Model-Minded Development

George Fairbanks
GOTO Berlin
10 November 2016
How does your team build software?
How does your team build software?

Simple process

1. Get new requirement
2. Write test case
3. Edit code minimally so test passes
4. Refactor to remove code duplication
How does your team build software?

Simple process

1. Get new requirement
2. Write test case
3. Edit code minimally so test passes
4. Refactor to remove code duplication
Developers weave models

Developers keep in mind many abstract yet complex models that constrain the code they write

- Domain driven design
- Design patterns
- Architectural styles
- Test Driven Design (TDD)
- Programming styles (functional, OO, procedural, etc.)
- ...
What is Model-Minded Development?

It’s the general idea behind many specific ones:

Modeling should be a first-class activity in software development
Example behavior with/without models

**Without models**

1. Get new requirement
2. Write test case
3. Edit code minimally so test passes
4. Refactor to remove code duplication

**With models**

1. Get new requirement
2. Does it challenge our models?
   a. E.g., client-server still OK?
   b. E.g., domain model still OK?
3. Revise models
4. Write test case
5. Revise code to match model
This talk

Main sections:

1. Programming is **theory building**

2. Programming is **distributed cognition**

3. What are our software theories / models?
Programming is theory building
[P]rogramming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

Emphasis added
Understanding addition

Knowing that

Memorized facts

- Know that $3+4=7$
- Memorize addition table
- No answers after, eg, $6+6$

No understanding; no theory

Gilbert Ryle (1949): Knowing How and Knowing That
Understanding addition

Knowing that

Memorized facts

- Know that $3+4=7$
- Memorize addition table
- No answers after, eg, $6+6$

No understanding; no theory

Knowing how

General theory of addition

- Can add small numbers
- More mistakes with larger numbers
- Addition makes numbers bigger

Understanding; a theory

Gilbert Ryle (1949): Knowing How and Knowing That

theory \approx model
Understanding software

**Knowing that**

Memorized facts

- The Foo module handles reporting
- It uses Angular 2
- 25k LOC

No understanding; no theory

**Knowing how**

Understanding of the program

- Can give impromptu “chalk talk” about any aspect of the program
- Zooms in or out
- Knows what works well vs what is kluged
while (true) {
    observe world;
    if (surprise) {refactor theory};
}
Theory building in science

- Over time, tune theory to match observations
- Goal: theory matches future observations

Refactor the theory

How well I understand ≈ How well theory matches future observations
while (true) {
    pick up new requirement;
    if (surprise) {refactor theory};
}
Theory building in programming

- Over time, tune program to satisfy requirements
- Goal: avoid “dead ends”, program can always be adapted to next requirement

Refactor the theory
Elegant theories are better

Worse understanding

Better understanding
Often, though, continuous refactoring prepares the way for something less orderly. Each refinement of code and model gives developers a clearer view. This clarity creates the potential for a breakthrough of insights. A rush of change leads to a model that corresponds on a deeper level to the realities and priorities of the users. Versatility and explanatory power suddenly increase even as complexity evaporates.
Programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

Peter Naur (1985)

Programming as Theory Building
Programming should be regarded as a production of a program and certain other texts.

Every feature we add to this geocentric model makes it harder to add the next feature!
Example behavior with/without models

Without models

1. Get new requirement
2. Write test case
3. Edit code minimally so test passes
4. Refactor to remove code duplication

With models

1. Get new requirement
2. Does it challenge our models?
   a. E.g., client-server still OK?
   b. E.g., domain model still OK?
3. Revise models
4. Write test case
5. Revise code to match model
Building theories!
Programming is distributed cognition
Long division

\[
\begin{array}{c|c}
45 & 13095 \\
\hline
?? ?? \\
\hline
45 & 13095
\end{array}
\]
Long division

\[
\begin{array}{c|c}
45 & 13095 \\
90 & \\
- & \\
40 & \\
\end{array}
\]
Long division

\[
\begin{array}{c|c}
45 & 13095 \\
90 & \\
-- & \\
409 & \\
405 & \\
-- & \\
4 & \\
\end{array}
\]
Long division

\[ 291 \]
\[ \underline{\div 45} \]
\[ 13095 \]
\[ \underline{90} \]
\[ \underline{--} \]
\[ 409 \]
\[ \underline{405} \]
\[ \underline{---} \]
\[ 45 \]
\[ \underline{45} \]

With scribbles on paper, we can solve problems that don’t fit in our heads
Distributed cognition

- Distributed cognition is solving problems with “scribbles on paper”
- It can amplify cognitive ability
Distributed cognition

- Distributed cognition is solving problems with “scribbles on paper”
- It can amplify cognitive ability
- Programming requires it
  - What size program can you write in your head?
- Source code is our “scribbles” (our external representation)
Internal - external model alignment

?????

--------

45 | 13095

But not just any scribbles
Roman numerals, really?

XLV | XIIIIXCV
Arabic numerals, unhelpful positions
Misaligned models = poor performance

Diagram from Stroop effect, not long division
Distributed cognition is fragile:

“Different representations of a problem can have dramatic impact on problem difficulty even if the formal structures are the same.”

-- Jaijie Zhang and Donald Norman 1994
You and your scribbles

- It’s a **magic trick**
  - You + the scribbles

- It is **fragile**
  - Bad scribbles --> trick fails

- **If it’s just you** then
  - Use whatever scribbles you want
  - Write whatever code works for you

... what if you are on a **team**?
Teamwork + external representation

Ann 45

Diane

Bob 409 405

Carl 45 45
How do teams steer ships?
Challenges

Challenge 1: **Internal-external**
- Make sense of external representation (charts, logs, ...)

Challenge 2: **Team**
- No one person knows everything
- Team shares external representations
Success factors

Challenge 1: **Internal-external**
- Make sense of external representation (charts, logs, …)

Challenge 2: **Team**
- No one person knows everything
- Team shares external representations

Success requires:
- Good theories / models
- Good external representations
- Compatible models across team
Where are we?

- Talked about **theory building**
  - Parallels between science and programming

- Talked about **distributed cognition**
  - Can solve bigger problems with a pencil
  - ... but only if models align between paper and head

- So ... what are our models / theories?
Software models / theories
3 categories of models

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solution</th>
<th>Math, logic, &amp; friends</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3 categories of models

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solution</th>
<th>Math, logic, &amp; friends</th>
</tr>
</thead>
<tbody>
<tr>
<td>What your subject matter experts talk about</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Often: nouns, verbs, and relationships between them</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Long history in object-oriented programming</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 3 categories of models

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solution</th>
<th>Math, logic, &amp; friends</th>
</tr>
</thead>
</table>
| What your subject matter experts talk about | The tech known only by software engineers | ● Design patterns  
● Architecture models  
● Coding styles  
● Databases, event queues, data structures |
### 3 categories of models

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solution</th>
<th>Math, logic, &amp; friends</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>What your subject matter experts talk about</strong>&lt;br&gt;Often: nouns, verbs, and relationships between them</td>
<td><strong>The tech known only by software engineers</strong>&lt;br&gt;- Design patterns&lt;br&gt;- Architecture models&lt;br&gt;- Coding styles&lt;br&gt;- Databases, event queues, data structures</td>
<td><strong>Theories known across sciences and philosophy</strong>&lt;br&gt;- Logic&lt;br&gt;- Set theory&lt;br&gt;- Category theory&lt;br&gt;- Metamodeling</td>
</tr>
</tbody>
</table>
If you think it, express it in code

Code bridges internal and external
● Code is the “scribbles on paper”

Remember:
● Challenge 1: Internal-external
● Challenge 2: Team

Keep the magic trick working
Show all 3 models in code

Domain
- Ubiquitous language
- Data structure, object, & function names

Solution
- Pattern names appear in code
- Revealing architecture in code

Math, logic, & friends
- First class functions
- Use the libraries!
  - Sets.difference, not the spelled-out equivalent
# Reusable and ad hoc models

<table>
<thead>
<tr>
<th>Reusable models</th>
<th>Ad hoc models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td></td>
</tr>
<tr>
<td>● Industry standard ontologies</td>
<td>● Your domain model</td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td></td>
</tr>
<tr>
<td>● Patterns (gang of 4 book)</td>
<td>● Your patterns</td>
</tr>
<tr>
<td>● “Programming styles” book (Cristina Videira Lopes)</td>
<td>● Your styles</td>
</tr>
<tr>
<td><strong>Math, logic, &amp; friends</strong></td>
<td>● Logic, set theory, category theory, metamodeling</td>
</tr>
<tr>
<td>● Logic, set theory, category theory, metamodeling</td>
<td>● Rare</td>
</tr>
</tbody>
</table>
The “von Neumann architecture”
Imagine helping a friend’s startup, doing “design recovery”
You are recognizing existing patterns / models

- “Hey, it looks like your system, broadly speaking, has 3 tiers ...”
- “Is it reasonable to group these boxes as related to batch processing?”
- “Can you point out which boxes have persistent state, which are stateless?”

Reusable models must be linked
(Sitting in a book they do nothing)
Models and your project

- What are the important models for your team?

- Where do they live?
  - Speech, whiteboards, docs, code?

- How do they fit into your process?
  - When do you update them?
  - How strict are you?

Different projects, different answers
Model-Minded Development
Example behavior with/without models

**Without models**

1. Get new requirement
2. Write test case
3. Edit code minimally so test passes
4. **Refactor to remove code duplication**

**With models**

1. Get new requirement
2. **Does it challenge our models?**
   a. E.g., client-server still OK?
   b. E.g., domain model still OK?
3. **Revise models**
4. Write test case
5. Revise code to match model
Goal: “Balanced” systems and models

Systems and models ready for the next requirement

Refactor the theory
Challenge 1: **Internal-external**
- Make sense of external representation (charts, logs, …)

Challenge 2: **Team**
- No one person knows everything
- Team shares external representations

Success requires:
- Good theories / models
- Good external representations
- Compatible models across team
### 3 categories of models

<table>
<thead>
<tr>
<th>Domain</th>
<th>Solution</th>
<th>Math, logic, &amp; friends</th>
</tr>
</thead>
<tbody>
<tr>
<td>What your subject matter experts talk about</td>
<td>The tech known only by software engineers</td>
<td>Theories known across sciences and philosophy</td>
</tr>
</tbody>
</table>
| Often: nouns, verbs, and relationships between them | - Design patterns  
- Architecture models  
- Coding styles  
- Databases, event queues, data structures | - Logic  
- Set theory  
- Category theory  
- Metamodelling |
| Long history in object-oriented programming | Libraries in every language; core of functional programming |
Model-Minded Development

George Fairbanks
GOTO Berlin
10 November 2016