

# 6.871 Expert System: WDS - Web Design Assistant System

Timur Tokmouline

May 11, 2005

## 1 Introduction

Today, despite the emergence of WYSIWYG software, web design is a difficult and a necessary component of our lives, and the hardest part of web design is, arguably, getting started.<sup>1</sup> The web has become an integral part of our daily lives: we do research, communicate with friends, read news - all online. As a result, publishing material on the web has become a necessity. In general web design of usable web pages is hard since it requires laying out the material in such an order that a visitor would be able to see the most important portions right away. There are, in general, no answers as to what a good layout is. There are many qualitative suggestions. However, translating those qualitative suggestions into concrete actions on the portions of the website, to the best of my knowledge, has not yet been accomplished.

The current array of WYSIWYG tools available to the consumers to ease the pain of web development is quite primitive since the consumer must tell the WYSIWYG editor exactly what to do. As such, the WYSIWYG tools, like Microsoft Frontpage, do not let the user get any closer to starting on a good web page design. The real hard task, figuring out a suggestion for a decent starting point, still remains a hard and unsolved problem.

In this paper, we present a Web Design Assistant System, or WDS, that employs the Generate-Test-Debug approach [1] to randomly create and then iteratively improve web page layout of user-supplied content. WDS aims to facilitate web page design process by producing a reasonable starting point for web design, given content description from the user of the system. WDS evaluates the quality of a layout by simulating a visitor inspection on the layout. The results are reported to the debugger, which uses knowledge about web layout improvement to create a new layout, where more important items are more readily accessible to a typical user. From inspection of program output thus far, it seems that WDS

---

<sup>1</sup>WYSIWYG is an abbreviation for "what you see is what you get" style editors, such as Microsoft Word or Microsoft Frontpage.

does in fact generate a reasonable starting point for web design in that the resulting web page layout is not optimal but does place important items in places likely to be noticed by typical visitor.

## 2 The Task: Generating a Starting Point for Web Design

The task at hand is to generate a reasonable starting point for web design of a single web page. The user should be able to take the program output and improve that starting point into a usable web site. The program's output is not expected to be a perfect, production-ready webpage. Instead, the program output should make the most important features accessible to a visitor, who has a limited time to invest in detailed website reading.

We make several important assumptions about the user of our program:

- The user knows what the content is. The user knows what material will eventually placed on the web in form of a web site.
- The user knows how to decompose the content into a sequence of paragraphs and images.
- The user knows which portions of the content are more important than others. The user is not expected to know this exactly. In fact, we ask the user to take the best guess on ranking the different portions of the content.

## 3 An Example: Layout for a simple scientific abstract

In this section, we give a detailed description of inputs accepted and corresponding outputs produced by the system. We do so by discussing an example that involves layout of a simple scientific abstract.

### 3.1 Getting Started

The assumed starting point is that the user has figured out the content to be placed on the web site and has decomposed this content into paragraphs of text and properly sized images. An example of the starting point is shown in Figure 1.

### 3.2 User Input: Describing the content

As shown in Figure 2, the user is asked to describe the items that make up the content one by one. We assume, for simplicity, that the web page has at most a total of 16 content items

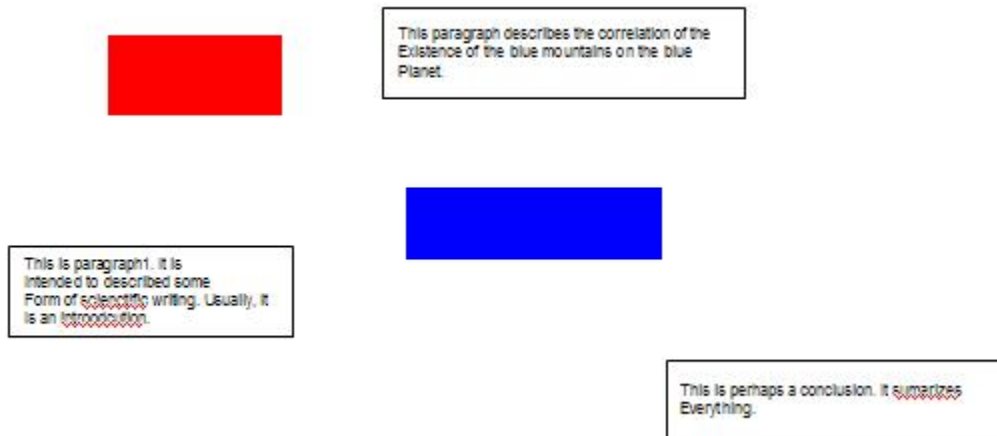


Figure 1: Step 1: The user has figured out the content and has decomposed the content into paragraphs and images.

on it. For each particular item, the user enters the type of item, the item content, and the rank of this particular item. The type of content is either text or image. The item content depends significantly on the item type.

- *For a paragraph of text*, the content is just the paragraph entered on a single line, as shown in Figure 2.
- *For an image*, the content is a description of the dimensions followed by the picture source, as shown in Figure 3. In describing image data, we enter first the width in pixels, then height in pixels, and then the file source of the image.

As demonstrated in Figure 2, the user then enters the rank for each image and text items. The rank is assumed to be an integer between 1 and 16. We do not expect the rank to perfectly reflect the ordering of items in importance - the rank is a subjective metric.<sup>2</sup>

When the user has completed entering the information about the items, the user enters -1 for item type. At that point in time, the content description step is complete.

### 3.3 User Input: Characterizing a Typical Web Page Visitor

It is hard to predict the audience of the web page, since the web design assistant program has no knowledge in the domain of the content. As shown in Figure 4, the user is therefore

<sup>2</sup>We deal with the subjectivity of the metric by avoiding optimizing solutions. Rather we aim to perform reasonably in generating a starting point for web page design.

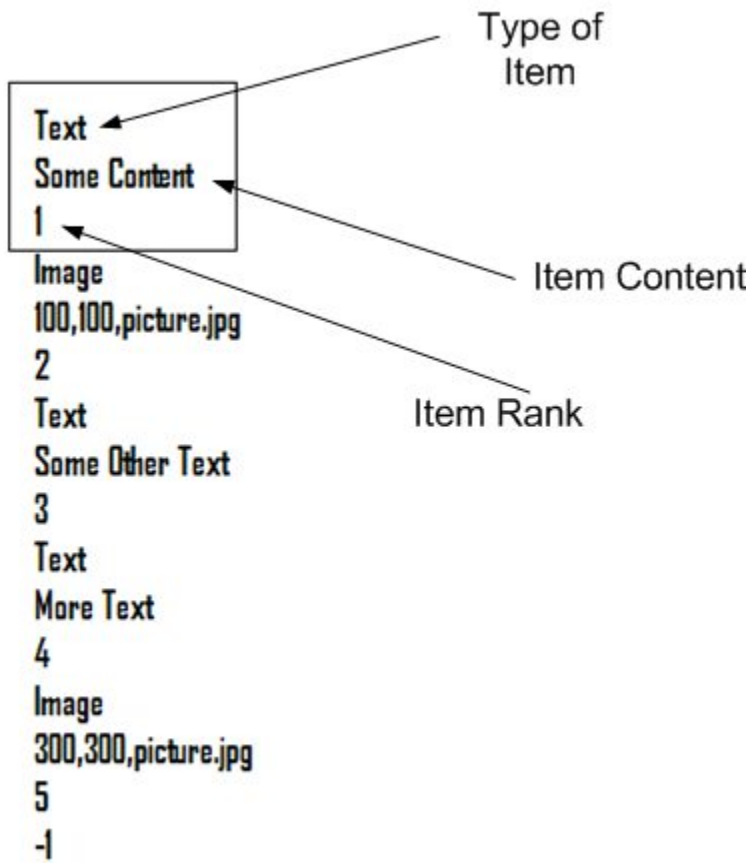


Figure 2: Describing a Text Item: A single paragraph of text

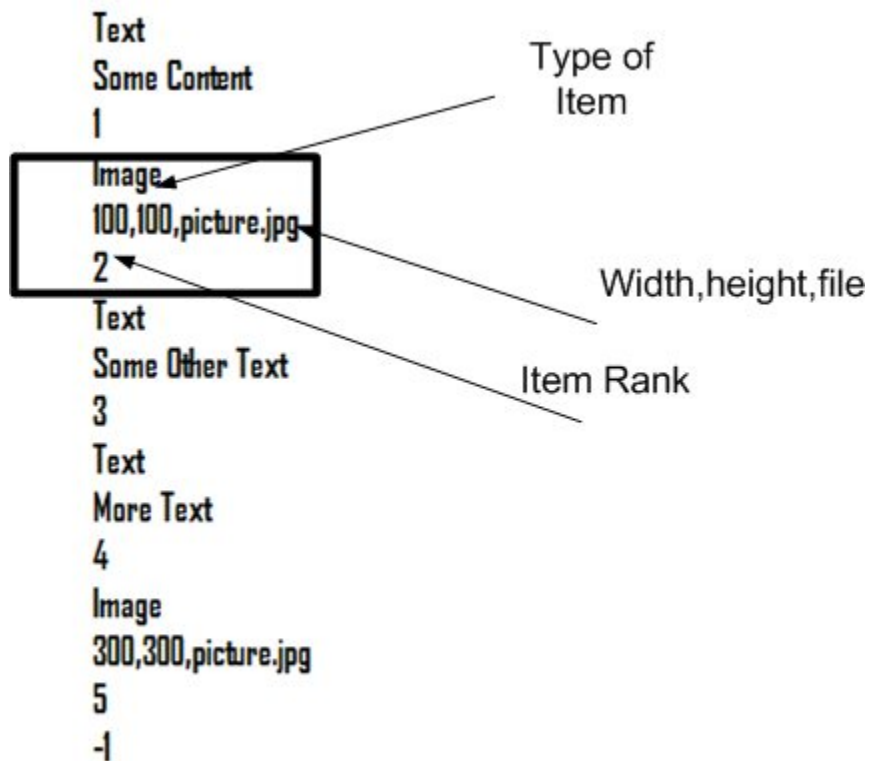


Figure 3: Describing an Image Item: An image described by width, height, and content source file.

```
viewer.totalVisitorTime=20
```

Figure 4: Entering the Amount of Time The expected visitor will spend on the web page.

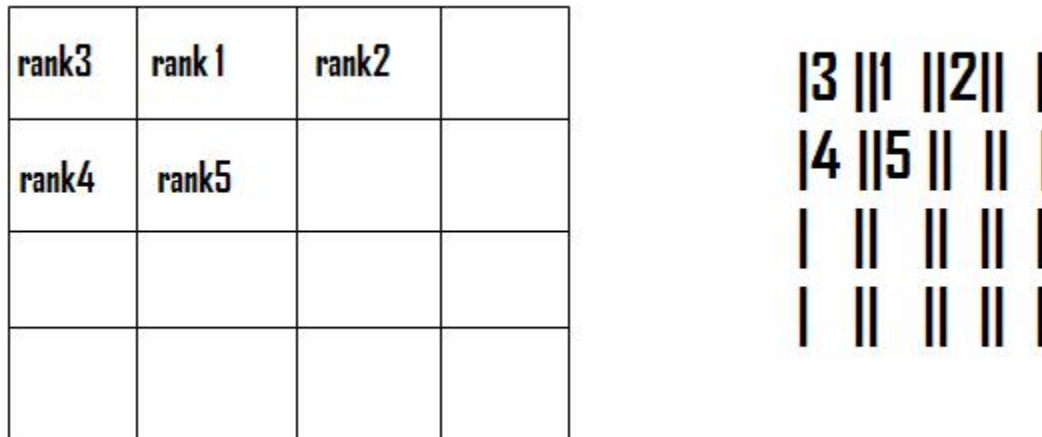


Figure 5: Correspondence between the 4x4 grid used by the program and the actual web page layout.

asked to input the expected approximate amount of time (in seconds) that a typical visitor would spend on the web page.

### 3.4 The program output - In Detail

During the processing stage, the program randomly assigns the individual content items into a 4 by 4 grid, as shown in Figure 5. The cells of the grid do not overlap. The idea then is to move the items around in the grid until a desirable output is achieved. The program simulates a real user looking at the web page. The feedback from this inspection process is then used to improve the version of the web site. The program then selects a potentially different user and simulates this user looking at the improved web page.

The process continues in this manner, iteratively, until it completes the number of cycles specified by the developer. For the sample run the number of cycles is 4. The output of the program at each iteration is shown in Figure 6. At each cycle, the program prints the type of the selected simulated user selected as well as the items this user has had time to inspect in the order the user had inspected the items. Based on this feedback from the simulated user, the web page is improved by moving some of the items into different places. The degree to which the web page has been changed between the iterations is noted by the Net Manhattan Distance. We will discuss this metric later, but its purpose is to give a cumulative impression of the degree to which the web page layout has changed. High Net Manhattan Distance value corresponds to a large change in the web page layout.

It is worth noting here that the user selected during each iteration may be of different type. That is, they may not necessarily look at the items in the same order. Some users may spend more time looking at text items than on images. Some users may differentiate

```

-----
Welcome to Web Design Assistant!
written by Timur Tokmouline
for 6.871 Final Project
-----

```

```

The initial web page layout is:

```

```

|  ||  ||  ||  |
|  ||2 ||3 ||4 |
|  ||5 ||1 ||  |
|  ||  ||  ||  |

```

```

-----
-----
Visitor Report for visitor #0
Selected viewing pattern 2
Item 2 took 15000.0
The selected viewer has seen 1
items

```

```

_____new web page layout
is:_____

```

```

|2 ||3 ||4 ||  |
|5 ||1 ||  ||  |
|  ||  ||  ||  |
|  ||  ||  ||  |

```

```

Net manhattan distance from
previous layout = 10

```

```

-----
-----
...

```

Figure 6: The Program Output: Successive iterations of simulated user inspection are followed by "debugging" improvements to the web layout in form of item movements inside the grid. Net Manhattan Distance indicates the degree to which the web page layout has changed between subsequent iterations.

between small images, large images, and medium images. Improvement based on feedback of one user does not guarantee an improved performance by users chosen in later iterations.

### 3.5 The program output - analysis

The final output of the program is shown in Figure 7 and its translation into the pictorial representation containing the paragraphs and images is noted in Figure 5. We note that this output is not a candidate for publishing. However, it does communicate visually the suggested positions of the important features to the user, and is therefore a reasonable starting point for further improvement by the user.

## 4 Approach: Generate Test Debug

The approach selected for tackling the specific problem of the iterative improvement described in the previous section is the Generate-Test-Debug (GDT) paradigm pioneered by Reid and Davis [1]. The essence of the approach is pictured in Figure 8. In short, the main idea of Generate-Test-Debug is to create a solution, to test the quality of the solution (possibly by simulation), and to use the feedback from the testing stage to improve on the original solution, and then to test the new and improved solution. Generate-Test-Debug makes sense in this domain, since web design is, by nature, iterative [3] in that a web page usually starts out as a simple draft and is then improved after interactions with the user.

The first stage, generate, is taken to be random allocation of all items (images and text paragraphs alike) into individual cells of a 4x4 grid. It is assumed that after enough iterations, the layout will approach a reasonable endpoint.

The second stage, test, involves running a simulated user on a web page layout. This simulated user is run to determine the sequence of images and text paragraphs that were seen by the user. The output from this Test stage is then a listing of the items seen and the amount of time spent looking at each item. We note that before each iteration, we choose a different user type from a pool of  $n$  different user types with probability  $1/n$ .

The third stage, debug, serves its purpose by modifying the web page layout according to feedback from the simulated user in hope of improving the web page layout. In this stage, the items are shifted around the grid. The specifics of this process pertain to the knowledge and will be discussed in the next section.

Finally, to evaluate the evolution of the web layout, we use a Net Manhattan Metric (NMM). The Net Manhattan Distances metric is a sum of all Manhattan Distances [5] between the positions of items on subsequent layout grids. That is, to compute NMM, we compare two 4x4 grids representing the web page layouts of two subsequent iterations, computing the manhattan distance between the position of each item on one board to the



```

-----
Welcome to Web Design Assistant!
written by Timur Tokmouline
for 6.871 Final Project
-----

```

```

The initial web page layout is:

```

```

|  |  |  |  | |
|  |2 |3 |4 |
|  |5 ||1 |  |
|  |  |  |  |

```

```

-----
-----
Visitor Report for visitor #0
Selected viewing pattern 2
Item 2 took 15000.0
The selected viewer has seen 1
items

```

```

_____new web page layout
is:

```

```

|2 ||3 ||4 ||  |
|5 ||1 ||  ||  |
|  ||  ||  ||  |
|  ||  ||  ||  |

```

```

Net manhattan distance from
previous layout = 10

```

```

.....

```

```

Net manhattan distance from
previous layout = 6

```

```

_____The Final Layout
is _____:

```

```

|3 ||1 ||2 ||  |
|4 ||5 ||  ||  |
|  ||  ||  ||  |
|  ||  ||  ||  |

```

Figure 7: Visual representation of the final layout. The program prints out the grid representation in its textual interface. The correspondence is drawn to an actual pictured layout for a qualitative evaluation of the result.

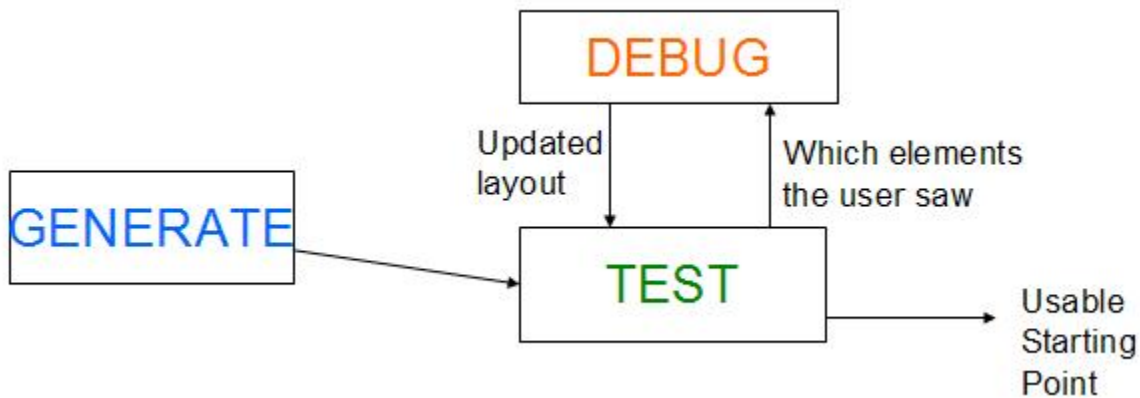


Figure 8: Generate-Test-Debug approach mandates that the tester not only say if the solution works, but be more precise so that the feedback may be reused to improve the original solution.

position of that same item on another board. We then sum the manhattan distances over all the items.<sup>3</sup>

The main idea behind the Net Manhattan Distance is that it is larger when comparing two layouts that are more different. Consequently, if the debug stage changed the layout significantly, NMM will be quite high. The main idea of comparing NMM over the course of multiple iterations is to see how well a debugging scheme adapts the web page layout - there should not be significant changes made by the debugger on consistent basis. In other words, we don't want to be in need of significant changes to the layout to accomodate different users. *A debugging scheme that consistently has a high NMM does not serve the system well.*

4

## 5 Knowledge

Following the structure of the Generate-Test-Debug (GDT) approach described in the previous section, the knowledge in the application is split into 3 portions. The Generate stage is executed only initially and its only purpose is to randomly allocate Image and Text items into the 4 by 4 grid - we assume that it carries no knowledge.<sup>5</sup> The Test stage contains

<sup>3</sup>The assumption here is that none of the items were lost or gained during the debugging phase between the two iterations.

<sup>4</sup>We'll discuss the application of the Net Manhattan Distances in more detail in the Evaluation Section, where we justify our choice of the different parameters.

<sup>5</sup>We note that making this stage more intelligent may lead to better results. But we leave this to future work.

Figure 9: According to [poynterextra.org](http://poynterextra.org), first-time visitors to a news site tend to look first in the upper left corner. They then look in the ring pattern surrounding the upper left corner. Finally, they look in the remaining outside ring region. Please see their website for the figure.

the knowledge about how a user looks at a web page. Finally, the Debug stage contains knowledge of how to improve the web page layout between successive evaluations by the simulated users. The rest of the assumptions are (in one way or another) encoded into the framework - the infrastructure that bridges the user input and the three stages of the GDT.

## 5.1 Knowledge in Framework

The bulk of knowledge contained in the framework concerns assumptions about how different portions of a web page come together to form one meaningful web page. Namely, the framework encapsulates the following assumptions:

- Maximum of 16 items are allowed on the website. This is not an unrealistic assumption - if a visitor inspects items one by 1, it seems nonsensical that the typical visitor would go through the entire web page. Part of the reason for this is the availability of other web sites with similar content just a click away.
- The visitor does not scroll down to see the contents at the bottom of the page. According to one usability study [2], this assumption turns out to be a valid as well.
- Manipulation of items within a 4x4 grid can lead to a reasonable starting place. This assumption draws strength from the fact that we wish to communicate relative positions of the items in a setting where a new visitor is able to go through a reasonable amount of the more important items (as ranked by the user). The 4x4 grid may be a coarse layout infrastructure, but it serves its purpose as long as the user is able to see the places that will most likely attract a new visitor.

## 5.2 Knowledge in Tester

The knowledge enclosed in the Tester stage mainly consists of concepts needed to simulate user viewing a web page. The simulated viewer can be defined by the following three concepts:

- *an ordering of the 16 squares* in which the simulated web page visitor looks at the individual areas of the 4x4 grid that represents web page layout. The assumption is that the viewer never looks in two grid-squares at once - this seems to be the valid

assertion since the eyes usually concentrate on a single point. One study done by poynterextra.com indicates that the viewing pattern of first time news article viewers is distributed according to the pattern shown in Figure 9 [6].

- To simulate the reality that the web page will be visited by different people, we select a different type of a visitor every iteration. In more detail, we select a different ordering of the 16 squares from a pool of  $n$  different ordering types with probability  $1/n$ . Making each ordering equally likely is sensible since we don't have apriori information about the type of user that will be visiting the web site (except for the ring-shaped pattern with priority increasing toward the upper-left corner).
- *an upper bound on the amount of time the simulated visitor spends on the site.* This parameter is defined by the user - the system leverages the user's expectation about the type of potential visitors.
- *a viewer function* that maps individual items (images and text paragraphs) to the individual amounts of time it takes for the user to inspect these items.

### 5.3 Knowledge in Debugger

The debugger contains the knowledge about improving the web page layout between the subsequent simulated user inspections of the web page. In particular, the debugger relies heavily on the *update function* that maps the previous location and the feedback from the simulated viewer to a new location of the item within the grid.

## 6 What Works

Given the assumptions listed in the previous section, I designed a Web Design Assistant System (WDS). Given my experimentation with WDS thus far, WDS performs well in its designated domain:

- From what I have seen thus far, the system produces an arrangement of items within a grid that highlights the more important items by placing them into a region where expected visitor is expected to see them first. *This result is sensible since this was the original goal of the system.*
- As a consequence, the result produced by WDS is not perfect but is a decent starting place for further design. Namely, the output from WDS highlights the places the more important items need to be placed in.

## 7 What Doesn't Work

Just like any expert system, WDS has the following drawbacks:

- Just because the user has seen an item doesn't imply that the user understood the content of the item - reading a text doesn't imply understanding the text. As usability ultimately aims to relay information to the visitor, we cannot say that results of WDS are necessarily usable. The results highlight locations where the simulated user is highly likely to see the important items - WDS doesn't guarantee that the actual user would understand the items.
- Building on the previous point, WDS has no knowledge of the domain described by the website. Namely, WDS cannot parse and understand the content of the individual web items. As such, WDS cannot assist in breaking down the text into more understandable pieces.
- WDS describes website layout coarsely using a 4x4 grid. WDS has no knowledge of how to resize components, such as images. As figuring out the appropriate component sizes is usually a significant part of web design, WDS falls quite short in that area.

## 8 Current System Parameters

Currently, the system is using a *One - Square - Improvement* update function, shown in Figure 12 and a *ImgDetail* viewing function shown in Figure 11. Finally, the system uses the set of simulated user grid sequences shown in Figure 10. This set consists of 10 sequences that conform to the pattern exposed by poyterextra.org [6] and 1 that does not.

## 9 Lessons Learned

Building systems that intend to help users with designing usable web pages is hard. The usability knowledge is available in qualitative form (for example at usability.gov [2]), but it is not entirely clear how to quantify it. As a consequence, figuring how to use the standard knowledge representation techniques to represent the usability knowledge is hard. In some sense, knowledge of how users look at a webpage is not expert knowledge - rather, it relates to the very basic questions of how the human visual system works and the issues with generalizing this knowledge to take into account a typical user. When coding the application and writing this very paper, I started to realize that my research wasn't just drawing on forms of usability knowledge but on different machine learning techniques (for example, how to avoid overfitting the webpage to a particular user type during the debugging stage). It

seems that it becomes increasingly hard to separate knowledge from the inference engine (the framework) when one starts dealing with the workings of the visual system.

## 10 Conclusion

We presented WDS, a web design assistant system whose purpose is to make web page design easier by generating a starting point for the user, given simple descriptions of the content. WDS aims to accomplish its goal using the Generate-Test-Debug [1], which integrates knowledge about how the user inspects content on the web page, knowledge about how to improve a webpage, and an assumption that web page may be decomposed into a sequence of images and text paragraphs.

## References

- [1] R. Davis, Generate Test Debug.
- [2] usability.gov <http://usability.gov/guidelines/layout.html#six>
- [3] Gould, Boies, and Lewis. 1991.
- [4] [http://9rules.com/whitespace/design/the\\_hardest\\_part.php](http://9rules.com/whitespace/design/the_hardest_part.php)
- [5] 6.034 Recitation 4 Notes.
- [6] <http://www.poynterextra.org/eyetrack2004/main.htm>

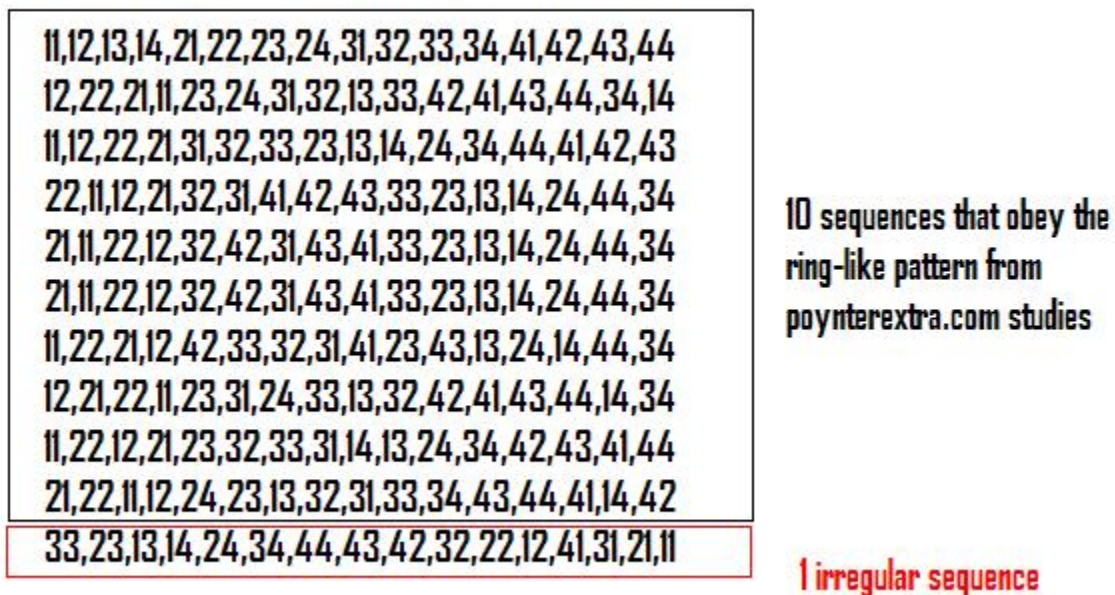


Figure 10: Currently Used Sequences: Each sequence represents a different types of user. The top 10 sequences represent users who conform to the ring-like website inspection pattern, progressing from top-left corner to the lower right corner. The last sequence doesn't obey the same pattern.

```
public double computeViewingTime(WebItem webItem) {
    if(!webItem.isText()) {
        ImgRcrd imgInfo = parseImg( webItem.getContent());
        // image classification
        // medium >=(210x230 pixels), and
        // large >=(365x240 pixels).
        // small otherwise
        // for times use the 1:3:5 ratio observed on poynterextra.org
        if( imgInfo.width>=365 && imgInfo.height>=240)
            // large img
            return 3000;
        else if( imgInfo.width>=210 && imgInfo.height>=230)
            // medium img
            return 9000;
        else
            // small img
            return 15000;
        } else {
            return 5000;
        }
    }
}
```

Figure 11: Currently Employed Viewer Function: at this point, the viewer functions exist only as java functions. Yet they still encapsulate knowledge of how long it takes to view each element. This viewer function differentiates amongst the different types of images, with larger images taking longer to view.



```

public void updateLayout(WebGrid webGrid, Point prevPos, WebItem wItem,
    List<ViewerResult> vrList) {
    if( wItem.getRank()-1 < vrList.size()) {
        // viewer got to see the ith square
        ViewerResult itemResult = vrList.get(wItem.getRank()-1);
        // the desirable position
        Point goalPos = itemResult.pos;
        // compute the vector in which to move
        double x = goalPos.x - prevPos.x;
        double y = goalPos.y - prevPos.y;
        double magnitude = Math.sqrt(Math.pow(x,2) + Math.pow(y,2));
        // normalize the movement vector
        double unitX = x/magnitude;
        double unitY = y/magnitude;
        // round to either 0 or 1
        int roundedX = (unitX >= 0)? (int)Math.ceil( unitX): (int) Math.floor( unitX);
        int roundedY = (unitY >= 0)? (int)Math.ceil( unitY): (int) Math.floor( unitY);
        // compute the new position
        webGrid.addClosestTo(wItem, prevPos.x+roundedX, prevPos.y+roundedY);
    } else {
        // viewer didn't even see this item
        // try to go into the direction of the upper left corner
        webGrid.addClosestTo(wItem, (prevPos.x>1)?prevPos.x-1:1, (prevPos.y>1)?prevPos.y-1:1);
    }
}
}

```

Figure 12: The Currently Employed Update Function: also exists as a java function. Though it only encapsulates specific knowledge about where to place a particular item based on viewer feedback. This particular function computes the preferred direction and tries to place the item at most 1 square from its original position.