RESEARCH CENTRE ON SOFTWARE TECHNOLOGY

Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications

Giuseppe A. Di Lucca

Massimiliano Di Penta

RCOST - Research Centre on Software Technology,
University of Sannio - Benevento, Italy

Comprehension of today Web Applications: main problems and open issues

- WAs more and more highly interactive and *dynamic*
  - HTML pages can be dynamically built by *server pages*
  - pieces of code (e.g., client-side scripts) can be dynamically generated
- Current RE of WA is mainly based on *static* code analysis
- Pure static analysis of pages would not suffice to deal with the WA dynamicity
  - Static analysis is likely to give only an imprecise and approximate picture
- Dynamic analysis needed for a proper understanding of
  - the client-side logic (changing according to user inputs)
  - session and cookie data
  - DBMS tables and queried entities
  - the frequency of exercising a particular link
  - the type of link actually exercised (e.g., hyperlinks, submit with GET or POST)
  - ……

Integration of static and dynamic analysis

WA static & dynamic analysis have to be properly integrated and complemented

- static analysis is not able to capture the dynamicity of the WA
- a pure dynamic analysis may fail to detect features/components that are not exercised by the execution of instrumented WAs

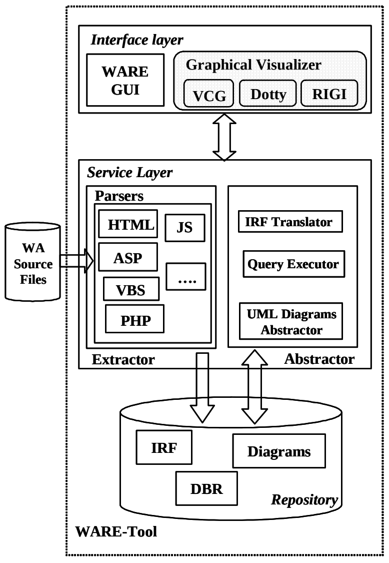
Two WA analyzers have been integrated

- WARE supporting the RE of WA by static analysis
- WANDA that extracts facts from execution traces of an instrumented WA.

successfully adopted in the past to perform RE tasks over WAs

The two tools have been integrated/improved to identify group of equivalent Built Client Pages (BCPs).

WARE - *Web Applications Reverse Engineering*

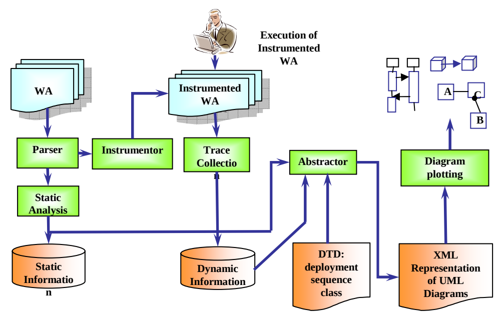


An integrated environment supporting the recovery of design/analysis documents (UML diagrams)

A three layers architecture

- *Interface Layer* : the user interface providing access to the functions offered by the tool
- *Service Layer* : implements the tool services
  - *Extractor*, static analyzer, retrieve relevant information and store it in IRF files
  - *Abstractor*, to abstract further information and documents
- *Repository* stores facts extracted/abstracted from the WA mapped onto a relational database

WANDA - Web ApplicatioNs Dynamic Analyzer



WA automatically instrumented and information collected during WA executions stored into a database.

Information contained in the database is then used to abstract UML diagrams

- *interface layer*: to access the original WA and the traces collected by the execution of the instrumented WA; responsible for the visualization of the results and diagrams.
- *business logic layer*: the analysis and transformation subsystems; it manipulates the information at various levels of abstraction
- *data layer*: responsible for storing the extracted static and dynamic information, as well as the abstracted UML documentation

WARE & WANDA integration and improvement

Mainly by making compatible and consistent the data stored in the databases produced by the two tools.

– tables of the WANDA database were modified to use the same identifiers produced by WARE

– some 'correspondence' tables were created as well.

a very low impact of modifications on the code

Main improvements:

- the static production of a Page Control Flow Graph (PCFG) and the identification of the Linearly Independent Paths (LIPs) in the server pages
- the automatic instrumentation of the server pages to record the actual executed paths along user sessions and the LIPs coverage analysis.

The new features of the integrated environment have been implemented as a Java application, accessible by a new user interface added to the pre-existing ones

Built Client Pages (BCPs)

- Client pages dynamically built, at run time, by server pages
- BCPs may make difficult the comprehension of a WA, when an appropriate documentation does not exist
  - the BCPs generated from executions of the same server page can differ each other

    (e.g., a page containing a form and an error page, according to user inputs)
  - the resulting set can be very large.

To reduce the comprehension effort, the BCPs can be grouped into equivalence classes, where each equivalence class will include a set of BCPs sharing common features

Identification and Grouping of Equivalent Built Client Pages (1)

A ***Built Client Pages*** is made up of a ***control component***

the set of items - such as the HTML code and scripts - determining the page layout, business rule processing, and event management

and of a ***data component***

the set of items - such as text, images, multimedia objects - determining the information to be read/displayed from/to a user

- Pages with the ***same control component***, but different data component, can be considered as ***equivalent pages***, belonging to a same ***Equivalence Class***

  they show a same behaviour/they have the same meaning
- A single *equivalent page* will be used to represent each Equivalence Class of equivalent Built Client Pages of a given Server Page

  clone detection techniques can be used to identify clusters of equivalent BCPs

Identification and Grouping of Equivalent Built Client Pages (2)

Identification of clusters of equivalent BCPs usually obtained by exploiting the clone detection techniques

e.g. according to a Levenshtein distance over structural information

this may be expensive

it requires that all the possible kind of BCPs are generated and then clone analysis to be carried out

A new method exploiting the results of static and dynamic analysis of the WARE and WANDA

- to reduce the effort for identifying the BCP equivalence classes
- to know if all the possible BCPs from a server pages have been considered

Identification and Grouping of Equivalent Built Client Pages (3)

A stepped process performed for each server page generating BCPs:

1. represent the page code by a Page Control Flow Graph (PCFG);
2. identify the Linear Independent Paths (LIPs) in the PCFG;
3. identify the LIPs along which any BCPs is generated;
4. identify, by analysing the traces of the executed server page paths, the LIPs, or combination of LIPs, covered by the executions generating BCPs;
5. group in the same equivalence class the BCPs generated by the executed paths covering the same LIPs or combination of LIPs

The steps of the process

Represent the page code by a Page Control Flow Graph (PCFG)

The code of server pages is be modeled by a PCFG

- A Node represents a sequence of statements unconditionally executed, or a predicate of a conditional statement branching the control flow.
  - The statements may be script statements, HTML tags including text sentences, …
- The Edges in the PCFG represent the control flow transfer between statements; edges from predicate nodes are labeled with (TRUE, FALSE).

obtained using WARE static analysis capabilities

Identify the Linear Independent Paths in the PCFG generating BCPs

- The PCFG of each server page is statically analyzed to identify the LIPs it includes.
- The statements along each LIP are examined to identify the ones generating BCPs.
  - The LIPs not including any statement generateing a BCP will nomore considered

The WARE tool provides to compute the number of LIPs for each server page.

The steps of the process

Execution traces analysis

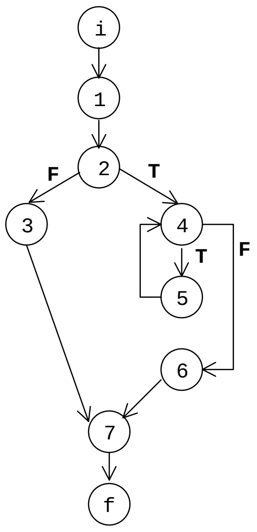
- The instrumented WA is executed and user execution traces are collected, by WANDA, over a period of time
- The collected execution traces of the server pages are analyzed to identify which LIPs have been covered in each execution generating a BCP
- Each BCP is associated to the covered LIP or combination of LIPs whose execution generated it.

Identifying the BCP equivalence classes

- The BCPs generated by the execution of the same LIPs, or combination of LIPs, are equivalent BCPs.
  - they will have the same *control component* or just differ because a subset of the *control component* is present more than once in the page, due to the execution of cycles along the PCFG

  equivalent BCPs may have a different *data component*.
- All the BCPs associated to the same LIPs, or LIPs combination, are grouped into an equivalence class

A simple example



Let us consider a server page SP whose PCFG is showed in the figure

three LIPs are identified :

1) i, 1, 2, 4, 6, 7, f;
2) i, 1, 2, 4, 5, 4, 6, 7, f;
3) i, 1, 2, 3, 7, f;

each LIP is represented by the node sequence it includes

- some BCPs generated along the paths *1* and *2*
- no BCP is generated along path *3*

A simple example

Let us assume that the following set of Execution Paths (EP) has been recorded for the page SP:

EP(SP) = {

a) (i, 1, 2, 4, 5, 4, 6, 7, f),
b) (i, 1, 2, 4, 6, 7, f)
c) (i, 1, 2, 4, 5, 4, 5, 4, 5, 4, 6, 7, f)
d) (i, 1, 2, 3, 7, f)
e) (i, 1, 2, 4, 6, 7, f)
f) (i, 1, 2, 3, 7, f)
g) (i, 1, 2, 4, 5, 4, 5, 4, 6, 7, f)
h) (i, 1, 2, 3, 7, f)

}

the traces *d*, *f*, *h*, are not considered, corresponding to LIPs with no BCPs

Two Equivalence Classes:

EC1 = (*b*, *e*) and EC2 = (a, c, g)

A simple example

EC1 = (*b*, *e*)
b) (i, 1, 2, 4, 6, 7, f)
e) (i, 1, 2, 4, 6, 7, f)

EC2 = (*a*, *c*, *g*)
a) (i, 1, 2, **4, 5, 4**, 6, 7, f),
c) (i, 1, 2, **4, 5, 4, 5, 4, 5, 4,** 6, 7, f)
g) (i, 1, 2, **4, 5, 4, 5, 4,** 6, 7, f)

BCPs in EC1 have the same control component

They are generated by executing just the same sequence of statements in SP

BCPs grouped in EC2 generated along paths differing each other due to the different number of repeated executions of nodes *4* and *5*.

Nevertheless, we consider that the control component of these BCPs is the same because they, eventually, differ for some repeated elements (such as a different number of rows in a table) without changing the page behavior/meaning

A Case Study

The effectiveness of the proposed approach has been validated by applying it on some small/medium sized WAs

The results from a (real-world) small WA providing information for tourist itineraries in the Sannio countryside, and allowing to make reservations for trips by bus (*http://www.pietrelcinaservice.it)*

Server pages coded by PHP script language

Main results from the WARE tool analysis:

- 26 PHP server pages and 12 HTML static client pages.
- 23 server pages dynamically generate at least a BCPs,

  not possible to identify haw many different pages were generated by each server page
- The PCFGs of the 23 server pages were drawn and the LIPs of each of them identified
- most of the server pages generate more than 2 BCPs. .

A Case Study

The application was instrumented and dynamically analyzed by WANDA :

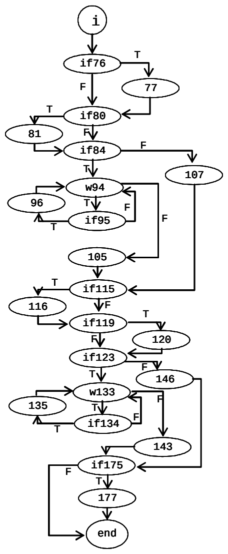
- 209 execution traces were recorded
  - each execution trace corresponded to a user session
  - all the PCFG paths exercised in each server page were recorded

- The analysis of the execution traces revealed a large number of BCPs,
- The exercised PCFG paths corresponding to each server page executions were selected and grouped together.
- To identify which LIPs each path trace covered, each group of path traces was analyzed with respect to the LIPs of corresponding executed server pages,.
- The path traces generating a BCPs and covering the same LIP, or combination of LIPs, were identified and the BCPs generated along them were associated to a same equivalence class.

The dynamic analysis allowed to identify links between BCPs and between BCPs and the static pages, too

A Case Study

Results from the server page carrello.php (shopping cart)



#	Path Traces
1	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - w94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end
2	i - IF76 - IF80 - IF84 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
3	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end
4	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 – W133 - IF175 - 177 - end
5	i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
6	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end
7	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - W133 - IF175 - 177 - end
8	i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
….	…….
20	i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end

- 12 LIPs (the highest number of LIPs in a page)
- 20 executions of this page were considered
- A BCP was generated along each of the executed paths.
- the executed paths were grouped into 7 equivalence classes, according to the LIP they covered
- each class corresponds to a BCP equivalence class.

A Case Study

#	EC Composition	Covered LIP
1	1-3-6	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end
2	2-9-16-19	i - IF76 - IF80 - IF84 - W94 – 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
3	4-7-13	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - 177 - end
4	5-8-11	i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
5	10-12-17-18-20	i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end
6	14	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 – IF115 – IF119 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end
7	15	i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 – IF115 – IF119 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end

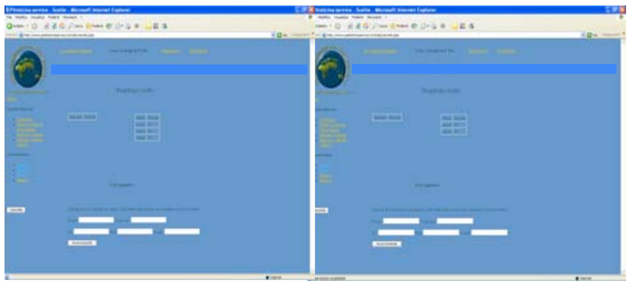
- ‘EC Composition’: identifiers of the path traces grouped into each equivalence class
- ‘Covered LIP’ : covered LIPs in the PCFG

- The paths grouped in each class just differ for the number of times some elements (e.g., table rows) appear in the page

  Usually that number is determined from user inputs.

- not all the LIPs were covered by the considered executed paths, thus other BCPs equivalence classes may be defined for the uncovered LIPs

A Case Study



Equivalent BCPs grouped in the EC 2

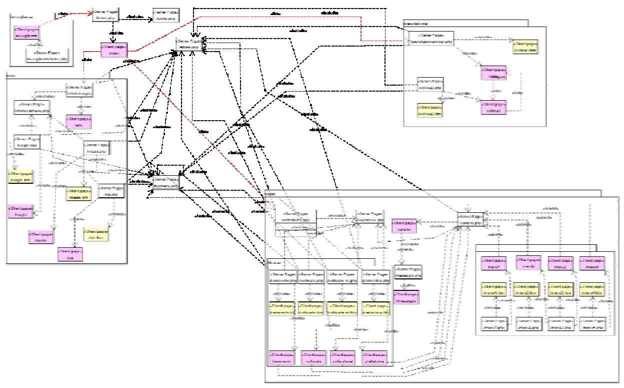
A manual validation of the BCP Equivalence Classes was carried out:

- Every equivalence class actually included equivalent BCPs
- some equivalence classes could be merged together because they included groups of similar pages
  - the method can be improved by better distinguishing between *‘similar LIPs’*

    i.e. different LIPs that do not produce radical changes in the control component of the equivalent BCPs.

A Case Study

The clustering of BCP into equivalence classes allowed to produce UML class diagrams (where equivalence classes are represented) by far better understandable than the original ones (containing all BCPs), thus improving the WA comprehension



Conclusions and future work

- Static analysis of existing Was has been already used in a number of approaches proposed to gain WA comprehension, while dynamic analysis has been used by a fewer approaches
- both WA static and dynamic analysis has been exploited to identify the client pages dynamically generated at run time by server pages and to group together sets of similar built client pages to be considered as a single equivalent page.
- Equivalent BCPs are identified by grouping together the ones due the execution of the same LIPS in a server page.
- The results from a case study showed that the approach effectively allowed the set of BCPs to be correctly identified as well as the groups of equivalent BCPs.
    - The case study also showed that the greater is the number of collected execution traces, the better are the results the approach produces.
- Future work will be devoted to identify ‘similar LIPs’ in the server pages, whose identification will allow reducing the number of BCPs equivalence classes, and thus the comprehension effort.