Crawlability Metrics for Web Applications

N. Alshahwan 1, M. Harman 1, A. Marchetto 2, R. Tiella 2, P. Tonella 2

1University College London, UK
2Fondazione Bruno Kessler, Trento, Italy

ICST 2012 - Montreal - Apr 19, 2012
Motivations and Context

Automated exploration (crawling) of web applications is substantial from many perspective:

- for testing and validity checking client-side content (links, html content, etc);
- for exposing server-side failures such as crashes, errors and exceptions;
- for constructing models of web applications.

However, some websites are easier to explore than others

- For static sites, exploring the application reduces to a graph traversal task,
- In general, the problem is undecidable and potentially infinite.
A sample crawling scenario

- A crawler is exploring a web application ...

[ dash lines mark links and actions not discovered yet ]
A sample crawling scenario

- Using an initial URL, the crawler downloads the first page (home page):

- Page 1 contains Form 2
- The crawler could choose either to follow the link from Page 1 or to try to submit a request for the form ...
- ... suppose it follows the link ...
A sample crawling scenario

- It discovers Page 2 and Form 1 ...

- It fills in Form 1 following some input data generation strategy, e.g. providing some random strings, or some more specific data randomly drawn from some predefined sets such as integers, emails, dates, etc.
A sample crawling scenario

- It discovers Page 6 which turns out to be a blind alley ...

- The Web Application is reset, the crawler restarts from the seeding URL, it reaches again Form 1,
- ... and tries other inputs ...
A sample crawling scenario

- It discovers Page 7 which turns out to be another blind alley ...

- The Web Application is reset, the crawler restarts from the seeding URL, it reaches again Form 1, and tries other inputs ...
- ... again, and again ...
- ... but every time it ends up in Page 6 or Page 7
- ... after a timeout it decides to try to submit to Form 2 ...
A sample crawling scenario

- But the Crawler is sent back to Page 1 ...

- ... again, it tries with different inputs but ...
- ... every time it ends up in Page 1 ...
- ... after a timeout it decides to stop as no other actions could be tried.
Crawlability

- Crawlability \(\simeq\) how easy is for a crawler to explore a WebApp

- Crawlability depends on:
  - what part of the application was discovered according to some application structure
  - the power of the crawler, i.e. its capabilities
  - crawlability of forms (as links are easily traversed)
The structure of an application is defined by a “conceptual model”:
- Conceptual client pages, i.e. sets of (client) pages classified as equivalent (e.g. search result pages)
- Forms
- Server side components

A Crawler is characterized by:
- Its input generation capabilities:
  - Random strings, legal (test) credentials, numbers, email, dates, etc.
- Its page classification algorithm:
  - Page name or title, string comparison, tree comparison, etc.
Crawlability Metrics

- Crawlability metrics (CRAW) try to capture our intuition for the concept of “crawlability” for a form.

- They are differently defined in different usage scenarios:
  - If an application model is available:
    \[
    CRAW = \frac{\#\text{Discovered Pages}}{\#\text{Pages}}
    \]
  - Otherwise (the common case):
    \[
    CRAW = \frac{\text{Dynamic metric for actual coverage}}{\text{Static metric for possible coverage}}
    \]
Crawlability Metrics

Static Metrics (possible coverage / computed from source code):

- **STM [statements]**
  Number of server-side executable statements

- **ICC [interprocedural cyclomatic complexity]**
  McCabe’s cyclomatic complexity computed on the CFG (control flow graph) obtained after in-lining of procedures/functions.

- **SICC [sliced ICC]**
  ICC computed on the part of the CFG which is retained after conditioned slicing using constraints from a crawler request.

Dynamic Metrics (actual coverage / depend on crawler’s activity):

- **CSTM [covered statements]**
  Statements actually covered handling the set of requests performed on a form

- **FOUT [fan-out]**
  Number of different pages discovered by the crawler behind a form
Crawlability Metrics

Combining dynamic and static metrics:

\[ CRAW_0 = \frac{CSTM}{STM} \quad \text{i.e. coverage ratio} \]

\[ CRAW_1 = \frac{FOUT}{ICC} \]

\[ CRAW_2 = \frac{FOUT}{SICC} \]

Rationale:

- \( CRAW_0 \): there’s a linear relationship between the number of generated (discovered) pages and the size of the executable (executed) code

- \( CRAW_1 \) (\( CRAW_2 \)): \( ICC \) (\( SICC \)) estimates the total number of pages: a page corresponds to a linear-independent path in the CFG
<?php
$request = $_SERVER['REQUEST_METHOD'];
if ($request == "GET") {
    if (!isset($_SESSION['message'])) {
        $_SESSION['message'] = "";
    } ?>
<h1>New User</h1>
<form action="registration.php" method="post">
    Name: <input name="userid" type="text"/>
    Password: <input name="passwd1" type="password"/>
    Password Confirmation: <input name="passwd2" type="password"/>
    <input type="submit" value="Register"/>
    <input name="action" type="hidden" value="register"/>
</form>
<?php
} else if ($request == "POST") {
    switch($_POST["action"]) {
    case "register":
        check_data($_POST);
        // registration code ...
        break;
    default:
        header("Location: internal_error.php");
    }
} else { header("Location: internal_error.php"); }
?>
<?php
$request = $_SERVER['REQUEST_METHOD'];
if ($request == "GET") {
    if (!isset($_SESSION['message'])) {
        $_SESSION['message'] = "";
    } ?>
    <h1>New User</h1>
    <form action="registration.php" method="post">
        Name: <input name="userid" type="text"/><br/>
        Password: <input name="passwd1" type="password"/><br/>
        Password Confirmation: <input name="passwd2" type="password"/><br/>
        <input type="submit" value="Register" />
        <input name="action" type="hidden" value="register" />
    </form>
<?php
} else if ($request == "POST") {
    switch($_POST["action"]) {
    case "register":
        check_data($_POST);
        // registration code ...
        break;
    default:
        header("Location: internal_error.php");
    }
} else { header("Location: internal_error.php"); } ?>
Sliced ICC - Motivation for using - 2/3

\[ ICC = 5 + ICC'(check\_data) \]
If a specific form type is concerned ...

\[ SICC = 1 + SICC'(\text{check\_data}) \]

```
(2) inputs();

(3) $request = \$_SERVER["REQUEST\_METHOD"];

(4) $request == "GET"

⇒

\$_SERVER["REQUEST\_METHOD"] == "POST" \land \$_POST["action"] == "register"

(6) $tmp1

(5) $tmp1 = (!isset($$_SESSION["message"]));

(10) ($$_POST["action"] == "register")

(9) $request == "POST"

(13) header("Location: internal\_error.php");

(12) header("Location: internal\_error.php");

(11) check\_data($$_POST);

(14) end;

(8) echo("html\_1");
```
Testing Framework

- by ranking forms by increasing crawlability, testers can be guided to provide test inputs for forms which are more likely to hide unexplored parts of the application under test.
Evaluation - Research questions

- **RQ1 - Crawlability as indicator of unexplored pages:** Are the crawlability metrics good indicators of crawlability, thus of the presence of unexplored pages?

- **RQ2 - Role of conditioned slicing:** Does conditioned slicing improve crawlability metrics based on the computation of the interprocedural cyclomatic complexity?
Evaluation - Research questions

- **DFOUT [Delta Fan OUT]**: # of distinct conceptual client pages the crawler is unable to download when a given form is reached.

- To answer RQ1:
  - Compute correlation (Spearman’s $\rho$) between $CRAW$ and $DFOUT$
  - Train a threshold-based binary classifier to predict whether $DFOUT > 0$ given $CRAW$ as input

- To answer RQ2:
  - Consider correlation and prediction accuracy with and w/o conditioned slicing.
For each subject the following steps were performed:

1. installation
2. crawling with WATT tool for 10 hours
3. automatic computation of metrics $CRAW_0$, $CRAW_1$, $CRAW_2$
4. manual computation of $DFOUT$
5. data correlation and classifier training/evaluation
The WATT crawler - inputs generation and strategy

WATT crawler was run with the following capabilities:

- words (strings with only letters), numbers and generic strings
- correct credentials for login forms

WATT crawling strategy for the experiment:

1. It starts from the initial URL and performs a depth first visit.
2. The next action (form or link) is chosen randomly from the set of untried available actions on the last visited page.
3. If no untried actions available, the crawler selects randomly from all actions.
4. The sequence of actions (trace) performed is saved (information about forms and links, input values used).
5. A trace is terminated when:
   - a page with neither links nor forms is reached,
   - no new page is encountered,
   - a user-defined number of actions is reached,
   - a user-defined time limit is reached.
6. Whenever the crawling is restarted, the state of the application, i.e. the database content, is reset to its initial value.
The WATT crawler - page classification

WATT crawler classifies pages based on a similarity metric $Sim(P_i, P_j)$:

$$P_i \simeq P_j \text{ iff } Sim(P_i, P_j) \geq \sigma$$

where similarity is computed from the sequences of html tags of the two pages:

$$Sim(P_i, P_j) = 2 \times \frac{|LCS(Seq(P_i), Seq(P_j))|}{|Seq(P_i)| + |Seq(P_j)|}$$

Where:

- $\sigma \leq 1$ is fixed
- $Seq(P_i)$ is the sequence of tags for page $P_i$.
- $LCS(X, Y)$ is the longest common subsequence of $X$ and $Y$.
- $\bar{X} = \langle \bar{x}_k \rangle_{k=1,\ldots,m}$ is subsequence of $X = \langle x_i \rangle_{i=1,\ldots,n}$ iff $\bar{x}_k = x_{i_k}$ and $1 \leq i_k < i_{k+1} \leq n$. 

## Evaluation - Subjects and Crawling activity

### Subjects:

<table>
<thead>
<tr>
<th>Web app</th>
<th>PHP files</th>
<th>Lines of PHP code</th>
<th>Downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAQForge</td>
<td>19</td>
<td>834</td>
<td>5,264</td>
</tr>
<tr>
<td>WebChess</td>
<td>24</td>
<td>2,701</td>
<td>24,751</td>
</tr>
<tr>
<td>News Pro</td>
<td>30</td>
<td>5,473</td>
<td>n.a.</td>
</tr>
<tr>
<td>TimeClock</td>
<td>62</td>
<td>14,980</td>
<td>22,328</td>
</tr>
</tbody>
</table>

### WATT crawler activity (10 hrs):

<table>
<thead>
<tr>
<th>Web app</th>
<th>Discovered</th>
<th>Total</th>
<th>Requests on</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pages</td>
<td>Forms</td>
<td>Traces</td>
</tr>
<tr>
<td>FAQForge</td>
<td>22</td>
<td>23</td>
<td>6,676</td>
</tr>
<tr>
<td>WebChess</td>
<td>28</td>
<td>65</td>
<td>784</td>
</tr>
<tr>
<td>News Pro</td>
<td>58</td>
<td>17</td>
<td>12,033</td>
</tr>
<tr>
<td>TimeClock</td>
<td>65</td>
<td>21</td>
<td>112</td>
</tr>
</tbody>
</table>
Evaluation - Experimental Results

Plot of metrics $DFOUT$ and $CRAW_2$ for the application $WebChess$ (samples are sorted by increasing $CRAW_2$).
## Evaluation - Experimental Results - Correlation

<table>
<thead>
<tr>
<th>Web app</th>
<th>Rho</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CRAW(_0) = Coverage</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAQForge</td>
<td>-0.049</td>
<td>0.823</td>
</tr>
<tr>
<td>WebChess</td>
<td>0.172</td>
<td>0.172</td>
</tr>
<tr>
<td>News Pro</td>
<td>-0.661</td>
<td>0.003</td>
</tr>
<tr>
<td>TimeClock</td>
<td>-0.395</td>
<td>0.003</td>
</tr>
</tbody>
</table>

| **CRAW\(_1\) = FOUT / ICC** |
| FAQForge    | -0.23  | 0.290  |
| WebChess    | -0.05  | 0.681  |
| News Pro    | -0.76  | 0.002  |
| TimeClock   | -0.87  | < 0.001 |

| **CRAW\(_2\) = FOUT / SICC** |
| FAQForge    | -0.09  | 0.681  |
| WebChess    | -0.72  | < 0.001 |
| News Pro    | -0.54  | 0.04   |
| TimeClock   | -0.86  | < 0.001 |

- **RQ1:**
  - CRAW\(_2\) presents a (negative) correlation with \(DFOUT\) in three out of four cases
  - CRAW\(_k\) presents a (negative) correlation with \(DFOUT\) in two out of four cases

- **RQ2:**
  - CRAW\(_2\) correlates for WebChess, while CRAW\(_0\) and CRAW\(_1\) not
  - FAQForge: due to limitations of the slicing algorithm \(SICC\) is overestimated
Evaluation - Experimental Results - Classifier

Results for the *leave-one-out* cross-validation:

<table>
<thead>
<tr>
<th>Web app</th>
<th>CRAW₀</th>
<th>CRAW₁</th>
<th>CRAW₂</th>
<th>a Priori</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAQForge</td>
<td>0.35</td>
<td>0.39</td>
<td>0.61</td>
<td>0.87</td>
</tr>
<tr>
<td>WebChess</td>
<td>0.51</td>
<td>0.51</td>
<td>0.95</td>
<td>0.62</td>
</tr>
<tr>
<td>News Pro</td>
<td>0.82</td>
<td>0.94</td>
<td>0.88</td>
<td>0.59</td>
</tr>
<tr>
<td>TimeClock</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
<td>0.75</td>
</tr>
</tbody>
</table>

*Percentage of correctly guessed samples*

- For three out of our four subject, a classifier trained on CRAW₂ can predict with high accuracy which forms are crawlable and which are not.
- On WebChess CRAW₂ is the only metric which performs better than the “a priori” classifier.
Conclusions and Future Work

The work proposed:

- A set of metrics to assess crawlability of web applications,
- A study which:
  - validated such metrics on real world web applications and
  - showed that simple metrics ($CRAW_0$) are quite effective with well modularized web sites, while dispatcher-based web sites may require more sophisticated metrics, e.g. $CRAW_2$ which resort to conditioned slicing.

Future work comprises:

- Improvements of the WATT crawler’s capabilities (e.g. input data generation from database content) and the slicing algorithm,
- A deeper investigation of the relation between effectiveness of metrics and application architectures,
- The extension of the study to a bigger set of applications.
Thanks!