Information Needs in Programming

Thomas LaToza
A few minutes in the life of a developer
What percentage of the last week have you spent...

## A few hours in the life of a professional software developer

<table>
<thead>
<tr>
<th>collaboration</th>
<th>Developer assigned bug by team</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming</td>
<td>Reproduces error</td>
</tr>
<tr>
<td></td>
<td>Browser hits error message (500 internal error)</td>
</tr>
<tr>
<td></td>
<td>Attaches debugger</td>
</tr>
<tr>
<td></td>
<td>Browse to page again, hit null reference exception</td>
</tr>
<tr>
<td></td>
<td>Hypothesize from call stack which function might be responsible</td>
</tr>
<tr>
<td></td>
<td>Browse through code</td>
</tr>
<tr>
<td></td>
<td>Uses debugger to change values &amp; experiment</td>
</tr>
<tr>
<td></td>
<td>Make change, recompile, check, doesn’t work</td>
</tr>
<tr>
<td></td>
<td>Navigates slice, wrong values came from objects</td>
</tr>
<tr>
<td></td>
<td>In complicated code doesn’t understand</td>
</tr>
<tr>
<td>collaboration</td>
<td>Walks to B’s office and asks where data comes from</td>
</tr>
<tr>
<td></td>
<td>B working on high profile feature in area</td>
</tr>
<tr>
<td>programming</td>
<td>Tries to make change, still doesn’t work</td>
</tr>
<tr>
<td>collaboration</td>
<td>Walks back to B, realize related to C’s feature, C at lunch</td>
</tr>
<tr>
<td></td>
<td>After lunch, A and B walk to C’s office,</td>
</tr>
<tr>
<td>design</td>
<td>A, B, C change design to work with new feature</td>
</tr>
<tr>
<td>collaboration</td>
<td>Bug passed from A to C to change feature</td>
</tr>
</tbody>
</table>

Developers use a variety of techniques for obtaining information about code

Productivity in programming

- expertise
- development environments
- programming languages
- code quality
- team practices

Effort to obtain information

- time to market
- software quality
Overview: Towards a theory of information needs in programming

• What is **information** about code, and how do developers obtain it?

• **What** information about code do developers need?

• How does **expertise** help?

• How can we **design** more productive human-code interactions?
Understanding how humans interact with code

observations

Screencasts

Transcripts

surveys

38. Of the time I spent understanding existing code last week, the percent of time I spent

Examining source code
- Examining source code check-in comments and diffs
- Examining high-level views of source code (UML diagrams, class hierarchies, call graphs, …)
- Running the code and looking at the results
- Running the code and examining it with a debugger
- Using debug or trace statements

0% 1% 2% 3% 4% 5% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

39. Other techniques used last week (if you answered “other” above)
(Max Characters: 256)

40. This technique was effective for understanding existing code
What is information in programming, and how do developers obtain it?
Developers have hierarchic goals, including answering questions

<table>
<thead>
<tr>
<th>task</th>
<th>Investigate and fix a design problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>question</td>
<td>Why is an event being issued by forcing a cache update?</td>
</tr>
<tr>
<td></td>
<td>How is BufferHandler using its buffer field? Are there any other mutations on it?</td>
</tr>
<tr>
<td></td>
<td>Read methods of BufferHandler</td>
</tr>
<tr>
<td>action</td>
<td>Why is there a buffer member variable that is never used?</td>
</tr>
<tr>
<td></td>
<td>Investigate references to BufferHandler:buffer</td>
</tr>
<tr>
<td></td>
<td>Why is doDelayedUpdate() a member of BufferHandler?</td>
</tr>
<tr>
<td></td>
<td>Reads methods along path, concludes that BufferHandler tracks update delays</td>
</tr>
<tr>
<td></td>
<td>Why wouldn’t isFoldStart() call getFoldLevel()</td>
</tr>
<tr>
<td></td>
<td>Reads isFoldStart(), getFoldAtLine()</td>
</tr>
<tr>
<td></td>
<td>Concludes isFoldStart() doesn’t call because of short circuit evaluation</td>
</tr>
<tr>
<td>Implement fix</td>
<td></td>
</tr>
<tr>
<td>Assure correctness</td>
<td></td>
</tr>
<tr>
<td>Set conditional break point</td>
<td></td>
</tr>
<tr>
<td>Check that jEdit still appears to work correctly</td>
<td></td>
</tr>
<tr>
<td>Repro original bug by reinserting</td>
<td></td>
</tr>
</tbody>
</table>

LaToza and Myers. Designing useful tools for developers. PLATEAU 2011.
Developers seek and learn facts about code

**JEditBuffer**

```
getFoldLevel
```

getFoldLevel has effects

READ getFoldLevel LEARN *getFoldLevel has effects*

“Yes, I am indeed surprised that get fold level has side effects. That is surprising to me.”
Developers seek to explain facts to discover hidden constraints which might be violated by a change

```
```

LEARN 2 - folds updated by buffer changes are updated on screen
2 EXPLAINS buffer mutating operations result in doDelayedUpdate call

“When you're inserting text you could actually doing something that makes the folds status wrong. … In the quick brown fox. If fox is under brown and I'm right at fox and I hit backspace. Then I would need to update my fold display to reflect the new reality which is that it's in a different place. It's now a child of quick, not a child of brown.”
Developers critique poor facts, proposing changes to make

CRITIQUE doDelayedUpdate triggers update
“And the second thing that I don't like is that it is firing these updates. ... It shouldn't be relying on one of these guys to be calling this update routine manually.”
Developers propose changes to accomplish changes, subject to respecting discovered constraints

PROPOSE
“So, in an ideal world, when would this notification going out go out. Cause there's this insert here. Then there's a fireTransactionComplete from the insert. This might be a good place to notify somebody to update this data structure about the fold levels.”
Developers reason about the implications of changes

- **SEEK**: Read relevant methods looking for facts
- **CRITIQUE**: Fact A is bad design
- **LEARN**: Fact A is true
- **EXPLAIN**: Fact A is true to make fact B true
- **PROPOSE**: Change facts A1, B1 to facts A2, B2
- **IMPLEMENT**: Change code to reflect facts A2, B2

LaToza, Garlan, Herbsleb, Myers. Program comprehension as fact finding. FSE 07.
Developers seek changeable facts to accomplish their goals

START updateCaretStatus called 7 times

GOAL updateCaretStatus called fewer times
Developers constantly deal with uncertainty, parsing information into facts that may or may not be true.

START `updateCaretStatus` called 7 times

GOAL `updateCaretStatus` called fewer times
“Cause I’m thinking that when I perform the action of switching from one buffer to another buffer, somewhere it calls a method that indicates that the buffer has been edited. But I didn’t edit the buffer. I’m just switching between buffers. So that has to be removed.”
Developers make path choice decisions

START updateCaretStatus called 7 times

PROPOSE updateCaretStatus called from EditBus events

HYPOTHESIZE EditBus has an event for every updateCaretStatus call

Path choice decision
Does EditBus have the right events?

SEEK updateCaretStatus called from EditBus events

GOAL updateCaretStatus called fewer times

HYPOTHESIZE EditBus has an event for every updateCaretStatus call

SEEK updateCaretStatus called from EditBus events

UPDATECaretStatus called fewer times

IMPLEMENT updateCaretStatus called from EditBus events

OPTIMISTIC ASSUMPTION

ABANDON PATH

PESSIMISTIC ASSUMPTION
Wrong path choice decisions waste time

START updateCaretStatus called 7 times

PROPOSE updateCaretStatus called from EditBus events

HYPOTHESIZE EditBus has an event for every updateCaretStatus call

GOAL updateCaretStatus called fewer times

ABANDON PATH

TEST SEEK subscribe to EditBus

IMPLEMENT updateCaretStatus called from EditBus events
What information about code do developers need?
What activity are you currently working on?

Participants
17 professional developers

Task
picked one of their own coding tasks involving unfamiliar code

Setup
think-aloud

~90 minutes

Transcripts
(386 pages)
annotated with observer notes

Coding Activity

Interesting. This looks like, this looks like the code is approximately the same but it's refactored. But the other code is.

Changed what flags it's ???

He added a new flag that I don't care about. He just renamed a couple things.

Well.

So the change seemed to have changed some of the way these things are registered,

but I didn't see anything that talked at all about whether the app is running or whether the app is booted. So it seems like, this was useless to me.

Domain knowledge – changes to some functionality are not relevant to other functionality
Information needs in programming activities by % of programming time

For tasks in code in your own codebase that you haven’t seen recently

Circle size: % of time  Edge thickness: % of transitions observed

LaToza and Myers. Developers ask reachability questions. ICSE 2010.
<table>
<thead>
<tr>
<th>Code Understanding</th>
<th>% agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding the rationale behind a piece of code</td>
<td>66%</td>
</tr>
<tr>
<td>Understanding code that someone else wrote</td>
<td>56%</td>
</tr>
<tr>
<td>Understanding the history of a piece of code</td>
<td>51%</td>
</tr>
<tr>
<td>Understanding code that I wrote a while ago</td>
<td>17%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task Switching</th>
<th>% agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Having to switch tasks often because of requests from my teammates or manager</td>
<td>62%</td>
</tr>
<tr>
<td>Having to switch tasks because my current task gets blocked</td>
<td>50%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modularity</th>
<th>% agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being aware of changes to code elsewhere that impact my code</td>
<td>61%</td>
</tr>
<tr>
<td>Understanding the impact of changes I make on code elsewhere</td>
<td>55%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Links between Artifacts</th>
<th>% agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finding all the places code has been duplicated</td>
<td>59%</td>
</tr>
<tr>
<td>Understanding who “owns” a piece of code</td>
<td>50%</td>
</tr>
</tbody>
</table>
What hard to answer questions about code have you recently asked?

Rationale (42)
Why was it done this way? (14) [15] [7]
Why wasn’t it done this other way? (15)
Was this intentional, accidental, or a hack? (9) [15]
How did this ever work? (4)

Testing (20)
Is this code correct? (6) [15]
How can I test this code or functionality? (9)
Is this tested? (3)
Is the test or code responsible for this test failure? (1)
Is the documentation wrong, or is the code wrong? (1)

Debugging (26)
How did this runtime state occur? (12) [15]
What runtime state changed when this executed? (2)
Where was this variable last changed? (1)
Is this object different from that object? (1)
Why didn’t this happen? (3)
How do I debug this bug in this environment? (3)
In what circumstances does this bug occur? (3) [15]
Which team’s component caused this bug? (1)

Intent and Implementation (32)
What is the intent of this code? (12) [15]
What does this do (6) in this case? (10) [16] [24]
How does it implement this behavior? (4) [24]

Refactoring (25)
Is there functionality or code that could be refactored? (4)
Is the existing design a good design? (2)
Is it possible to refactor this? (9)
How can I refactor this (2) without breaking existing users? (7) [9]
Should I refactor this? (1)
Are the benefits of this refactoring worth the time investment? (3)

History (23)
When, how, by whom, and why was this code changed or inserted? (13) [7]
What else changed when this code was changed or inserted? (2)
How has it changed over time? (4) [7]
Has this code always been this way? (2)
What recent changes have been made? (1) [15] [7]
Have changes in another branch been integrated into this branch? (1)

Implications (21)
What are the implications of this change for (5) API clients (5), security (3), concurrency (3), performance (2), platforms (1), tests (1), or obfuscation (1)? [21] [15] [24]

Testing (20)
Is this code correct? (6) [15]
How can I test this code or functionality? (9)
Is this tested? (3)
Is the test or code responsible for this test failure? (1)
Is the documentation wrong, or is the code wrong? (1)

Implementing (19)
How do I implement this (8), given this constraint (2)? (10)
Which function or object should I pick? (2)
What’s the best design for implementing this? (7)

Control flow (19)
In what situations or user scenarios is this called? (3) [15] [24]
What parameter values does each situation pass to this method? (1)
What parameter values could lead to this case? (1)
What are the possible actual methods called by dynamic dispatch here? (6)
How do calls flow across process boundaries? (1)
How many recursive calls happen during this operation? (1)
Is this method or code path called frequently, or is it dead? (4)
What throws this exception? (1)
What is catching this exception? (1)

Contracts (17)
What assumptions about preconditions does this code make? (5)
What assumptions about pre/post(2) conditions can be made?
What exceptions or errors can this method generate? (2)
What are the constraints on or normal values of this variable? (2)
What is the correct order for calling these methods or initializing these objects? (2)
What is responsible for updating this field? (1)

Performance (16)
What is the performance of this code (5) on a large, real dataset (3)? (8)
Which part of this code takes the most time? (4)
Can this method have high stack consumption from recursion? (1)
How big is this in memory? (2)
How many of these objects get created? (1)

Teammates (16)
Who is the owner or expert for this code? (3) [7]
How do I convince my teammates to do this the “right way”? (12)
Did my teammates do this? (1)

Policies (15)
What is the policy for doing this? (10) [24]
Is this the correct policy for doing this? (2) [15]
How is the allocation lifetime of this object maintained? (3)

Type relationships (15)
What are the composition, ownership, or usage relationships of this type? (5) [24]
What is this type’s type hierarchy? (4) [24]
What implements this interface? (4) [24]
Where is this method overridden? (2)

Data flow (14)
What is the original source of this data? (2) [15]
What code directly or indirectly uses this data? (5)
Where is the data referenced by this variable modified? (2)
Where can this global variable be changed? (1)
Where is the data structure used for this purpose? (1) [2] [24]
What parts of this data structure are modified by this code? (1) [24]
What resources is this code using? (1)

Location (13)
Where is this functionality implemented? (5) [24]
Is this functionality already implemented? (5) [15]
Where is this defined? (5)

Building and branching (11)
Should I branch or code against the main branch? (1)
How can I move this code to this branch? (1)
What do I need to include to build this? (3)
What includes are unnecessary? (2)
How do I build this without doing a full build? (1)
Why did the build break? (2) [59]
Which preprocessor definitions were active when this was built? (1)

Architecture (11)
How does this code interact with libraries? (4)
What is the architecture of the code base? (3)
How is this functionality organized into layers? (1)
Is our API understandable and flexible? (3)

Concurrency (9)
What threads reach this code (4) or data structure (2)? (6)
Is this class or method thread-safe? (2)
What members of this class does this lock protect? (1)

Dependencies (5)
What depends on this code or design decision? (4) [7]
What does this code depend on? (1)

Method properties (2)
How big is this code? (1)
How overloaded are the parameters to this function? (1)

Most frequently reported hard-to-answer questions about code: Rationale

• Why was it done this way? (14)

• Why wasn’t it done this other way? (15)
  • focus on specific alternative being considered

• Was this intentional, accidental, or a hack? (9)
  • Is there a hidden reason that this was done as it is?

• How did this ever work? (4)
Design rationale questions encompass a variety of topics

naming, code structure, inheritance relationships, where resources are freed, code duplication, lack of instrumentation, lack of refactoring, reimplementing instead of reusing, algorithm choice, optimizations, where behavior is implemented, parameter validation, visibility, exception policies
Rationale questions

• Developers, wondering if a change might break something, ask rationale questions to discover constraints on existing code.

• Developers have many strategies to answer rationale questions, including finding past commits, looking for related emails / design docs, asking teammates, making exploratory changes.
How does expertise help?
Studies have long found large productivity differences

- Studies have long found 10x differences in productivity (task time) between developers (e.g., [Sackman et al. 1968])
  - Why?
- Industry interest in hiring strong developers that are “10xers”
  - Who are such developers? Can it be taught?
Categorized developers into “novices” and “experts” based on expertise, compared how they worked

<table>
<thead>
<tr>
<th>YRS industry experience</th>
<th>KLOC largest program</th>
<th>YRS Java experience</th>
<th>significant research programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>4</td>
<td>few years</td>
</tr>
<tr>
<td>0</td>
<td>7.5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>1</td>
<td>few years</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>75</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>10</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>136</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>10 “novices”</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 “experts”</td>
<td>2.5</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10.5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Median</td>
<td>2.5</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>
Novice changes addressed the symptom of a design problem

Code smells

1. Ignoring the return value of a getter
   Using getter for its effects
   - 1 novice change
   - 1 novice change
   - 8 novice + 1 expert changes

2. “Architecturally questionable”
   Changing buffer state from another component
   - Added debug statement
   - Removed call to getFoldLevel
   - BufferHandler triggers refresh by a setter

Diagram:

- JEditBuffer
  - undo
  - remove
  - redo
  - ...
  - isFoldStart
  - fireTransactionComplete
  - transactionComplete
  - setFoldLevel
  - doDelayedUpdate
  - foldLevelChanged

- BufferHandler
  - JEditBuffer
    - JEditTextArea
    - TextAreaPainter
    - paintValidLine
    - invalidateLineRange

- Code smells
  - Ignore return value of a getter
  - Use getter for its effects

- Architecture questionable
  - Change buffer state from another component
Most expert changes addressed the cause of a design problem

1. **Code smells**
   - Ignoring the return value of a getter
   - Using getter for its effects
     - 1 novice change
     - Added debug statement
     - 1 novice change
     - Removed call to `getFoldLevel`
     - 8 novice + 1 expert changes
     - `BufferHandler` triggers refresh by a setter

2. **“Architecturally questionable”**
   - Changing buffer state from another component
     - 2 expert changes
     - Moved call from `BufferHandler` to `JEditBuffer`
Excerpts learned facts at a higher level of abstraction

**EXPERTS**

“Well, this is just updating a cache” (1 min)

**NOVICES**

“What it did was it...computes the new line number and fires an event. But I didn’t see it change any state.” (38 mins, 10 mins reading getFoldLevel)

“So what it does, it starts off from this line, it has this firstInvalidFoldLevel, it goes through all these lines, it checks whether this fold information is correct or not, which is this newFoldLevel, this is supposed to be the correct fold level. If that is not the case in the data structure, it needs to change the state of the buffer. It creates this, it does this change, it sets the fold level of that line to the new fold level.” (51 mins, 12 mins reading getFoldLevel)
Experts explained more, helping them better respect constraints & reason about implications

**EXPERT**

“What's going on is that when you're inserting text you could actually be doing something that makes the folds status wrong. … If fox is under brown and I'm right at fox and I hit backspace. Then I would need to update my fold display to reflect the new reality, which is that it's in a different place. “

**3 NOVICES + 1 Expert**

Gave up moving update after
- Explaining why call was there
- Explaining purpose of BufferHandler
- Bug

Rejected task’s critique from a false fact (Expert)
Supports results from psychology that experts see the world differently through schemas

- Memory for **random** chess boards: **same** for experts and novices
- Memory for position from **actual** game: much better for **experts** than novices
- [deGroot 1946; Chase & Simon 1973]
How can we design more productive human-code interactions?
The state of our understanding of human-code interactions

- Many empirical studies, increasing amount of data
  - Repository studies, lab studies, field studies
  - Do results generalize? What do they tell us?
  - How does eye tracking data inform understanding of code interactions?
- Many tools, languages, and techniques for assisting developers
  - Do they help? When? How much?
  - Do method recommenders work? Is Java a better language than C?
The need for scientific theory in software engineering

- Extensive data on movements of celestial bodies collected by Tycho Brahe in the late 1500s
- Helped provide data for Kepler to formulate laws of planetary motion in the early 1600s
- Explain data, help make predictions
Productivity Mechanisms in Programming

- **Information minimization**
  - Processing information about code takes time.
  - Reducing information to process reduces time, increases productivity

- **Feedback**
  - Edits to code have implications.
  - More information about implications increases quality.
  - Faster information reduces rework time, increasing productivity.
Information minimization: Identifying task-relevant information

• What’s the minimal amount of **relevant** information for a programming task?

• “Proto” theories exist in software engineering.

• **Modularity** posits that relevant information is related code that can be placed nearby
  • Information hiding (OO), coupling / cohesion, module systems, aspects, refactoring

• **Traversal tools** posit that relevant code is structurally connected through dependencies
  • Impact analysis, slicers, debuggers

• **Recommenders** posit that past co-changes predict future co-changes across modules.
  • Concern management, method recommenders

• How relevant is information provided —> productivity benefits of approach
Relevance model: Design by contract

- Developers can work with only information local to a function by using contract to reason about interactions with callers and callees.

- If developers had more precise function specifications, they would be more productive because they do not need to read code outside their function.
How well does design by contract work?

- Specifications can’t capture all the information developers need for many tasks
- Developers need information about decisions that crosscut code
  - Interpretation of requirements & specifications
  - Representation of state in data structures
- Building good interface often as much work (or more) than doing implementation
  - Some need for more global view to make interface design decisions
Relevance model: Search

- Developers have very **specific** questions.
  - How [does] application state change when m is called denoting startup completion?
  - What resources are being acquired to cause this deadlock?

- Search tools enable developers to (1) directly **express** specific questions and (2) find **matching** information.

- Can increase productivity dramatically over structural relationship traversal.
Feedback: Gutter visualizations & live programming
Theory may offer new insight into human-code interactions

• Developers think about code in terms of facts about code.

• Developers seek to explain facts to discover the hidden implications of changes and prevent bugs.

• Explaining facts involves answering rationale questions about design decisions, which is hard with today’s tools.

• **Documentation** can help minimize information by directly communicating facts developers might need to know.

• But documentation lacks feedback, no mechanisms to check that code and documentation are consistent.
What if documentation had feedback?

Objectify is used for datastore serialization
Seems to be one of the more popular persistence frameworks for AppEngine.
Comment  Like  Dislike  Delete

Updates and deletes should be immediately stored
Not sure how long it takes to persist, but don’t want to have inconsistent state.

Call
ofy().save().entity(...)
35 matches

followed by
now()
30 satsified  5 missing

Comment  Like  Dislike  Delete

All Entity classes must register themselves
Objectify requires that classes be registered before they can be loaded or serialized. This is done in the class initializer for Project.

For classes with
@Entity
... class CLASSNAME
12 matches

Required call
ObjectifyService.register(CLASSNAME.class);
in Project.static()
12 satisfied
Conclusions

• Productivity in programming can be understood through modeling time & effort to obtain task-relevant information about code.

• Designing more effective human-code interactions involves increasing information minimization & feedback.

• Translating “proto-theories” about productivity into theory requires understanding relationships & gathering evidence to test.

• Theory can help lead to new avenues for empirical research, stronger evidence for (or against) productivity benefits, and new approaches for increasing developer productivity.