Automated Conversion of Table-based Websites to Structured Stylesheets Using Table Recognition and Clone Detection

Andy Y. Mao James R. Cordy Thomas R. Dean

School of Computing
Queen's University
Kingston, Ontario, Canada
{mao, cordy, dean}@cs.queensu.ca

Abstract

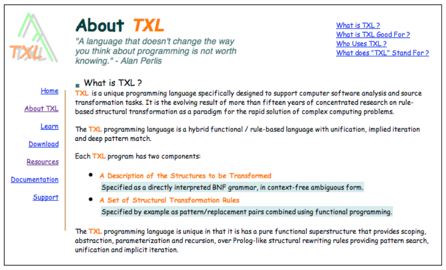
Web standards such as XHTML and CSS are rapidly coming into practice and have many advantages, including compatibility, consistency across browsers, and increased ease of maintenance. Unfortunately large numbers of existing websites still use the deprecated table-based layout style in which page style is unique to each page. Existing tools for automating the transition to stylesheets provide little help, converting page-by-page using a flattened structure and local inline styles rather than a common CSS stylesheet. This approach ignores hierarchical structure and defeats the main purpose of moving to the new standard, losing all of the advantages.

In this work we present an automated method for converting table-based layout websites to standards-compliant modern CSS stylesheet-based websites using a two-step process. Pages of the site are first converted page-by-page using table recognition technology to preserve hierarchical structure and layout semantics in local styles. Software clone detection technology is then utilized to recognize common layout styles in the pages and extract and minimize them to a common CSS stylesheet for the site. The result is a maintainable, efficient modern standards-compliant website with the same look and feel as the original but with all the maintenance advantages of a custom programmed new site.

1 Introduction

Long before the maturity of World Wide Web standards, websites implemented standard layouts and look-and-feel of pages using table-based layouts that are copied from one page to another, often because the original sites were generated by early website editors such as Claris Home Page. Many of these websites are still alive and actively maintained, and indeed a large number of popular websites still use traditional table-based layouts. Now that web standards designed for expressing and maintaining common layout and style such as XHTML, DIV layout and separate CSS stylesheets have matured, it is highly desirable to migrate existing legacy websites to the new technology.

The use of a separate common CSS stylesheet for a site is an example of the clear advantages offered by such a conversion. Suppose, for example, that we wished to change an entire website from left-handed logo form to right-handed, as shown in Figure 1. To make this change to a traditional table-based layout site, every single page of the site would have



About TXL

"A language that doesn't change the way you think about programming is not worth knowing." - Alan Perlis

What is TXL ?
What is TXL Good For ?
Who Uses TXL ?
What does "TXL" Stand For ?

Home
About TXL
Learn
Download
Resources
Documentation
Support

What is TXL ?

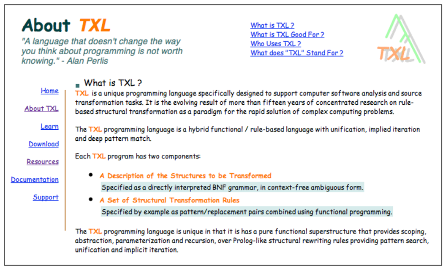
TXL is a unique programming language specifically designed to support computer software analysis and source transformation tasks. It is the evolving result of more than fifteen years of concentrated research on rule-based structural transformation as a paradigm for the rapid solution of complex computing problems.

The TXL programming language is a hybrid functional / rule-based language with unification, implied iteration and deep pattern match.

Each TXL program has two components:

- A Description of the Structures to be Transformed
  Specified as a directly interpreted BNF grammar, in context-free ambiguous form.
- A Set of Structural Transformation Rules
  Specified by example as pattern/replacement pairs combined using functional programming.

The TXL programming language is unique in that it is has a pure functional superstructure that provides scoping, abstraction, parameterization and recursion, over Prolog-like structural rewriting rules providing pattern search, unification and implicit iteration.



About TXL

"A language that doesn't change the way you think about programming is not worth knowing." - Alan Perlis

What is TXL ?
What is TXL Good For ?
Who Uses TXL ?
What does "TXL" Stand For ?

Home
About TXL
Learn
Download
Resources
Documentation
Support

What is TXL ?

TXL is a unique programming language specifically designed to support computer software analysis and source transformation tasks. It is the evolving result of more than fifteen years of concentrated research on rule-based structural transformation as a paradigm for the rapid solution of complex computing problems.

The TXL programming language is a hybrid functional / rule-based language with unification, implied iteration and deep pattern match.

Each TXL program has two components:

- A Description of the Structures to be Transformed
  Specified as a directly interpreted BNF grammar, in context-free ambiguous form.
- A Set of Structural Transformation Rules
  Specified by example as pattern/replacement pairs combined using functional programming.

The TXL programming language is unique in that it is has a pure functional superstructure that provides scoping, abstraction, parameterization and recursion, over Prolog-like structural rewriting rules providing pattern search, unification and implicit iteration.

Figure 1: Left- to Right-handed Logo Example

to be hand edited to move table elements between columns one by one using copy-paste. The amount of work and level of tedium in implementing even this simple change would almost certainly lead to errors and anomalies. By contrast, implementing this change in a common CSS stylesheet version of the same site would involve only change to one style in the stylesheet file, leaving all pages untouched and vastly reducing the effort and chances for error.

In addition to the clear advantage of a consistent common style across a site, web standards also offer many other advantages, including compatibility with modern website editing and searching tools, greater browser independence, and enhanced ease of maintenance. Ideally every website should be redesigned to use these new standards from scratch, but in practice the effort to do so for large websites with substantial investment can be prohibitive.

Existing automation for migrating table-based legacy websites to the new web standards such as that offered by Adobe Dreamweaver [1] is at best cursory, preserving layout of individual pages separately by absolute pixel placement and localized inline DIV styles, thus losing all hierarchical structure and commonality of style. The result is a conversion equivalent to per-page plotting of page elements (Figure 2), yielding a website that is actually less maintainable than the original and defeating the whole purpose of moving to the new standards.

In this paper we propose a more realistic and ambitious automated conversion, leveraging table analysis and source transformation technology already proven in the document recognition and software reengineering domains. Our conversion recognizes and preserves hierarchical structure and commonality of style across

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html> <head>
<title>Table test</title>
<style type="text/css">
   div { border:1px red solid;}
</style>
</head>

<body>
<div id="Layer1" style="position:absolute;
   left:15px; top:18px; width:244px; height:22px;
   z-index:1; vertical-align:middle">content 1</div>
<div id="Layer2" style="position:absolute;
   left:286px; top:18px; width:294px; height:46px;
   z-index:2; vertical-align:middle">content 2</div>
<div id="Layer3" style="position:absolute;
   left:610px; top:18px; width:161px; height:22px;
   z-index:3; vertical-align:middle">content 3</div>
<div id="Layer4" style="position:absolute;
   left:15px; top:42px; width:244px; height:22px;
   z-index:4; vertical-align:middle">content 4</div>
   . . .
</body>
</html>
```

Figure 2: Example Dreamweaver Conversion
Positions are absolute, style attributes are embedded and all table hierarchy is lost.

the entire site. The result of this automated conversion is a site that preserves the look and feel of the pages of the original site while preserving hierarchical layout structure. A common CSS style file which is essentially identical to one that would be authored in a disciplined hand-crafted migration is inferred from style similarity (Figure 3).

Our method utilizes a four step approach, in which web pages are first converted from HTML to XHTML using a source transformation based on robust parsing [8]. The table structure of each page is then analyzed using table recognition methods [22] to separate lay-

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html> <head>
<title>Table test</title>
</head>

<style>
#top_left {float:left; margin:auto; width:250px}
#top_left_container_1 {float:left; margin:0;
    width:250px}
#top_left_content_1 {float:left; margin:auto;
    width:250px; border:1px red solid;}
    . . .
</style>

<body>
<div id="top_left">
  <div id="top_left_container_1">
    <div id="top_left_content_1">
      content 1
    </div>
    <br clear="both"/>
  </div>
  <div id="top_left_container_1">
    <div id="top_left_content_1">
      content 4
    </div>
    <br clear="both"/>
  </div>
</div>
   . . .
</body>
</html>
```

Figure 3: Example Conversion by Our Process
Positions relative, style attributes in a separate stylesheet and nesting hierarchy preserved.

out table structure from intentional data tables and to elucidate implicit hierarchy represented by row-spanning (ROWSPAN) and column-spanning (COLSPAN) attributes as explicit nested tables. The augmented explicit table layout structure is then page-by-page converted to the web standard DIV-based layout with a separate CSS style file for each page, preserving the original look and feel. Finally, software clone detection technology [14] is utilized to recognize common styles and synthesize a single minimized CSS stylesheet file for the site, converting each page to use the common stylesheet. The entire process architecture can be visualized as shown in Figure 4.

The remainder of this paper is organized as follows. Section 2 outlines the phases of our process in detail and gives small examples of each transformation on example HTML code.

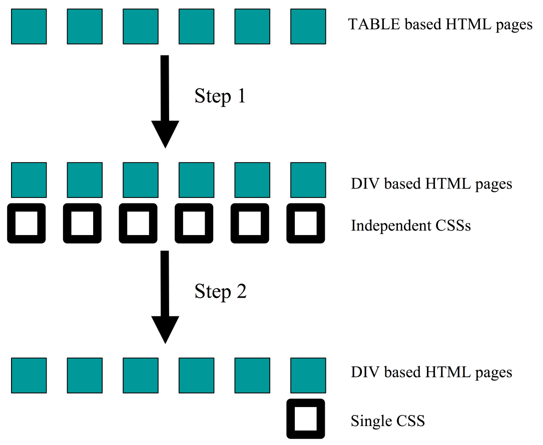


Figure 4: Conceptual Process
Our process consists of two major steps. In Step 1, table recognition is used to infer and preserve hierarchical structure in a conversion from tables to DIVs, separating style information into a stylesheet for each page. In Step 2, clone detection is used to unify and minimize styles into a single consistent stylesheet file for the entire site.

Section 3 demonstrates the entire process using our experience in converting a real entire table-based legacy website to modern XHTML / CSS standards. Section 4 relates our work to that of others, and Section 5 discusses limitations and future extensions of our process. Finally, Section 6 summarizes and concludes the paper.

2 Approach

The overall purpose of our approach is to provide an automated transformation system that preserves the layout structure of the pages of a web site specified by HTML table layout into a nested hierarchical DIV structure while retaining the original look and feel, and to recognize and extract DIV styles to a single unified, minimized, maintainable CSS style file for the site. The first part is achieved by converting each page separately, creating a CSS stylesheet for it independent of the others. The second part uses clone detection techniques to recognize and minimize common styles into a single unified stylesheet used by all the pages. Figure 4 shows a conceptual view of our method.

All steps of our approach are implemented as source transformations implemented in the TXL source transformation language [6]. Each phase consists of a number of source transformations, strung together to achieve the result (Figure 5). In this section we outline the details of these transformations, using small examples to demonstrate the technique.

2.1 XHTML Conversion

In the first transformation, web pages are independently converted from HTML to XHTML using a TXL grammar that utilizes robust parsing [2] to correct for HTML exceptions to XML form. Robust parsing is a method that attempts to parse each page as an XHTML document, adapting to exceptions where for example closing tags are missing. The exceptions are isolated into special nonterminal forms that are then targeted for correction by TXL source transformation rules, resulting in a valid XHTML page. Figure 6 shows an example of this transformation.

2.2 Table Recognition

To preserve the original hierarchical layout semantics of the original table layout pages into nested DIV sections, we must first understand what the intended structure is. In HTML table layout, some of the intended structure is encoded in the ROWSPAN and COLSPAN attributes of table elements (Figure 7). Since DIV sections have no such corresponding feature, we must first make this implicit substructure explicit by transforming the original pages to eliminate ROWSPAN and COLSPAN while retaining the layout semantics they imply.

In order to do this we have adapted ideas from the table recognition literature in pattern analysis and machine intelligence research [22]. The methods we have adapted are called *projection* and *partitioning*. The basic idea is that a table cell that spans two or more other cells in a row or column implies a projected or nested structure on parallel rows or columns that embeds their corresponding cells in a sub-table. Table recognition methods such as Handley [9] and Hu et al. [11] use this idea in analysis of higher level structure of tables in documents.

In our case we have implemented this idea using a table partitioning and nesting transformation which forms a part of our conversion to DIV structures. To assist in the analysis, we compute an approximate layout for each page by assigning position information to every table cell using custom attributes. This information is used in the analysis only - the final result is constrained only by the page's original style attributes. The conversion proceeds by partitioning ROWSPAN and COLSPAN structures into nested tables, separating them from parallel unspanned rows or columns and reducing all table structures to simple ones without any spans. Nesting of tables retains the relationship between the elements so that layout is not lost. Figure 8 shows the result of table partitioning the example of Figure 7.

In some table layout sites, there could of course be a conflict between rowspans as shown in Figure 9. Just as such cases cause problems for table recognition algorithms, it leads to an ambiguity for our method and our prototype conversion system requires hand intervention to handle such (relatively rare) cases.

2.3 Table Identification

While the same HTML "TABLE" feature is used, not every table in a web page represents layout information - some tables are intended to actually be data tables. As part of our conversion, we must identify which tables should be converted to DIV structures and which not. In our prototype, this decision is made on a very simple criterion - if the HTML table structure has a table header (THEADER) or table footer (TFOOTER) tag (i.e., if it has labeled columns or rows), then the table is assumed to represent a real data table and is not converted.

The identification of tables to be converted is done using another TXL source transformation that marks layout tables to be converted to DIV using a custom XML tag that also gives each table a unique name for use in attaching CSS styles to it later. Figure 10 shows part of the result of table identification on an example page.

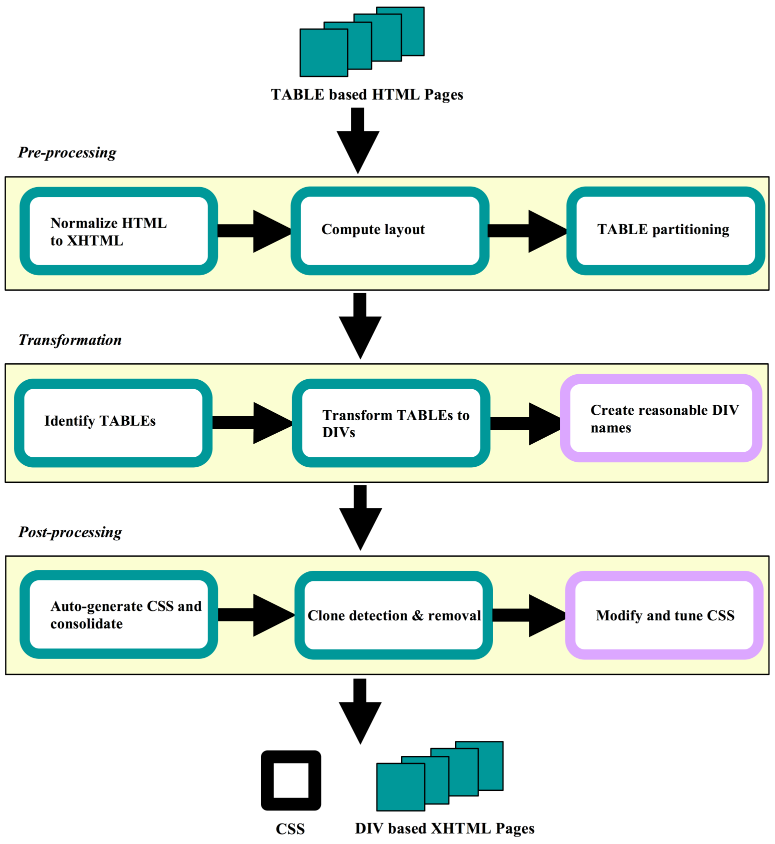


Figure 5: Implementation Architecture

2.4 Conversion to DIVs with Local Styles

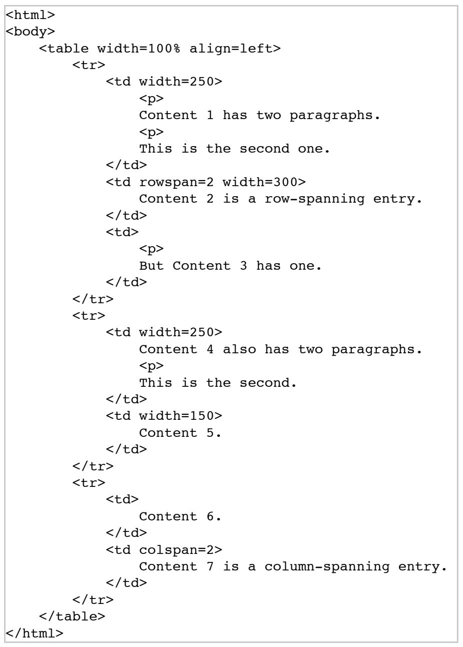
Following table partitioning and identification, identified layout tables in pages are converted from table structures to DIV partitions for each table element, each with its own individual local inline style preserving the style attributes of the original table cell. Figure 11 shows part of the result of the DIV conversion of an example page. Relative positioning implied by table rows is maintained in the result using the FLOAT="LEFT" style attribute.

Like all of our stages, the conversion to DIV is done using TXL source transformation rules. Figure 12 shows the main transformation rule *replaceTableByDIV* for identified tables. In the usual TXL style, this one rule automatically searches to match and convert every identified table in the input. As part of the transformation, it uses the transformation subrule *replaceTrByDIV* to convert each row of the table, and so on. The generated DIVs are uniquely identified by the table id generated in the table identification step. These ids will attach each DIV to its corresponding style in the next step.

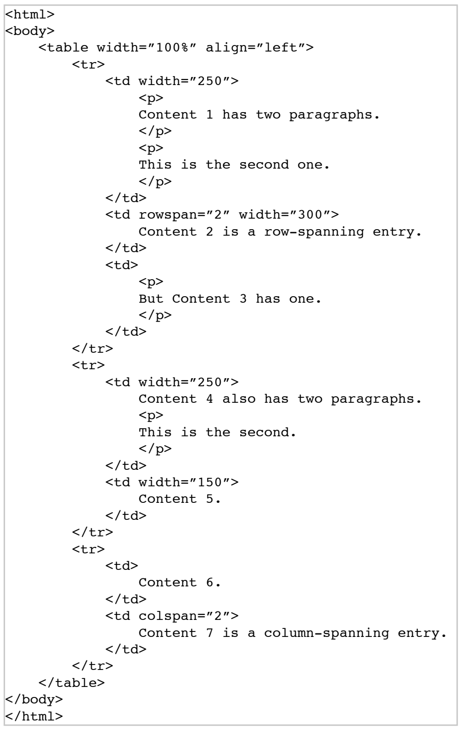
A web-based human interface allows the operator of the conversion process to choose more meaningful names for the generated DIVs at this stage (Figure 13). As the operator enters new names, the interface automatically changes the XHTML source of the generated DIVs to correspond. Figure 14 shows the converted DIV example of Figure 11 after renaming.

2.5 Separation of CSS Style Files

Following the conversion to DIV form with inline local styles, another transformation gathers and converts all styles in each page into an individual CSS style file for the page, using the table ids of the previous step. This is done

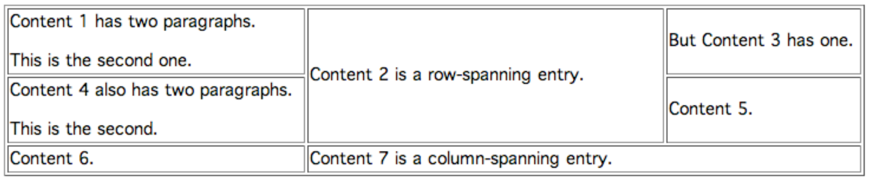


```
<html>
<body>
    <table width=100% align=left>
        <tr>
            <td width=250>
                <p>
                Content 1 has two paragraphs.
                <p>
                This is the second one.
            </td>
            <td rowspan=2 width=300>
                Content 2 is a row-spanning entry.
            </td>
            <td>
                <p>
                But Content 3 has one.
            </td>
        </tr>
        <tr>
            <td width=250>
                Content 4 also has two paragraphs.
                <p>
                This is the second.
            </td>
            <td width=150>
                Content 5.
            </td>
        </tr>
        <tr>
            <td>
                Content 6.
            </td>
            <td colspan=2>
                Content 7 is a column-spanning entry.
            </td>
        </tr>
    </table>
</html>
```



```
<html>
<body>
    <table width=”100%” align=”left”>
        <tr>
            <td width=”250”>
                <p>
                Content 1 has two paragraphs.
                </p>
                <p>
                This is the second one.
                </p>
            </td>
            <td rowspan=”2” width=”300”>
                Content 2 is a row-spanning entry.
            </td>
            <td>
                <p>
                But Content 3 has one.
                </p>
            </td>
        </tr>
        <tr>
            <td width=”250”>
                Content 4 also has two paragraphs.
                <p>
                This is the second.
                </p>
            </td>
            <td width=”150”>
                Content 5.
            </td>
        </tr>
        <tr>
            <td>
                Content 6.
            </td>
            <td colspan=”2”>
                Content 7 is a column-spanning entry.
            </td>
        </tr>
    </table>
</body>
</html>
```

Figure 6: Conversion to XHTML



Content 1 has two paragraphs. This is the second one.	Content 2 is a row-spanning entry.	But Content 3 has one.
Content 4 also has two paragraphs. This is the second.		Content 5.
Content 6.	Content 7 is a column-spanning entry.	

Figure 7: Row- and Column-spans in Table Layout
The layout specified by the table in Figure 6. (Borders shown to make the layout visible.)

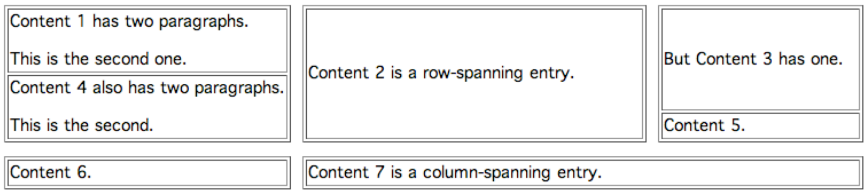
using a TXL transformation that extracts the local style parameters such as font and alignment from each DIV and creates a CSS style for it, named using the DIV's unique table id. As part of this transformation, the order of parameters in each generated style is normalized by sorting into alphabetical order in order to allow so that similar styles are more easily detected in the next phase.

Figure 15 shows a portion of the corresponding extracted CSS style file for the example of Figure 11. Following this step all pages of the site have been converted to DIV layout, each page with its own individual CSS style file.

2.6 Clone Detection on Styles

While the DIV conversion results in a completely migrated XHTML, DIV and CSS-based web standard website, it still has the undesirable property that each page has its own CSS style file. The remaining problem is the integration of these styles to a single uniform stylesheet for the entire website. To achieve this result we employ clone detection technology [14] borrowed from our previous software re-engineering work [7] to recognize similar styles across pages and integrate them into a single global CSS stylesheet file.

The process of CSS style clone detection is



Content 1 has two paragraphs.

This is the second one.

Content 2 is a row-spanning entry.

But Content 3 has one.

Content 4 also has two paragraphs.

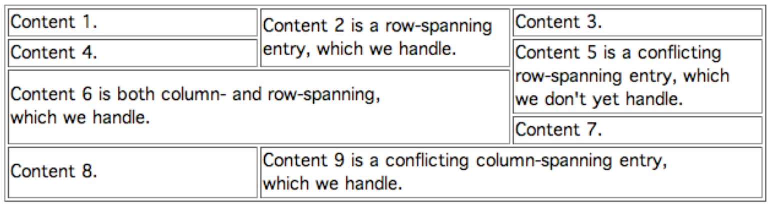
This is the second.

Content 5.

Content 6.

Content 7 is a column-spanning entry.

Figure 8: Table Partitioning to Eliminate Row and Column Spanning
Table partitioning converts COLSPAN and ROWSPAN attributes to equivalent nested table structures, reducing all layout tables to simple ones. (Borders shown to make the layout visible.)



Content 1.

Content 2 is a row-spanning entry, which we handle.

Content 3.

Content 4.

Content 5 is a conflicting row-spanning entry, which we don't yet handle.

Content 6 is both column- and row-spanning, which we handle.

Content 7.

Content 8.

Content 9 is a conflicting column-spanning entry, which we handle.

Figure 9: Conflictual Rowspan Example
Our prototype is not yet able to automatically handle row partitioning for this example, and hand assistance is required. (Borders shown to make the layout visible.)

```
<tag id="table1">
<table  float="left">
    <tag id="table1_tr1">
    <tr>
        <tag id="table1_tr1_td1">
        <td  width="250"  widthi="266">
        content 1
        </td>
        </tag>
    </tr>
    </tag>
    <tag id="table1_tr2">
    <tr>
        <tag id="table1_tr2_td1">
        <td  width="250"  widthi="266">
        content 4
        </td>
        </tag>
    </tr>
    </tag>
</table>
</tag>
```

Figure 10: Table Identification Example
XML custom tags ("<tag>") mark and uniquely name each component of tables identified as layout tables.

achieved by two linked source transformations, one which works on the pages' CSS files to detect and unify style clones and another that

```
<div id="table1" float="left">
    <div id="table1_tr1">
        <div id="table1_tr1_td1" width="250"
         widthi="266">
            content 1
        </div>
        <br clear="both"/>
    </div>
    <div id="table1_tr2">
        <div id="table1_tr2_td1" width="250"
         widthi="266">
            content 4
        </div>
        <br clear="both"/>
    </div>
</div>
<div id="table2" float="left">
    <div id="table2_tr1">
        <div id="table2_tr1_td1" width="300"
         widthi="266">
            content 2
        </div>
        <br clear="both"/>
    </div>
</div>
```

Figure 11: DIV Conversion Example
Style attributes such as WIDTH remain inline at this stage. The WIDTHI style attribute is an artifact of the layout stage of our process and will be removed later.

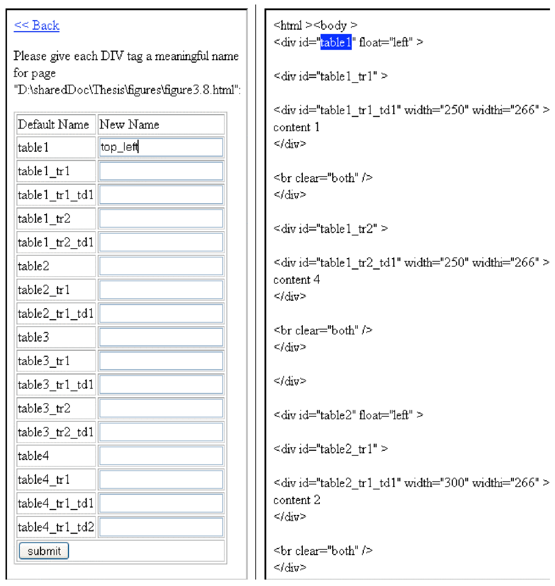
```
rule replaceTableByDIV
   replace [html_interesting_element]

      <tag 'id=TableIDParam [stringlit]>
         <table RptTableParams
               [repeat html_any_tag_parameter]>
            RptTableContents
               [repeat html_table_content]
         </table>
      </tag>

   construct TrtoDIV [repeat div_tag]
      _ [replaceTrByDIV each RptTableContents]

   by
      <div 'id=TableIDParam RptTableParams>
         TrtoDIV
      </div>
end rule
```

Figure 12: Main TXL Transformation Rule for DIV Conversion



<< Back

Please give each DIV tag a meaningful name for page "D:\sharedDoc\Thesis\figures\figure3.8.html":

Default Name	New Name
table1	top_left
table1_tr1	
table1_tr1_td1	
table1_tr2	
table1_tr2_td1	
table2	
table2_tr1	
table2_tr1_td1	
table3	
table3_tr1	
table3_tr1_td1	
table3_tr2	
table3_tr2_td1	
table4	
table4_tr1	
table4_tr1_td1	
table4_tr1_td2	

```
<html ><body >
<div id="table1" float="left" >

<div id="table1_tr1" >

<div id="table1_tr1_td1" width="250" widthi="266" >
content 1
</div>

<br clear="both" />
</div>

<div id="table1_tr2" >

<div id="table1_tr2_td1" width="250" widthi="266" >
content 4
</div>

<br clear="both" />
</div>

</div>

<div id="table2" float="left" >

<div id="table2_tr1" >

<div id="table2_tr1_td1" width="300" widthi="266" >
content 2
</div>

<br clear="both" />
</div>
```

Figure 13: Interface for Hand Renaming

works on the pages themselves to update the style references in DIV sections to refer to the new unified style names.

The first transformation, all of the CSS files for individual pages are concatenated into one merged file. A simple TXL pattern-matching rule searches the merged file for exact clones of each CSS style. Subsequent clones are marked with the name of the original style and a table of equivalences is output as a list to a clone table file (Figure 16). Once the clone table file has been output, the merged CSS file is optimized by removing all marked clones to yield a minimal CSS style file for the entire website.

```
<div id="top_left" float="left">
    <div id="top_left_container_1">
        <div id="top_left_content_1" width="250"
                widthi="266">
            content 1
        </div>
        <br clear="both"/>
    </div>
    <div id="top_left_container_2">
        <div id="top_left_content_2" width="250"
                widthi="266">
            content 4
        </div>
        <br clear="both"/>
    </div>
</div>
<div id="top_middle" float="left">
    <div id="top_middle_container">
        <div id="top_middle_content" width="300"
                widthi="266">
            content 2
        </div>
        <br clear="both"/>
    </div>
</div>
```

Figure 14: DIV Conversion Example After Renaming

The second transformation is then run on every page of the site, updating style references of each DIV according to the clone table. Each style reference is looked up in the clone table and changed to the name of the style of which it is a clone. The final result is a website with a single merged, optimized CSS file used by all pages of the site, as if the site had been hand crafted to the modern web standard.

2.7 Hand Tuning

The final step in the process is the hand tuning of the generated CSS styles to exactly match minor details of the original look and feel. Typically this involves adding a bit of extra margin space to the styles for some DIV blocks and removing an occasional redundant attribute. This step usually requires only a few minutes of web programmer time to complete.

3 Experience

Our method has been tested on a number of example websites with varying levels of table layout complexity ranging from simple layout

```
#top_left {
    float: left;
    margin: auto;
  }
#top_left_container_1 {
    float: left;
    margin: auto;
  }
#top_left_container_2 {
    float: left;
    margin: auto;
  }
#top_left_content_1 {
    float: left;
    margin: auto;
    width: 250;
    widthi: 266;
  }
#top_left_content_2 {
    float: left;
    margin: auto;
    width: 250;
    widthi: 266;
  }
#top_middle {
    float: left;
    margin: auto;
  }
#top_middle_container {
    float: left;
    margin: auto;
  }
#top_middle_content {
    float: left;
    margin: auto;
    width: 300;
    widthi: 266;
  }
```

Figure 15: Example Extracted CSS Style File
As style attributes are extracted to a CSS style file for each DIV converted page, they are removed from the DIVs in the page so that all style information appears only in the style file.

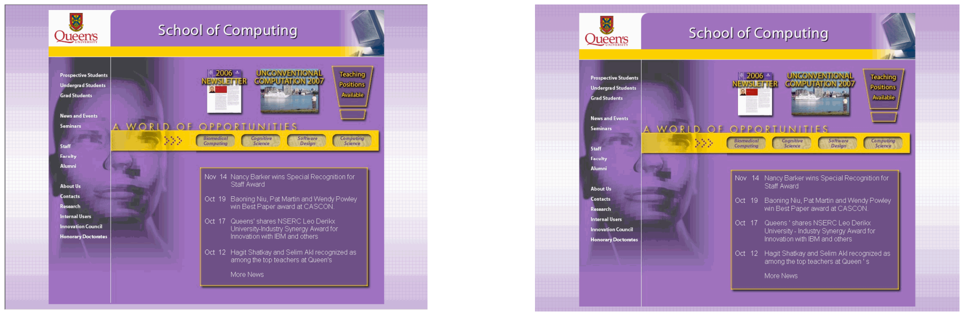
to complex ROWSPAN and COLSPAN structures in order to validate our table recognition algorithms and the ability of our method to preserve look and feel. In addition, two real entire table-based legacy websites, one with simple table layout and one with complex, one originally generated using Claris Home Page and one with MS Front Page, have been converted to test our clone detection and CSS generation methods. This section outlines our experiences with some of these examples, first with two simple layout sites and then two complex.

```
"top_left" -> "top_left_container_1"
"top_left" -> "top_left_container_2"
"top_left" -> "top_middle"
"top_left" -> "top_middle_container"
"top_left" -> "top_right"
"top_left" -> "top_right_container_1"
"top_left" -> "top_right_container_2"
"top_left" -> "a_top_left"
"top_left" -> "a_top_left_container_1"
"top_left" -> "a_top_left_container_2"
"top_left" -> "a_top_middle"
"top_left" -> "a_top_middle_container"
"top_left" -> "a_top_right"
"top_left" -> "a_top_right_container_1"
"top_left" -> "a_top_right_container_2"
"top_left" -> "b_top_left"
"top_left" -> "b_top_left_container_1"
"top_left" -> "b_top_left_container_2"
"top_left" -> "b_top_middle"
"top_left" -> "b_top_middle_container"
"top_left" -> "b_top_right"
"top_left" -> "b_top_right_container_1"
"top_left" -> "b_top_right_container_2"
"top_left_content_1" -> "top_left_content_2"
"top_left_content_1" -> "a_top_left_content_1"
"top_left_content_1" -> "a_top_left_content_2"
"top_left_content_1" -> "b_top_left_content_1"
"top_left_content_1" -> "b_top_left_content_2"
"top_middle_content" -> "a_top_middle_content"
"top_middle_content" -> "b_top_middle_content"
```

Figure 16: Partial Example Clone Table
As well as an integrated CSS style file, clone detection generates a clone equivalence table for use by the clone resolution transformation.

3.1 Queen's School of Computing Home Page

The home page of the School of Computing's website was authored and is maintained by hand in HTML using table layout, but with pre-existing CSS styles that must be retained in the result, making it an interesting different kind of challenge for our method. The layout is relatively simple, involving no ROWSPANs in the tables. Figure 17 shows the result of converting the front page of this site using our method, preserving existing style references (such as "class=wong") while introducing our own for the newly generated DIVs. New styles generated by our process are concatenated to the existing CSS style file for the site, yielding an identical look and feel (Figure 18).



Nov 14 Nancy Barker wins Special Recognition for Staff Award

Oct 19 Baoning Niu, Pat Martin and Wendy Powley win Best Paper award at CASCON.

Oct 17 Queens' shares NSERC Leo Derikx University-Industry Synergy Award for Innovation with IBM and others

Oct 12 Hagit Shatkay and Selim Akl recognized as among the top teachers at Queen's

More News

Figure 18: School of Computing Home Page Before and After Conversion

```
<div  id="topcontainer">
    <!--H1 -->
    <div  id="sep1">
        <div  id="topcontainercontent">
            . . .
        </div>
    </div>
    <div  id="sep1">
        <div  id="leftsep1"  class="wong">
            <!-- -->
        </div>
        <div  id="rightsep1">
            . . .
        </div>
        <br clear="both"/>
    </div>
    <div  id="sep1">
        <div  id="sep2bg">
            <!-- -->
        </div>
    </div>
</div>
```

Figure 17: School of Computing Home Page Following Conversion

```
#topnavitem1  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem2  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem3  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem4  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem5  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem6  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
#topnavitem7  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }

. . .
```

```
#topnavitem1  {
    float:  left;
    margin:  auto;
    widthi:  57;
  }
```

Figure 19: A Portion of the CSS Style File for the IEEE Kingston Website Before and After Clone Detection

3.2 IEEE Kingston Website

The IEEE Kingston Section website is small, consisting of only seven HTML pages and 2,781 lines of code. It is a simple layout site, with no ROWSPANs in its table structures, but uses a highly complex hierarchical set of nested table structures to layout its components. Due to this complexity, conversion of the site initially generated seven CSS files totally 5,598 lines of style specifications, posing a challenge for our clone detection and style minimization steps. Clone detection found 769 cloned styles that could be minimized and removed (see example Figure 19), reducing the final CSS style file for the site to only 323 lines. Due to the large number of similar generated styles, the renaming stage for this site used a significant amount of human interaction time. This points to a potential limitation of our method that may need to be addressed in future work.

3.3 James Cordy's Home Page

The home page of the second author's website uses one ROWSPAN in the table layout

```
<table cellspacing="0" cellpadding="0" width="100%">
    <tr>
        <td rowspan="4" width="190">
            . . .
        </td>
        <td nowrap>
            . . .
        </td>
        <td>
            . . .
        </td>
    </tr>
    <tr>
        <td>
            . . .
        </td>
        <td>
            . . .
        </td>
    </tr>
    <tr>
        <td colspan="2">
            . . .
        </td>
    </tr>
    <tr>
        <td colspan="2">
            . . .
        </td>
    </tr>
    . . .
</table>
```

Figure 20: Portion of Original Table Layout of James Cordy's Home Page

structures and controls layout using WIDTH specifications with both absolute and percent sizes as well as NOWRAP attributes, which are not available in CSS styles (Figure 20). This is typical of many legacy sites and makes a good example for our method. Following transformation, table partitioning has separated the ROWSPAN cells and maintained relative position using the FLOAT:LEFT style in the resulting CSS and DIV structure. Although there is no style corresponding to the NOWRAP attribute in the web standard, our transformation introduces a special NOBR DIV class, which is supported by the major browsers, to maintain the NOWRAP behaviour in the result (Figure 21).

3.4 TXL.ca Website

The TXL website was originally created using the early website authoring tool Claris Home

```
<div id="topleft">
    <div id="topleftbody">
        <div id="topleftcontent">
            . . .
        </div>
    </div>
</div>
<div id="topright">
    <div id="toprightbody1">
        <div id="toprightboycontent">
            <div class="nobr">
                . . .
            </div >
        </div>
        <div id="toprightboycontent">
            . . .
        </div>
    </div>
    <div id="toprightbody1">
        <div id="toprightboycontent">
            . . .
        </div>
        <div id="toprightboycontent">
            . . .
        </div>
    </div>
    <div id="toprightbody1">
        <div id="toprightbody3content">
            . . .
        </div>
    </div>
    <div id="toprightbody1">
        <div id="toprightbody3content">
            . . .
        </div>
    </div>
</div>
<br clear="both"/>
```

Figure 21: Corresponding Portion of Generated DIV Layout of James Cordy's Home Page

Page, which predates modern web standards and uses table layout and complex inline style attributes to achieve a pleasing result. Since the HTML code was originally automatically generated but is now hand maintained, this site is a prime candidate for our conversion. The site contains 38 HTML pages comprising approximately 4,996 lines along with two CGI scripts with dynamic page generation. Like many legacy retail sites, it includes pages with dynamic order forms and responses and makes a challenging realistic example.

The TXL site has a very complex table-based layout including multiple ROWSPANs at the same level. These pose a particular difficulty for table partitioning in that there is no unique solution. Our conversion resolves such ambiguities by choosing the ROWSPAN with the largest span to begin the partitioning, and then recursively repartitioning the next largest until all ROWSPANs have been resolved, giving a top-down implicit hierarchy.

The realistic size of this production website provided an interesting challenge for our clone detection and style minimization algorithms as well. The site has two different kinds of pages which use two quite different look-and-feel styles. Before the clone detection and removal process, the page-by-page conversion to DIV yielded 38 CSS files with 19,028 lines of style code. However, after consolidation, detection and removal of 2,850 style clones, the final sitewide CSS file was reduced to only 647 lines, a maintainable and reasonably sized stylesheet for such a complicated website (Figure 22).

The result of the conversion was a clearly maintainable web standards-compliant new website which is virtually indistinguishable from the original in look and feel (Figure 23) and has all the advantages of a hand crafted modern standard website in compatibility, maintainability and efficiency.

3.5 Other Examples

Several other websites have been used as examples for our process, including the IEEE ICCBSS conference website, which poses the new problem of multiple ROWSPANs at multiple simultaneous levels. At present our system cannot automatically partition such sites (which are relatively unusual), but it can still be applied, using a few minutes of hand partitioning for the pages with this problem. The conversion of this site yielded a home page which is literally indistinguishable from the original, while being completely migrated to the XHTML, DIV and CSS web standards.

3.6 Performance

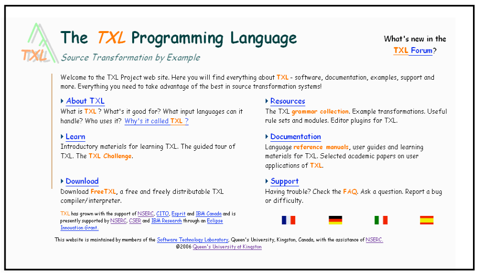
Our process is very fast, requiring for example only two or three minutes of transformation time for the entire TXL website conversion on a standard PC. The two human intervention steps, tag renaming and final style tuning, require a skilled web programmer to be done efficiently. Even so, each of these steps took only ten to 15 minutes of programmer time in the TXL website conversion, for a total of less than 30 minutes elapsed time to convert the 38 page table-based legacy site to a maintainable modern web standards compliant result.

```
#topcontainer {
    padding: 0;
    height: auto;
    margin: auto;
    width: 798px;
    text-align: center;
  }
#sep1 {
    float: left;
    margin: auto;
  }
#topcontainercontent {
    text-align: left;
    width: 798px;
    float: left;
    margin: auto;
  }
#leftsep1 {
    text-align: left;
    float: left;
    height: 24px;
    margin: auto;
    width: 675px;
  }
#rightsep1 {
    text-align: right;
    background-color: #ffcc00;
    float: left;
    height: 24px;
    margin: auto;
    vertical-align: top;
    width: 123px;
  }
#sep2bg {
    text-align: left;
    background-color: #ffffff;
    float: left;
    height: 1px;
    margin: auto;
    width: 798px;
  }
```

Figure 22: Part of Generated CSS Stylesheet for the TXL.ca Website

4 Related Work

Some commercial tools offer a rudimentary form of automated web standards conversion. For example, Adobe Dreamweaver [1] has a conversion that can automatically convert


The TXL Programming Language
Source Transformation by Example

What's new in the TXL Forum?

Welcome to the TXL Project web site. Here you will find everything about TXL - software, documentation, examples, support and more. Everything you need to take advantage of the best in source transformation systems!

About TXL
What is TXL ? What's it good for? What input languages can it handle? Who uses it? Why's it called TXL ?

Resources
The TXL grammar collection. Example transformations. Useful rule sets and modules. Editor plugins for TXL.

Learn
Introductory materials for learning TXL. The guided tour of TXL. The TXL Challenge.

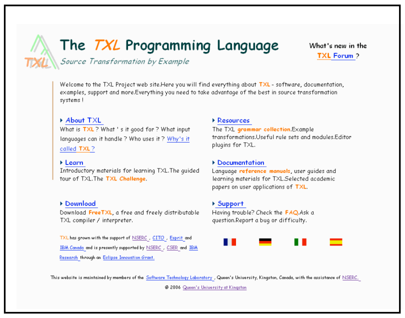
Documentation
Language reference manuals, user guides and learning materials for TXL. Selected academic papers on user applications of TXL.

Download
Download FreeTXL, a free and freely distributable TXL compiler/interpreter.

Support
Having trouble? Check the FAQ. Ask a question. Report a bug or difficulty.

TXL has grown with the support of NSERC, CITO, Esprit and IBM Canada and is presently supported by NSERC, CSER and IBM Research through an Eclipse Innovation Grant.

This website is maintained by members of the Software Technology Laboratory, Queen's University, Kingston, Canada, with the assistance of NSERC.
©2006 Queen's University at Kingston


The TXL Programming Language
Source Transformation by Example

What's new in the TXL Forum ?

Welcome to the TXL Project web site.Here you will find everything about TXL - software, documentation, examples, support and more.Everything you need to take advantage of the best in source transformation systems !

About TXL
What is TXL ? What ' s it good for ? What input languages can it handle ? Who uses it ? Why's it called TXL ?

Resources
The TXL grammar collection.Example transformations.Useful rule sets and modules.Editor plugins for TXL.

Learn
Introductory materials for learning TXL.The guided tour of TXL.The TXL Challenge.

Documentation
Language reference manuals, user guides and learning materials for TXL.Selected academic papers on user applications of TXL.

Download
Download FreeTXL, a free and freely distributable TXL compiler / interpreter.

Support
Having trouble? Check the FAQ.Ask a question.Report a bug or difficulty.

TXL has grown with the support of NSERC , CITO , Esprit and IBM Canada and is presently supported by NSERC , CSER and IBM Research through an Eclipse Innovation Grant.

This website is maintained by members of the Software Technology Laboratory , Queen's University, Kingston, Canada, with the assistance of NSERC.
© 2006 Queen's University at Kingston

Figure 23: TXL.ca Website Main Page Before and After Conversion
(Rendered in different browser window sizes.)

pages from table layout to DIV-based layout. However, these tools use only local CSS styles on a per page basis and do not recognize commonality, with the result that the converted page is no more maintainable than the original. In the case of Dreamweaver, the situation is even worse since in order to maintain an identical look the tool resorts to converting to absolute positions and sizes for all page elements, losing all structure and making it impossible to recognize style clones even by hand.

Many other researchers have worked in the broader area of web technology migrations, including several that have exploited the same source transformation engine, TXL, that underlies our work. Hassan et al. [10] have used TXL transformations to migrate web applications from ASP to the NSP web framework while preserving the original code structure and commenting. Xu et al. [20] have used TXL to modernize embedded Java code in JSP pages to custom-tag based JSP applications.

In other work, Jiang et al. [13] have used pattern matching techniques to automatically analyze server generated pages to aid in migrating web applications to web services. Ping et al. [15] have described an approach to migrating web applications from IBM Net.Data to JSP while separating database aspects from presentation logic. Ricca et al. [16] have used clustering techniques to identify static pages that can be transformed to dynamic pages.

Our method for clone detection is based on our previous work with Synytskyy [7] which addressed the problem of identifying near-miss clones in static and dynamic web pages. Our method adopts only exact clone detection from that work, but could be extended to parameterized styles using near-miss clone detection. Many other clone detection techniques could be used as well, for example Baxter's method based on abstract syntax trees [3] which also handles near-miss clones.

Our simple table partitioning strategy is only roughly based on the wide range of literature in table recognition [22], and relates most closely to methods such as Handley's [9] which address the higher-level logical structure of tables. Full table recognition is a much more ambitious task which includes table detection, functional and structural analysis, and finally table interpretation [12]. One piece of work that relates well to ours because it applies similarity-based reasoning to detect cells in tables is that of Chen et al. [4], which has been used to detect and analyze tables in airline information web pages. Yoshida et al. [21] have described a method to integrate HTML tables based on the category of object in the table. However, in these cases the focus has been on the contents of true tables rather than on table-based layout.

Finally, we have used the TXL transformation language [6] in implementing our source transformations. Any other modern source transformation system, for example ASF+SDF [18] or Stratego [19] could serve as well. Advantages of TXL over XSLT [5] in our application include its ability to handle malformed input using robust parsing [2], its ability to express complex nonstructural patterns [8], and previous experience in applying it to table recognition and clone detection tasks [23, 7].

5 Summary & Future Work

We have presented an automated process for migrating legacy websites using table-based layout to modern, maintainable XHTML, DIV and CSS web standards-based websites with all the advantages of a hand-crafted modern replacement. Our process uses table recognition and software clone detection technology with multi-pass source transformation to yield a high quality result that preserves hierarchical and layout structure of web pages while synthesizing a minimized common stylesheet for the entire site. The look and feel of converted websites is virtually identical to the original.

Our method has a number of limitations that bear further work. As noted in Section 3.2, websites with large numbers of small similar elements can make the renaming task an issue. This could be addressed with better name inference for generated DIVs, or better clues to the programmer indicating which styles are likely to be eliminated as clones. Multiple ROWSPANs at multiple simultaneous levels pose problems that must be assisted by hand. While this limitation can largely be overcome with better table analysis, to some extent it derives from the limited expressiveness of the DIV feature itself, and may always need some hand tuning. At present our system does not handle web applications written in server side languages such as ASP and JSP. This can potentially be addressed by adopting the multilingual parsing methods of Synytskyy et al. [17] to separate page factors using robust parsing. And of course the method needs to be validated on a larger number of legacy websites (of which there is no lack of examples).

Acknowledgements

This work is supported by the Natural Sciences and Engineering Research Council of Canada.

About the Authors

Andy Mao is an IT specialist in the global business services division of the IBM Pacific development center in Vancouver, BC. Prior to joining IBM, Andy was a web developer at Blast Radius Inc. from 2003-05. He holds a B.A. in Information Technology from York University and recently completed his Master's in Computing at Queen's University under the supervision of James Cordy and Thomas Dean. His research interests include source transformation, software re-engineering and e-commerce web applications.

James Cordy is a Professor and recent former Director of the School of Computing at Queen's University. From 1995 to 2000 he was Vice President and Chief Research Scientist at Legasys Corporation, a software technology company specializing in legacy software system analysis and renovation. Dr. Cordy is a founding member of the Software Technology Laboratory at Queen's and winner of the 1994 ITRC Innovation Excellence award and the 1995 ITRC Chair's Award for Entrepreneurship in Technology Innovation. He is a member of the ACM, a senior member of the IEEE and an IBM CAS Faculty Fellow.

Thomas Dean is an Associate Professor in the Department of Electrical and Computer Engineering at Queen's University and an Adjunct Associate Professor at the Royal Military College of Kingston. His background includes research in air traffic control systems, language formalization and five and a half years as a Sr. Research Scientist at Legasys Corporation where he worked on advanced software transformation and evolution techniques in an industrial setting. His current research interests are software transformation, web site evolution and the security of network applications.

References

[1] Adobe. Dreamweaver CS3. http://www.adobe.com/products/dreamweaver/.

[2] D.T. Barnard and R.C. Holt. Hierarchic Syntax Error Repair for LR Grammars. *Int. J. Computing and Information Sciences*, 11(4):231–258, 1982.

[3] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone Detection using Abstract Syntax Trees. In *IEEE 14th Int. Conf. on Software Maintenance*, pages 368–377, 1998.

[4] H. Chen, S. Tsai, and J. Tsai. Mining Tables from Large Scale HTML Texts. In *18th Conf. on Computational Linguistics*, pages 166–172, 2000.

[5] J. Clark. XSL Transformations (XSLT) v1.0. http://www.w3.org/TR/xslt, 1999.

[6] J.R. Cordy. The TXL Source Transformation Language. *Science of Computer Programming*, 61(3):190–210, 2006.

[7] J.R. Cordy, T.R. Dean, and N. Synytskyy. Practical Language-independent Detection of Near-miss Clones. In *CASCON '04: 2004 Conf. of the Center for Advanced Studies on Collaborative Research*, pages 1–12, 2004.

[8] T.R. Dean, J.R. Cordy, A.J. Malton, and K.A. Schneider. Agile parsing in TXL. *J. Automated Software Engineering*, 10(4):311–336, 2003.

[9] J. Handley. Table Analysis for Multi-line Cell Identification. In *Document Recognition and Retrieval VIII*, volume 4307, pages 34–43, 2001.

[10] A.E. Hassan and R.C. Holt. Migrating Web frameworks using Water Transformations. In *IEEE 27th Int. Conf. on Computer Software and Applications*, pages 296–303, 2003.

[11] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. Table Structure Recognition and its Evaluation. In *Doc. Recog. and Retrieval VIII*, volume 4307, pages 44–55, 2001.

[12] M. Hurst. Layout and Language: Challenges for Table Understanding on the Web. In *1st Int. Workshop on Web Document Analysis*, pages 27–30, 2001.

[13] Y. Jiang and E. Stroulia. Towards Reengineering Web Sites to Web-services Providers. In *8th European Conf. on Software Maintenance and Reengineering*, pages 296–308, 2004.

[14] R. Koschke. A Survey of Research on Software Clones. In *Duplication, Redundancy, and Similarity in Software*, number 06301 in Dagstuhl Seminar Proceedings, 2007.

[15] Y. Ping, J. Lu, T.C. Lau, K. Kontogiannis, T. Tong, and B. Yi. Migration of Legacy Web Applications to Enterprise Java Environments net.data to JSP Transformation. In *CASCON '03: 2003 Conf. of the Center for Advanced Studies on Collaborative Research*, pages 223–237, 2003.

[16] F. Ricca and P. Tonella. Using Clustering to Support the Migration from Static to Dynamic Web Pages. In *11th IEEE Int. Workshop on Program Comprehension*, pages 207–216, 2003.

[17] N. Synytskyy, J.R. Cordy, and T.R. Dean. Robust Multilingual Parsing using Island Grammars. In *CASCON '03: 2003 Conf. of the Center for Advanced Studies on Collaborative Research*, pages 266–278, 2003.

[18] M. G. J. van den Brand, J. Heering, P. Klint, and P. A. Olivier. Compiling Language Definitions: the ASF+SDF Compiler. *ACM Trans. on Prog. Languages and Systems*, 24(4):334–368, 2002.

[19] E. Visser. Program Transformation with Stratego/XT: Rules, Strategies, Tools, and Systems in Stratego/XT 0.9. In *Domain-Specific Program Generation*, volume 3016 of *LNCS*, pages 216–238, 2004.

[20] S. Xu and T.R. Dean. Transforming Embedded Java to Custom Tags. In *IEEE 5th Int. Workshop on Source Code Analysis and Manipulation*, pages 173–182, 2005.

[21] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web. In *1st int. Workshop on Web Document Analysis*, pages 31–34, 2001.

[22] R. Zanibbi, D. Blostein, and J.R. Cordy. A Survey of Table Recognition: Models: Observations, Transformations, and Inferences. *Int. J. Document Analysis and Recognition*, 7(1):1–16, 2004.

[23] R. Zanibbi, D. Blostein, and J.R. Cordy. The Recognition Strategy Language. In *ICDAR '05: 8th Int. Conf. on Document Analysis and Recognition*, pages 565–569 Vol. 2, 2005.