SWE 265P
Reverse Engineering and Modeling

Lecture 4

Duplication of course material for any purpose without the explicit written permission of the professor is prohibited.
“...always find the particular ‘point of interest’ and then do chaining – chain backwards (who calls this – then who calls that – then who calls that) and if you iterate enough, you’ll get back to main() at some point. You can also forward-chain all the way down into the utility libraries and the ‘deepest’ parts of the call stack, at least for the feature you are investigating.” – Eric Dashofy [General Manager & Deputy CIO, The Aerospace Corporation]
Today

• Last week’s material
• Key expert practices
• Structural vs behavioral models
• UML in more detail
• In-class practice
• Consuelo Lopez (MuleSoft)
Last week’s material

• Mental models
  – a representation of someone’s thought process of how something works in the real world
  – individual, uncertain, selective, flexible, dependent
  – external versus internal (software)
  – limitations

• Externalizing mental models
  – where have we been & where do we still need to go templates
  – UML class diagrams

• Videos

• Any questions?
Last week’s homework

• Which features did you choose to locate in the code?

• How did you approach locating the features?

• How difficult was it? (Why?)

• How confident are you that you found everywhere in the code where the features are implemented?

• What is the value of the diagram that you now have at hand?

• Any questions?
Key expert practices
KEP #1: focus on the essence
KEP #2: go as deep as needed
KEP #3: work with others
Structural versus behavioral models
## Structural versus behavioral models

<table>
<thead>
<tr>
<th>Structural</th>
<th>Behavioral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class diagram</td>
<td>Use case diagram</td>
</tr>
<tr>
<td>Package diagram</td>
<td>Activity diagram</td>
</tr>
<tr>
<td>Component diagram</td>
<td>Statechart diagram</td>
</tr>
<tr>
<td>Deployment diagram</td>
<td>Sequence diagram</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
- Clustering
  - response (end)
  - A+ guaranteed delivery
  - A+ see every state
  - A+ system-wide queuing
  - Pick up where left off
  - Debugging
  - Charting at every step
- Statistics
  - (WHO vs. trip)
  - (JMX?) / monitoring
- Threading
  - Message order w/ w/o out dot over
- Batching
  - Container for scripts
- Routing

DONKEY
DO NOT ERASE!

Source

F1

F2

F3

F4

F5

F6

F7

F8

F9

F10

F11

F12

Separate processing & archiving

1. No DB as temp store
   2. Map properly encoded

Single channel game folders:
1. Instant
2. Source transformed
3. Data transformed
April 4, 2011

Test design w/ LLP connector message flow:

Groovy + DSL (op overloading) = + delete

4/4/2011

- Test design w/ LLP connector message flow
- Groovy + DSL (op overloading) = + delete

- Triggers
  - Add observer pattern to register interceptors (state, alerting)

- Use common messages
  - Manage connectors

- Compile
  - Cache
  - Execute
Donkey Flow Diagram

Source

Destination

Source FT

Source Queue

Destination Queue

FT DT

Destination Connector Queue

DC

N* Destinations..
Channel Configuration with 3 destinations.
Destination 2 has a "wait for previous" flag set to true.
Destination 3 has a "wait for previous" flag set to false.
Channel is configured to respond from destination 3.
Channel configuration with 2 synchronized destinations and 1 unsynchronized destination. Channel responds from 1st destination.
Morals of the story

• Understanding what your current code really, really does is essential

• Modeling matters in the real world

• Structural and behavioral models are used

• Not all models are formal and precise
UML class diagram notation

abstract class

class
UML class diagram notation

[visibility] attribute [: type]
UML class diagram notation

[visibility] method([direction][parameter1][:: type], [direction][parameter2][:: type]...
, [direction][parametern][:: type])[: return type]
UML class diagram notation
UML class diagram notations

- association
- aggregation
- composition
- inheritance
- realization
- dependency
UML class diagram notations

0..1 0..1

1 n

n m

1..n 1..m
UML class diagram notation

realizes relation
UML class diagram notation

- **Inheritance**
  - Illustrated by the dashed line pointing from the base class to the derived class.

- **Aggregation**
  - Illustrated by the solid line pointing from a whole to a part.
Break
UML class diagram
Following the trail
Call graphs
Going backwards (call graphs)
Important frequent question #2 and #3

What parts of the code rely on this feature (or class, or method, or variable, or...)?

What parts of the code does this feature (or class, or method, or variable, or...) rely on?
Homework (team)

• With your team, decide upon two features that are essential in your system, and imagine that each of the two features will need to undergo some kind of change to be implemented by someone else.

• Prepare a packet, per feature, that would assist that other person in understanding where the feature is located, what other parts of the system may be relevant, and how those parts are relevant.
Homework (team)

• Due date: start of class next week

• Submit via a GitHub pull request that creates a homework_2 folder in your team’s folder, as many files as you need, with the following naming convention:
  – hw2_<team_name>_<filename>.<extension>

• Bring a printed copy of your diagrams

• Start early
Homework (individual)

- https://online.visual-paradigm.com/diagrams/tutorials/sequence-diagram-tutorial/
- https://www.youtube.com/watch?v=UI6lqHOVHic
- https://www.youtube.com/watch?v=pCK6prSq8aw
Homework (individual)

- Make sure to regularly update your personal diary, including an entry for today’s lecture
Optional advanced material

- Study design structure matrices
And now…

• ...welcome Consuelo Lopez!