

Traceability and Software Execution

Philipp Huber, Michael Schneider & Appiah-Bonna Emmanuel Twumasi

Relate session topics

Results/Answers to the assignment

1. How can traceability links be generated by observing the execution of software?
2. How can propagate changes in source code be propagated to requirements?

Relate session topics

 [Ramesh 2001]
Toward Reference Models for
Requirements Traceability

Establishing Traceability
Models

 [Jarke 2005]
Requirements Tracing

 [Dick 2005]
Design Traceability

Requirements and Design
Traceability

Automate
Tracing

 [Egyed 2002]

Trace back
from SW

 [BenCharrada 2012]

**Traceability and Software
Execution**

Traceability and Requirements Evolution

Generate Traceability Links by Observing the Execution of Software

❑ Trace Analyzer Tool

to generate traceability links through

- Transitive reasoning
Requirements → Software Artefacts
- Shared use of a “common ground” = Software

❑ Proposed in reading: Egyed2002.pdf

❑ Demonstrated based on a Video-On-Demand Software with complex interdependencies

Trace Analyser Tool Approach

- Define Requirements
- Define Scenarios
- Define Links
- Perform Scenarios by using the software
- Record Traces by Analyser Tool (Tool: Rational PureCoverage)

Requirements

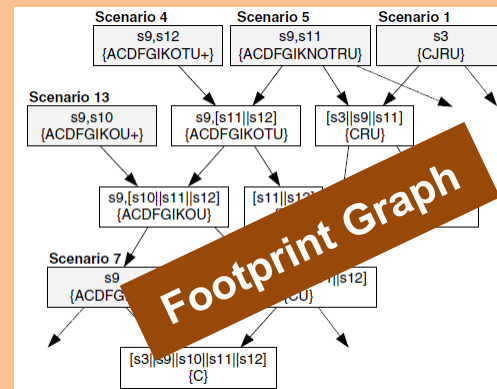
r0	Users should be able to display a list of available movies and select one from the list
r1	Play movie immediately after selection from list
r2	Users should be able to display information about a selected movie
r3	User should be able to select a movie
r4	3 seconds after selection
r5	3 seconds after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

Scenarios

Test Scenario	Artifact	Observed Java Classes
1. view movie list	[s3]	[C,J,R,U]
2. view textual movie information	[s4,s6][r2]	[C,E,J,N,R]
3. select/play movie	[s8,s9][r6]	[A,C,D,F,G,I,J,K,N,O,T,R,U]
4. press pause	[s9,s12][r8]	[A,C,D,F,G,I,K,O,T,U]
5. select movie	[s9,s11]	[A,C,D,F,G,I,K,N,O,T,R,U]
6. stop movie	[r3]	[C,R,U]
7. get textual movie information	[r4]	[A,C,D,F,G,I,K,N,O,T,R,U]
8. movie list	[r7]	[A,C,D,F,G,I,K,N,O,T,R,U]
9. VCR-like UI	[r0]	[C,J,N,R,T,U]
10. select movie	[r1]	[A,C,D,F,G,I,J,K,N,O,T,R,U]
11. select/play movie	[s9,s10]	[A,C,D,F,G,I,K,O,U]
12. press pause	[s9,s10][r3]	[A,C,D,F,G,I,K,O,U]

Links

Used Classes



Footprint Graph

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	R	S	T	U
r0			F							F				F				F		F
r1	F		F	F		F	F		F	F	F			F				F		F
r2			F		F															
r3	F		F	F		F	F		F		F									F
r4																				
r5																				
r6	F		F	F		F												F		F
r7	F		F	F														F		F
r8	F																			F
r9																				F
s3										P								P		P
s9										F	F	F				F				
s10																				P
s11			P												P			P		P
s12			P															P		P

Requirements – Class dependencies

Define Requirements, Scenarios and Links

Requirements

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

defined
requirements r0 – r9

implemented state
transitions s1 – s12

Scenarios

Links

<i>Test Scenario</i>	<i>Artifact</i>
1. view movie list	[s3]
2. view textual movie information	[s4,s6][r2]
3. select/play movie	[s8,s9][r6]
4. press stop button	[s9,s12][r8]
5. press play button	[s9,s11][r9]
6. change server	[s5,s7]
7. playing	[s9]
8. get textual movie information	[r5]
9. movie list	[r4]
10. VCR-like UI	[r7]
11. select movie	[r0]
12. select/play movie	[r1]
13. press pause	[s9,s10][r3]

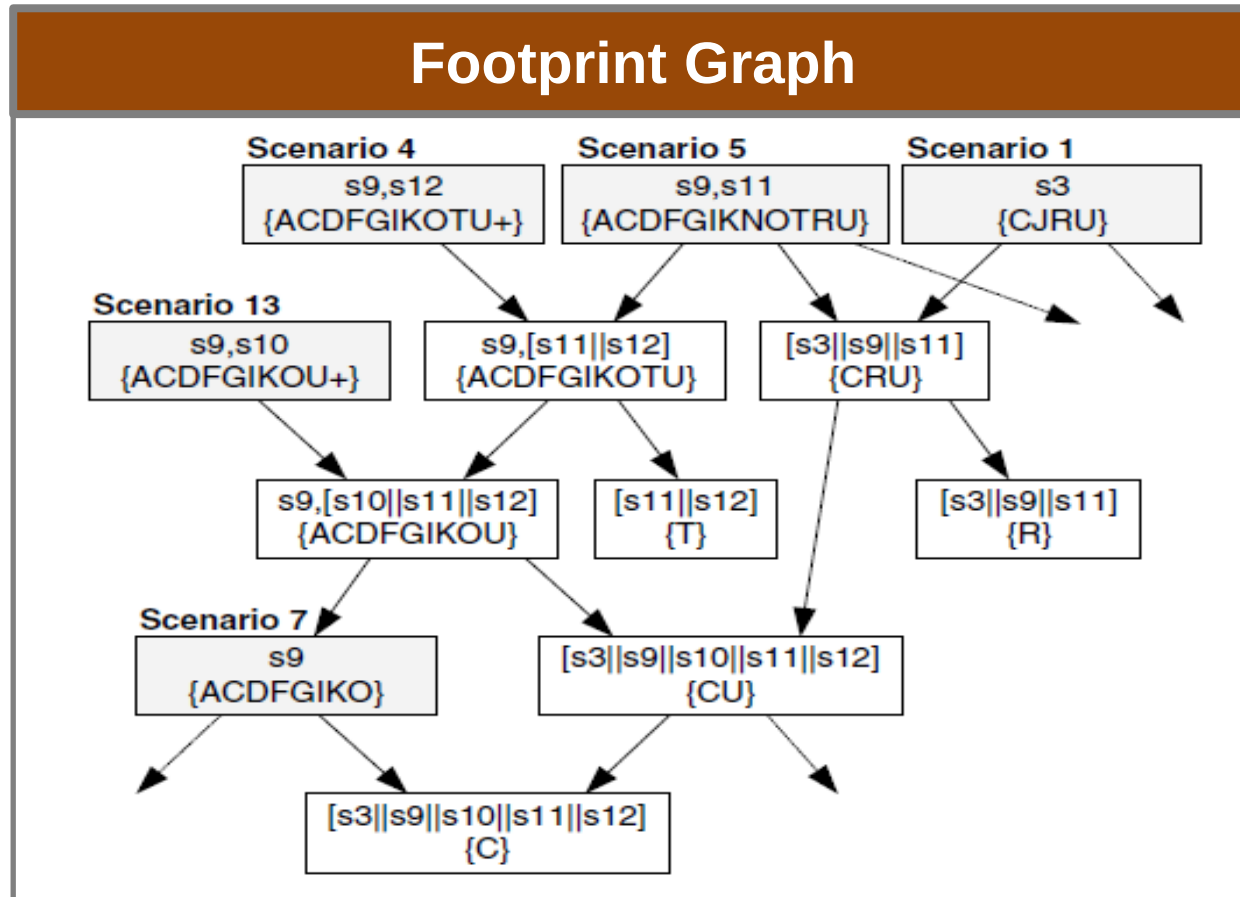
Trace Analyser Results (1/3)

Scenarios	Links	Used Classes
<i>Test Scenario</i>	<i>Artifact</i>	<i>Observed Java Classes</i>
1. view movie list	[s3]	[C,J,R,U]
2. view textual movie information	[s4,s6][r2]	[C,E,J,N,R]
3. select/play movie	[s8,s9][r6]	[A,C,D,F,G,I,J,K,N,O,T,R,U]
4. press stop button	[s9,s12][r8]	[A,C,D,F,G,I,K,O,T,U]
5. press play button	[s9,s11][r9]	[A,C,D,F,G,I,K,N,O,T,R,U]
6. change server	[s5,s7]	[C,R,J,S]
7. playing	[s9]	[A,C,D,F,G,I,K,O]
8. get textual movie information	[r5]	[N,R]
9. movie list	[r4]	[R]
10. VCR-like UI	[r7]	[A,C,D,F,G,I,K,N,O,R,T,U]
11. select movie	[r0]	[C,J,N,R,T,U]
12. select/play movie	[r1]	[A,C,D,F,G,I,J,K,N,O,R,T,U]
13. press pause	[s9,s10][r3]	[A,C,D,F,G,I,K,O,U]



Tracing between Scenarios, Requirements, State Transitions and Implemented Classes

Trace Analyser Results (2/3)



Tracing/Graph between Scenarios, State Transitions and Implemented Classes

Trace Analyser Results (3/3)

Requirements – Class dependencies

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
r0			F							F				F				F		F	F
r1	F		F	F		F	F		F	F	F			F	F			F		F	F
r2			F		F					F				F				F			
r3	F		F	F		F	F		F		F				F						F
r4																		F			
r5														F				F			
r6	F		F	F		F	F		F	F	F			F	F			F		F	F
r7	F		F	F		F	F		F		F			F	F			F		F	F
r8	F		F	F		F	F		F		F				F					F	F
r9	F		F	F		F	F		F		F			F	F			F		F	F
s3			P							P								P			P
s9	F		P	F		F	F		F		F				F						
s10			P																		P
s11			P											P				P		P	P
s12			P																	P	P

implemented
classes A-U

require-
ments
r0-r9

state
changes
s3-s12



Bi-directional Tracing between Requirements and implemented classes / source code

Identifying Outdated Requirements Based on Source Code Changes

- ❑ Approach to trace SW changes back to requirements specification
 - Analyse changes in source code
 - Identify outdated requirements specification
- ❑ Two main components of this approach
 - Code differencing technique:
Identify source code changes that may affect requirements
 - Techniques for extracting relevant keywords for these changes
(by automated tool)
- ❑ Proposed in reading: BenCharrada2012.pdf

Identify Outdated Requirements

Proceeding

- ❑ Identify Relevant Changes in source code
 - Changes in methods bodies
 - Additions of new elements (classes; methods; package; fields)
 - Additions and removals of element
 - Changes in methods signature
 - Changes in private elements affect external behaviour of the system
 - Additions of several methods
- ❑ Extract Keywords
 - Names of change related elements, commented code changes
 - Filtering and Grouping
- ❑ Identify Affected Requirements
 - Use standard Information Retrieval (IR) tools to find affected/missing requirements based on the extracted keywords

Identify Outdated Requirements Findings

- ❑ Whereas IR-tools are available for step 3, also step 1 and 2 can be automated significantly
- ❑ Two case studies on an existing SW application have been performed.
Evaluation of this approach:
 - Successfully found most of the relevant changes
 - Not relevant changes (e.g. pure technical changes) are successfully left
 - Low effort
 - Results are in most of the cases better than by class-based tracing techniques
- ❑ Two main limitation of the approach
 - It can miss some relevant requirements changes
 - It does not generate traceability links that are 100% correct

Question 1

??? Why/when is the software code the better
common ground = starting point
to identify changed requirements?

Question 2

??? What is your experience in tracing requirements to software code?
(forward – backward)