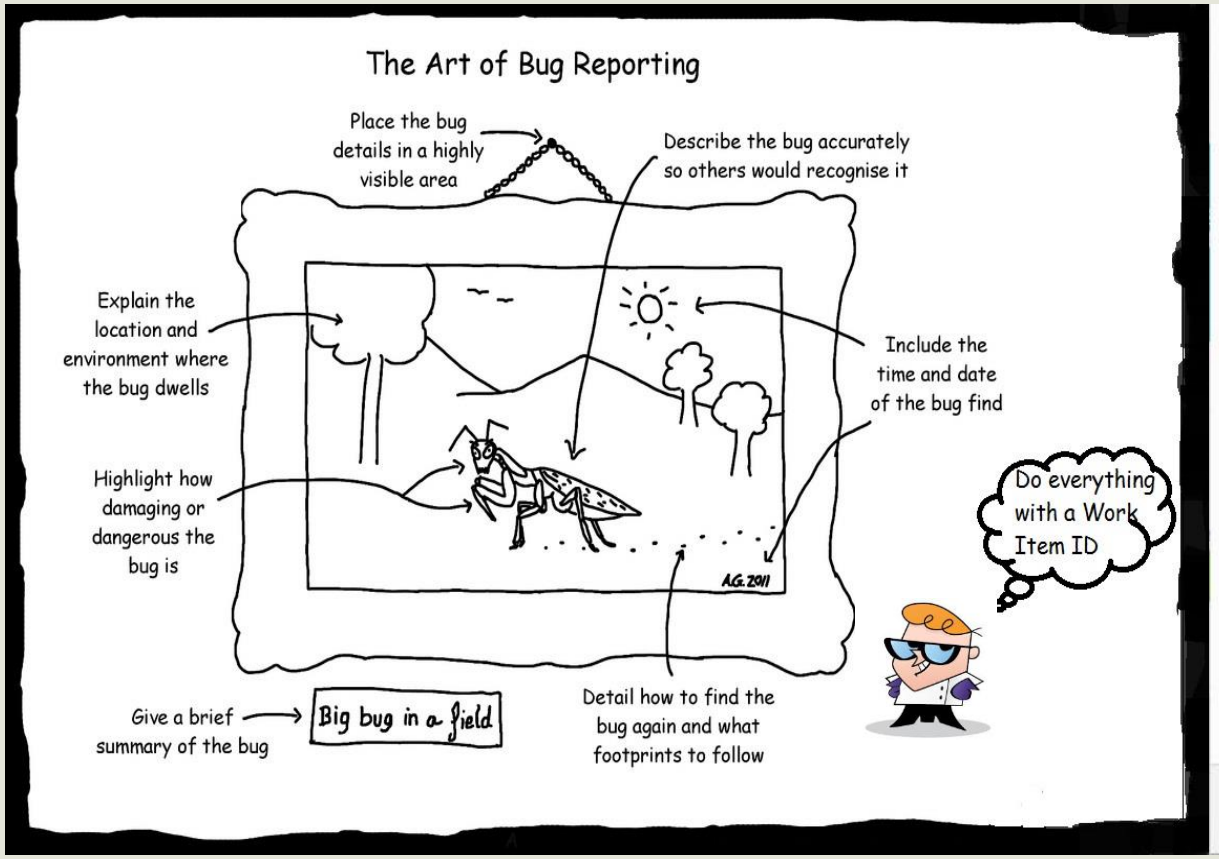
Main Focus

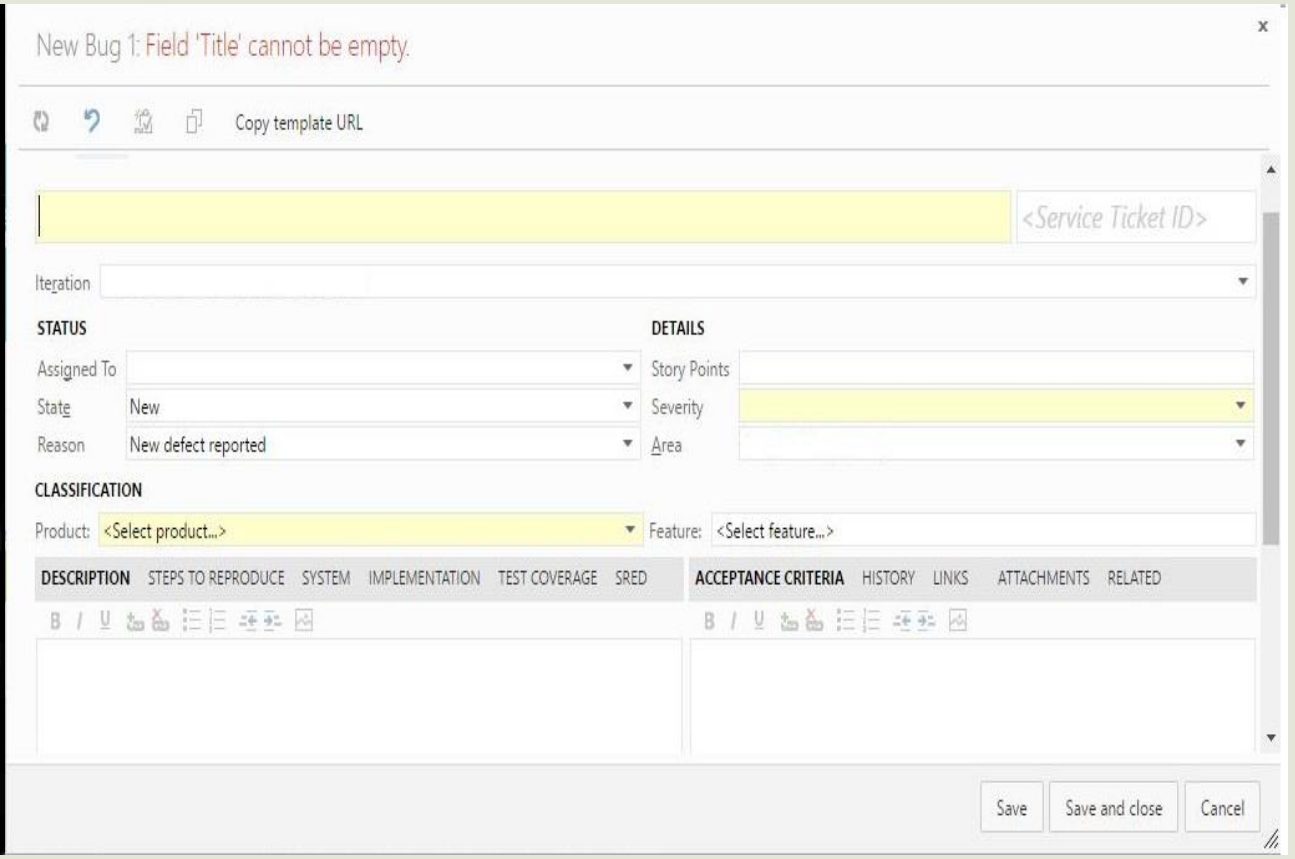
- Every BUG has a STORY behind it!!
- Aims to contribute towards a Contextually rich work- item- centric coordination in Bug Fixing tasks.
- Identifying common bug fixing coordination patterns by utilizing Bug Histories and survey results.
- Synopsis of common coordination patterns found in software companies.
- Strong dependence on social, organizational and technical knowledge.
- A general framework to design better tools and practices.

Related Work (3 Main Approaches)

- Reports are fundamentally extracted from electronic traces of team activity
  - Cataldo et al – Compute Coordination Requirements.
  - Valetto et al – Socio-technical compatibility in an organization.
- Detailed and rich accounts of activities of few software professionals.
  - Chong and Hurlbutt – Coordination of programming pairs.
  - Ko et al – Informational needs of Collocated Teams.
- Trends and Patterns of Right and Wrong in Software Development.
  - Souza and Redmiles – Characterize coordination of Software Professionals.

Bug Reporting





Study, Design and Execution

- Research Questions:
  - How is the process of fixing bugs coordinated in software teams?
  - What is the life cycle of bugs?
  - Common coordination patterns among team members?
  - Does resolution of bugs and coordination patterns affect socio-technical aspect?
  - Do electronic traces of interaction provide a good picture of coordination?
  - Is non- persistent knowledge necessary to understand the story of each bug fix?
- Research conducted in two phases:
  - Multiple-Case Case Study.
  - Validation of Case Study.

Multiple-Case Case Study

- Unit Analysis – History of a Closed Bug.
- Collection of conceptually related activities that were summarized at least once in the bug database during their life cycle.
- Assumptions:
  - Duplicates considered as component of the same conceptual entity.
  - Bugs do not begin their life when an entry is created or when entry is marked as closed.
  - Some bugs exhibit different symptoms at different times and therefore may have been recorded under different categories should be considered as a single bug.

Multiple-Case Case Study

- Selection Criteria:
  - Bug filed in Database with fundamental information.
  - Bug had Closed Resolution before experiment begun.
  - Bug was fresh with a Closed Resolution.
- Aim of the Selection – To reach the beginning of the story for a particular bug i.e. where the bug had been originally discovered.

Multiple-Case Case Study

- Process:
  - Gathering all the information related to the bug from the audit trail, data fields in bug tracking system, entities involved as well as links to source code repositories.
  - Tracing the Audit trail in the backward direction. It involved approaching the entities relevant to the bugs.
  - Interviewing the participants and obtaining as much information from them as feasible.
- Scenarios:
  - Process reached a dead end.
  - Inquiries lead to entities which were trivial for the study.

Multiple-Case Case Study

- Data Collection Results
  - A list of primary and secondary actors involved.
  - A list of relevant artifacts and tools.
  - A chronological list of information flow.
  - A chronological list of coordination events in a Bug’s history.

Cases Summary

Case	Type of Case	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

Survey

- A survey of 54 questions was constructed for software professionals at Microsoft.
- Questionnaire was sent to 1500 Microsoft employees at random.
- The selection of employees was divided between developers, testers and program managers.
- All the participants were given a chance to win a 500USD gift card.
- A 7.3% (110 responses) of response rate was achieved.
- A response to all the questions was optional but 100 participants answered all the questions.

Survey

- Survey asked each respondent about the history of a bug they recently closed.
  - Studying the record in the bug database.
  - Reading the email communications.
- Survey consisted of three main parts:
  - General information about the bug.
  - Questions pertaining to coordination patterns.
  - Questions related to bug database and whether the database painted the complete portrait of the life of the bug.

Survey Responses

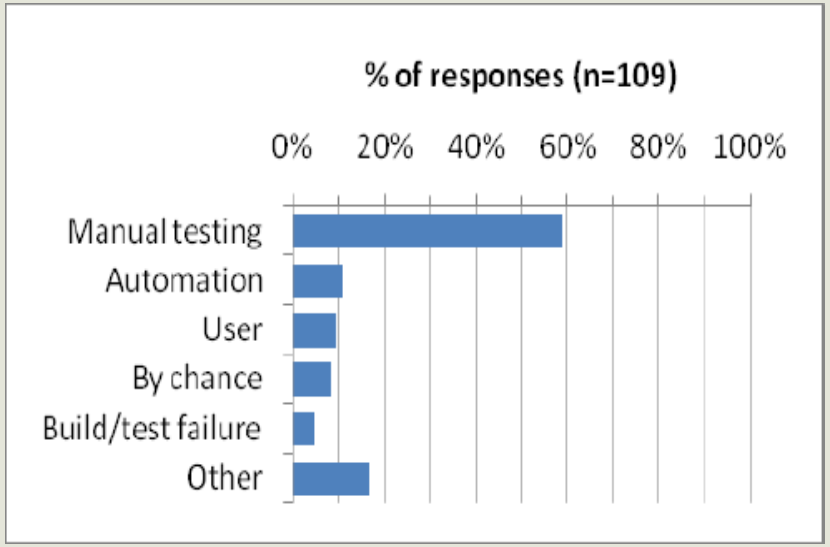


Fig1: How was the bug found?

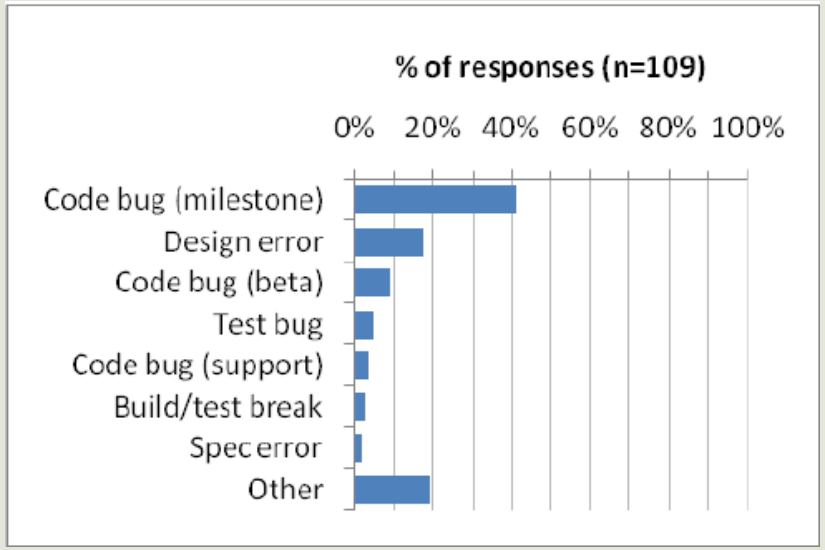


Fig 2: Type of Bug

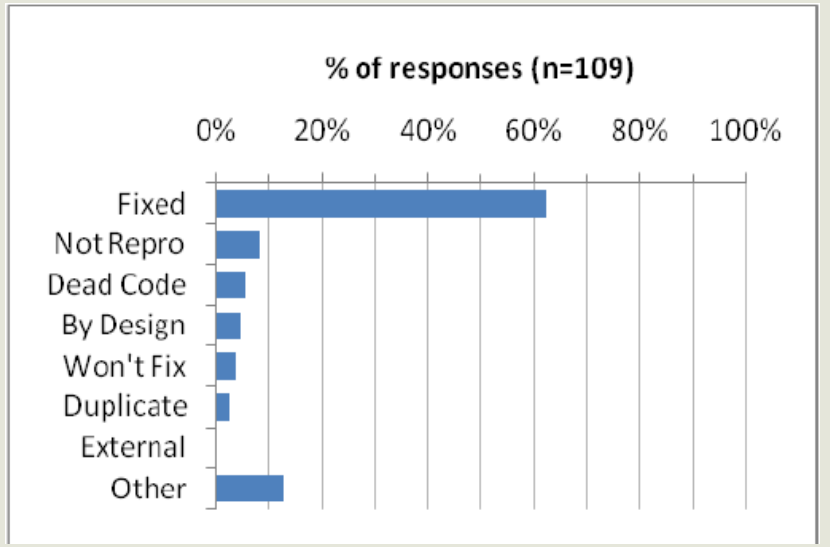


Fig 3: How was the bug closed?

Errors and Omissions (Levels of Data Collection and Analysis) L1

Level 1: Automated Analysis of Bug Record Data – Automation can be utilized to obtain information about:

- List of entities involved.
- Bug's Life Span.
- It's resolution and previous states.
- Owners.
- Code change-sets.
- Chronological List of events.

Errors and Omissions (Levels of Data Collection and Analysis) L2

- Level 2: Automated Analysis of Electronic Conversations and Other Repositories
  - Hints related to electronic communications can be utilized to construct a social network
  - The data from this social network can be filtered using timestamps, participants, keywords in order to locate more relevant material related to the bug in question.
  - Assumption: The automated social network is similar in structure and intensity to the actual communication among the entity members.

Errors and Omissions (Levels of Data Collection and Analysis) L3 and L4

- Level 3: Human Sense Making:
  - Automation still far away from human's capability to comprehend connections in data.
  - For Automation to work, we need to improve the current coordination dynamics and it should include information about social and political interactions among team members.
- Level 4: Direct Account of the History by its participants.
  - Interviewing is the ideal technique to gather non-documented information/events which is essential to understand the history of the bug.
  - Successful if collected before the bug is forgotten.
  - Helps in validating earlier conclusions.

Errors and Omissions (The Levels in Practice)

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

Fig: Events

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

Fig: Agents

Errors and Omissions (The Levels in practice)

- Erroneous data fields:
  - Basic data fields were sometimes incorrect
    - E.g.: Test Bug marked as Code Bug.
  - Duplicate bugs with different fields
    - E.g.: A bug marked as Closed and its duplicate marked as Resolved.
- Missing Data fields:
  - Missing links to source code change sets.
  - Missing details about a bug found while fixing another.

Errors and Omissions (The Levels in practice)

- People:
  - Obtaining lists of primary and secondary participants in a bug's history was constant source of error.
  - Geographic location of employees was incorrect therefore interviewing the participants was difficult.
- Events:
  - Face- to- face events had no record in the repositories.
  - Chronological order of events was difficult to comprehend.

Coordination Patterns

- Problem :
  - Although the bug histories were rich, varied and context dependent, they did not follow a uniform path.
  - To describe the lifecycle of bugs.
  - To describe the process of fixing the bugs.
- Solution:
  - Did not formulate a process for all bug histories.
  - Selected recurrent patterns based on observation.
  - Determined the patterns with rare occurrence but had a bigger impact in the bug history.

Coordination Patterns based on Communication Media

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 based on Bug Database and Working on Code

Bug database	
Close-reopen	A bug that is reopened because it had been incorrectly diagnosed or resolved, or because there is 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.
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.

Coordination Patterns based on Meetings and Other Patterns

Meeting	
Drop by your office	Getting a piece of information, or bouncing some ideas regarding the issue, face to face informally with a coworker 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 with remote participants	Any meeting where at least one member is attending remotely (could be a huddle or summit meeting).
Video conferences	Any meeting where video was used to communicate with at least one attendee (could be a huddle or summit meeting.
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 (Issues)

- Inefficient but used often:
  - Snowballing
  - Rapid fire email in public
- Efficient but not used often:
  - Summit
  - Forgotten

Coordination Patterns (Survey)

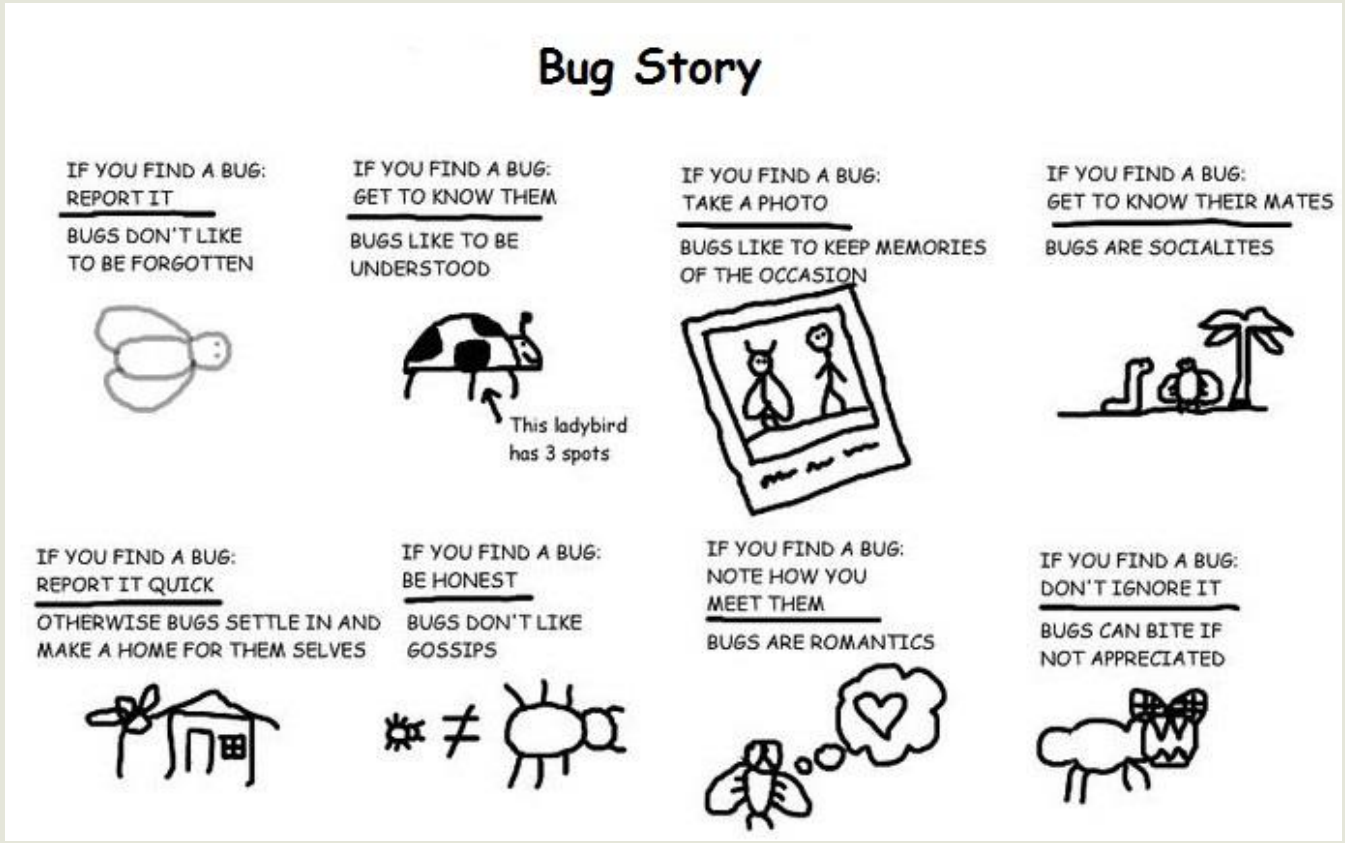
	Essential (E)	Occurred (O)	(E / O)
Probing for ownership	16%	30%	52%
Summit	1%	2%	50%
Probing for expertise	17%	34%	49%
Code review	30%	66%	46%
Triaging	32%	69%	46%
Rapid-fire email	13%	33%	38%
Infrequent, direct email	18%	49%	37%
Shotgun emails	7%	19%	37%
Drop by your office	23%	64%	36%
IM discussion	11%	32%	35%
Phone	7%	20%	33%
Meetings w/remote part.	4%	12%	33%
Huddle	2%	6%	33%
Air time in status meeting	8%	27%	29%

	Essential (E)	Occurred (O)	(E / O)
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 (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%

Implications

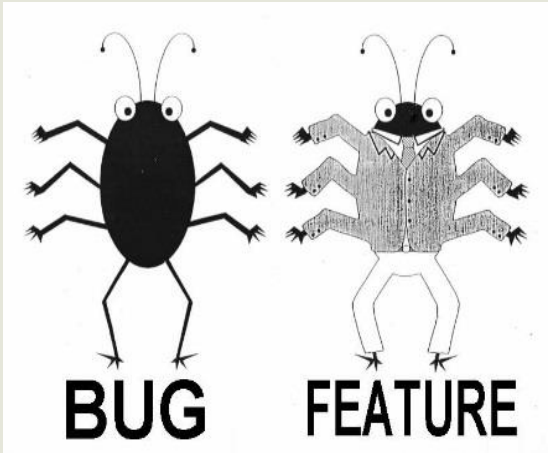
- For Tool Designers:
  - The states Active, Assigned, Resolved are useful to help manage the software development, but they are the poor approximations of the true lifecycle of the bug.
  - Assignment of Ownership is often problematic- who is responsible for this buggy code?
- For Researchers:
  - Provide elementary intelligence to automation tools to detect and ignore noise activity.
  - Studying standard operations of the participating organizations can help researchers to determine whether most coordination events leave minable electronic traces or not.

Every Bug has a Story !!



Bug / Feature


What is happening with that program now...
Is that a bug?
That's not a bug...
I call that a feature!


BUG
FEATURE

Conclusion

- Even simple bugs are strongly dependent on social, organizational and technical knowledge which cannot be retrieved through automated analysis of software repositories.
- Coordination patterns validated with the help of survey can be utilized as a framework to design better tools.

Discussion

- Do you really think that “Every Bug has a Story”?
- What were the limitations that you feel the paper had?
- Can you think of any other Coordination patterns?
- What is the lifecycle of the bug?