Testing-Based Trace Analysis

Exploring the Mapping between Requirements, Architecture, and Code

Alexander Egyed

Teknowledge Corporation

Annual Research Review (ARR)
March, 2005
Video-On-Demand System

Client

Server

By K. Dohyung
## Requirements

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td>Users should be able to display a list of available movies and select one from the list</td>
</tr>
<tr>
<td>r1</td>
<td>Play movie immediately after selection from list</td>
</tr>
<tr>
<td>r2</td>
<td>Users should be able to display textual information about a selected movie</td>
</tr>
<tr>
<td>r3</td>
<td>User should be able to pause a movie</td>
</tr>
<tr>
<td>r4</td>
<td>3 seconds max to load movie list</td>
</tr>
<tr>
<td>r5</td>
<td>3 seconds max to load textual information about a movie</td>
</tr>
<tr>
<td>r6</td>
<td>1 second max to start playing a movie</td>
</tr>
<tr>
<td>r7</td>
<td>Provide VCR-like user interface</td>
</tr>
<tr>
<td>r8</td>
<td>User should be able to stop a movie</td>
</tr>
<tr>
<td>r9</td>
<td>User should be able to start a movie</td>
</tr>
</tbody>
</table>
Architecture (Structure)

Structural Decomposition (class diagram)
Architecture (Behavior)

Behavior of Movie Display Class (statechart diagram)
Behavior of Streamer Class (statechart diagram)

Viewpoint of the “Playing” Part (Class Diagram)
Architecture (Scenario)

User → Display → Streamer

- User: select
- Display: play, connect, wait
- Streamer: stream
Low-Level Design
Java Code
Users should be able to display a list of available movies and select one from the list.

Play movie immediately after selection from list.

Users should be able to display textual information about a selected movie.

User should be able to pause a movie.

3 seconds max to load movie list.

3 seconds max to load textual information about a movie.

1 second max to start playing a movie.

Provide VCR-like user interface.

User should be able to stop a movie.

User should be able to start a movie.

Distributed Models

But One System
What are Trace Dependencies?

• Describe similarity among software artifacts
  - Requirements
  - Model elements (architecture, design)
  - Code

• Origin, rationale, decision, realization, impact
  - Why is this here, what happens if I change this, are we finished yet
Why Trace Dependencies?

- Understanding / Navigation
- Consistency checking
- Code generation
- Impact analysis
- Conflict analysis
- Requirements trade-off analysis
- Personnel turnover
- Evolution

*Links but not Mechanism!*
Consistency (1)

r1: Play movie automatically after selection from list *(Functionality)*

![Diagram showing the functionality of the video player](image-url)
Traceability (1)

rc: Play movie automatically after selection from list (Functionality)
Traceability (2)
How to find Trace Dependencies?

• Everybody ‘does’ trace dependencies
  – At least in their mind
  – Sometimes even documented (trace matrix)
• Tools for recording trace dependencies
  – Still a manual process, often complete trace required
Traceability Problem

the hard way

- requires the semantics of any two artifacts to understand the single trace between them
- formal, semi-formal, and informal knowledge
- the existence of one-to-many or even many-to-many traces
- context
- misused models and notations
- model change => trace change

$n^2$ complexity
| r0   | Users should be able to display a list of available movies and select one from the list |
| r1   | Play movie immediately after selection from list |
| r2   | Users should be able to display textual information about a selected movie |
| r3   | User should be able to pause a movie |
| r4   | 3 seconds max to load movie list |
| r5   | 3 seconds max to load textual information about a movie |
| r6   | 1 second max to start playing a movie |
| r7   | Provide VCR-like user interface |
| r8   | User should be able to stop a movie |
| r9   | User should be able to start a movie |

1,891 traces

\( n^*(n-1)/2 \) traces

348,195 traces

42,486 traces
Leveling the Playing field

- If you have an executable system
  - Run usage scenarios on the system \(\textcolor{green}{\text{automatable}}\)
  - Observe the execution trace (footprint) \(\textcolor{green}{\text{automated}}\)
  - Map usage scenarios to model elements \(\textcolor{red}{\text{manual}}\)

\(\Rightarrow\) Traceability from model elements to code
**R6:** One second max to start playing a movie

- **Behavior:** stop
- **Behavior:** play

+ Requirement \( <\rightarrow \) Play
+ Requirement \( <x> \) Stop

This is a footprint
r6: one seconds max to start playing a movie
Input

Model Element A

Footprint

Model Element B

Usage Scenario(s)

Footprint
Expressing Uncertainties

without sacrificing correctness

• Partiality Uncertainty:
  - isExactly
  - isNot
  - isAtLeast
  - isAtMost
  - ...

• Cluster (Grouping) Uncertainty
  - \([a,b]\) is \([1,2]\)
Reasoning with Uncertainties

**Structure**

Dependency:

\[
\text{[display, server\_access, streamer]} \text{ isExactly } [\text{A, C, D, F, G, I, K, O}];
\]

Dependency:

\[
\text{[main, movie\_sel, server\_access]} \text{ isAtMost } [\text{C, J, N, R, U}];
\]
Dependency:
\([\text{display, server\_access, streamer}] \text{ isExactly } [\text{A, C, D, F, G, I, K, O}]\); 

\(\text{//s about display, server\_access, streamer}\)

\([\text{A, C, D, F, G, I, K, O}]\)

\(\text{Includes: display, server\_access, streamer}\)

\([\text{B, E, H, J, M, N, P, Q, R, S, T, U}]\)

\(\text{Excludes: display, server\_access, streamer}\)

\(\text{//is not about display, server\_access, streamer}\)
Dependency:

[display, server_access, streamer] isExactly [A,C,D,F,G,I,K,O];

Dependency:

[main, movie_sel, server_access] isAtMost [C,J,N,R,U];

[C,J,N,R,U]
Includes: main, movie_sel, server_access

[A,C,D,F,G,I,K,O]
Includes: display, server_access, streamer

[J,N,R,U]
Excludes: display, server_access, streamer

[B,E,H,L,M,P,Q,S,T]
Excludes: main, display, movie_sel, server_access, streamer

[A,D,F,G,I,K,O]
Excludes: main, movie_sel, server_access

[C]
Included: server_access

[?] in
rest
out
The Trace Analyzer Tool

Three input dependencies produce the following results:

Handles uncertainties, detects shared code, tells about incompleteness, and uncovers inconsistencies
Automating Trace Analysis
the easier way

- Semantic differences among models is irrelevant
- Only 'n' input hypotheses required
- Developer may use their idiosyncratic models, context, notation, formal or informal
- BUT: need source code (incremental, prototype, legacy code)

Input: model to common representation mapping
- manual (user)
- testing, if source code available
Thank you!

Alexander Egyed
aegyed@teknowledge.com