CourseDiff: A System for Identifying and Reporting Changes to Course Websites

by

Igor Kopylov

S.B., C.S. M.I.T., 2009

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 2010

Author____

Department of Electrical Engineering and Computer Science
May 19, 2010

Certified by ____

Robert C. Miller
Thesis Supervisor

Accepted by____

Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

COURSEDIFF: A SYSTEM FOR IDENTIFYING AND REPORTING CHANGES TO COURSE WEBSITES

by

Igor Kopylov

ABSTRACT

CourseDiff is a prototype system that periodically samples course websites and notifies users via email when it identifies changes to those sites. The system was developed after conducting a study of 120 web pages from 50 MIT course websites sampled for two months during the spring semester of 2009. The study found that only 18% of changes to the HTML content of course website data are actually important to the content of the page. A closer examination of the corpus identified two major sources of trivial changes. The first is automatically generated content that changes on every visit to the page. The second is formatting and whitespace changes that do not affect the page's textual content. Together, these two sources produce over 99% of the trivial changes. CourseDiff implements an algorithm to filter out these trivial changes from the webpages it samples and a change reporting format for the changes that are identified as important. A small user test on part of the CourseDiff interface indicated that the system could feasibly be used by students to track changes to course websites.

Thesis Supervisor: Robert C. Miller

Title: Associate Professor

ACKNOWLEDGEMENTS

I would like to thank Rob Miller for being a terrific advisor and a constant source of inspiration. I came to Rob with a vague idea about aggregating website changes and he helped me polish it into a real system. A meeting with Rob always generated at least half a dozen new ideas and I can't express how valuable that was in getting CourseDiff up and running.

I would also like to thank everyone else in the User Interface Design group for offering criticism, having great discussions at Tea, and being an all-around cool bunch of people. Thanks especially to Rich Chan and Jones Yu for their feedback at our weekly web meetings.

Finally I'd like to thank my parents, Alla and Dmitri. When I informed them at age ten that I wanted to learn how to program, they shrugged their shoulders and got me a book on HTML. Had they not given me that and a countless number of other stepping stones I would never have made it this far. I can't thank them enough for all they have done for me.

Table of Contents

List of Figures

1. Introduction

The web is a versatile medium for publishing and viewing information. One of its key advantages is that, unlike physical media, the web allows for content that can easily change. As the web evolves over time, web authors can update and reformat their sites to augment, improve and better convey the information they contain. Unfortunately, these changes are not explicitly visible. Website visitors can only compare what they currently see to the memory of a past visit. They cannot easily evaluate how the site has changed and whether that change is important to them.

Furthermore, visitors do not see changes at the time that the author makes them, but only on a subsequent visit to the site. This lack of immediate visibility is unfortunate because information on the web is often time-sensitive. An announcement about a room change has very different consequences if it is received before a meeting than if it is received after. A pushed back deadline can relieve some stress, but not if it is received after the original deadline passes. A helpful hint can speed up a difficult task if it is received before the task is complete, but becomes useless immediately after. The web's lack of change visibility can thus present problems in many domains.

This thesis presents CourseDiff, a prototype system designed to deal with this problem in the domain of university course websites. CourseDiff visits user-specified course web pages automatically and emails a report to the user when it detects a change that may be important. Course webpages are an interesting sample for this research because the changing content is largely the important content. The static content of a course site (e.g. the professor's name, the classroom number, the lecture hours) contains facts that students and faculty already know a week into the semester. The changing content (e.g. announcements, assignments, materials), on the other hand, is both important and time-sensitive.

Identifying changes in documents is a classic problem in computer science [1, 2, 3]. Although "diff" algorithms were originally aimed at string content, algorithms for identifying changes in trees have also been proposed [4, 5]. More recent approaches [6, 7] have focused specifically on tree data in XML format (a common representation for webpages via XHTML). Other research has focused on giving users means to visualize changes to webpages. [8] examines webpages as screenshot images and reports changes based on pixel-level differences. [9] utilizes the underlying tree structure of webpages to highlight changes since a user's last visit. [10] combines both techniques to allow users to traverse a webpage's history.

An alternate approach is for site authors to notify the site's visitors of changes. One popular means of doing this is RSS (Really Simple Syndication) [11]. RSS allows a website author to publish a feed of updates in parallel to their site. Instead of visiting the site, users can use an RSS reader to poll the feed for updates and notify them if one is found. Although systems exist that can automatically update a site's RSS feed when content on the site proper is changed, this approach ultimately relies on the author (or web platform) to correctly identify and report changes to the site; the user has no role in the process.

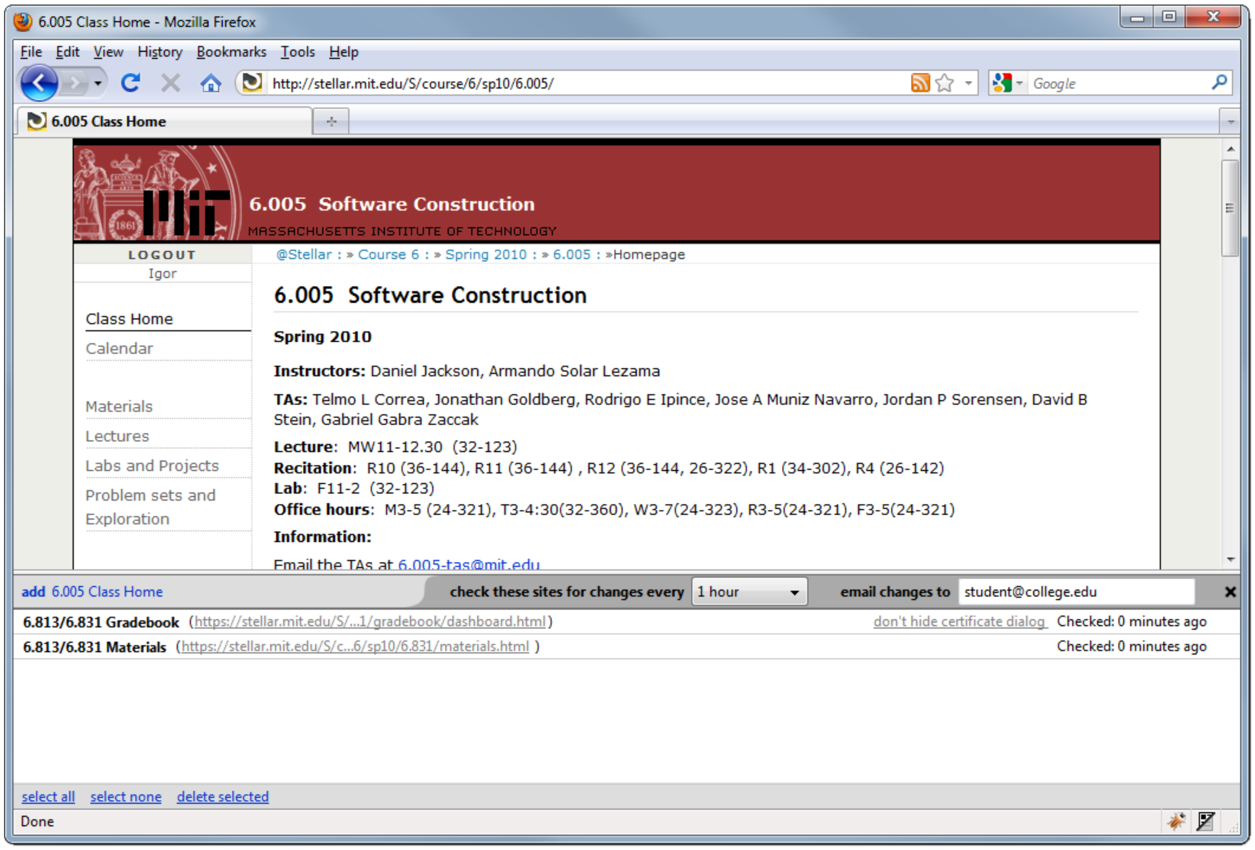
CourseDiff applies some of the previous change identification and visualization work to course websites. The goal of the system is to give users of these sites (students) timely and meaningful reports of changes without any involvement of the site's author (faculty).

CourseDiff's development was based on a study of 50 MIT course websites sampled during the spring semester of 2009. This study found that a rather small portion (18%) of changes to the html source of a course web page actually translates to visible and informative changes for the user. In other words, a naïve approach that reports every update to the html would flood the user with trivial changes. Although the initial change visibility problem would be resolved, it would be replaced with a large change filtering problem.

A closer inspection of the sampled corpus identified two main sources for the trivial changes. The first is automatically-generated web content. Examples of this include site counters, timestamps and randomly generated images. Each visit to a page that has this type of content may produce a different html source, but none of those visits may actually have changes that contain important information. The second source of trivial changes is formatting. Examples of this include changing colors or fonts and adding spacing or dividers. Such changes can be created intentionally by the site author, as well as introduced automatically by the site's platform. By taking both of these sources of trivial change into account, CourseDiff is able to filter out 99% of the trivial changes in the corpus while reporting 100% of the non-trivial ones.

Users interact with CourseDiff in two ways. The first is a configuration panel built into the browser, which allows the user to specify which webpages to track, how often to check for changes to those pages and where change reports should be sent. The second is the change report itself, which the user receives via email every time CourseDiff detects a non-trivial change to one of the specified pages.

The configuration panel displays a list of the pages that CourseDiff is currently tracking. At any time, the user can add the page they are currently visiting to the list by clicking the link in the top left corner of the panel. The user can also use this panel to delete and rename pages already on the list. Once the user is done interacting with the panel, they can close it to leave more room for browsing.


6.005 Class Home - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://stellar.mit.edu/S/course/6/sp10/6.005/

6.005 Class Home

6.005 Software Construction

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LOGOUT
Igor

@Stellar : » Course 6 : » Spring 2010 : » 6.005 : »Homepage

Class Home
Calendar
Materials
Lectures
Labs and Projects
Problem sets and Exploration

6.005 Software Construction

Spring 2010

Instructors: Daniel Jackson, Armando Solar Lezama

TAs: Telmo L Correa, Jonathan Goldberg, Rodrigo E Ipince, Jose A Muniz Navarro, Jordan P Sorensen, David B Stein, Gabriel Gabra Zaccak

Lecture: MW11-12.30 (32-123)
Recitation: R10 (36-144), R11 (36-144) , R12 (36-144, 26-322), R1 (34-302), R4 (26-142)
Lab: F11-2 (32-123)
Office hours: M3-5 (24-321), T3-4:30(32-360), W3-7(24-323), R3-5(24-321), F3-5(24-321)

Information:

Email the TAs at 6.005-tas@mit.edu

add 6.005 Class Home | check these sites for changes every 1 hour | email changes to student@college.edu

6.813/6.831 Gradebook (https://stellar.mit.edu/S/...1/gradebook/dashboard.html) don't hide certificate dialog Checked: 0 minutes ago

6.813/6.831 Materials (https://stellar.mit.edu/S/c...6/sp10/6.831/materials.html) Checked: 0 minutes ago

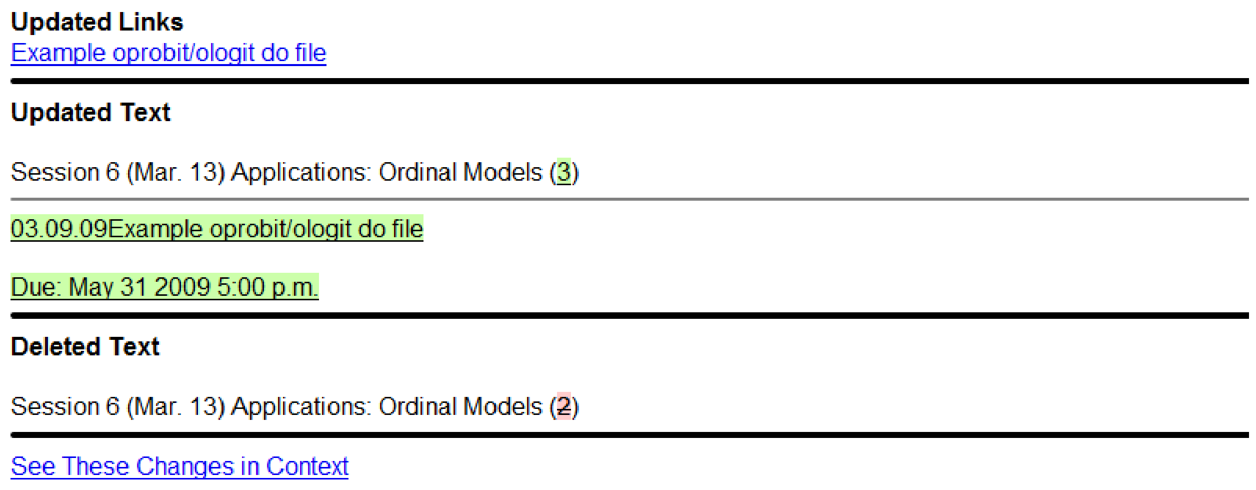
select all select none delete selected

Done

Figure 1 CourseDiff running in the browser

Change reports are sent to the email address the user specifies in the configuration panel. The report identifies both updated and deleted textual content. Text that was added as a result of the change is highlighted in green and underlined. Text that was removed is highlighted in red and crossed out. Any updated or deleted links are presented before the updated or deleted content, respectively. A link at the bottom of the report allows the user to view these updates and deletions in the context of the original page (i.e. overlaid onto the original page content).

Like the user interface, the CourseDiff backend has two high-level components. The first is an extension to the Firefox web browser. This extension adds the configuration panel to the browser and is also responsible for downloading each of the specified web pages, checking the downloaded content for changes and generating a change report if there are some. The second component is a web server, which is responsible for storing all of the change reports and emailing them to the user. When the extension identifies a non-trivial change it uploads a report to the server along with the user-specified email address. The server then generates the email and sends it to the user.



Updated Links
Example oprobit/ologit do file

Updated Text

Session 6 (Mar. 13) Applications: Ordinal Models (3)

03.09.09Example oprobit/ologit do file

Due: May 31 2009 5:00 p.m.

Deleted Text

Session 6 (Mar. 13) Applications: Ordinal Models (2)

See These Changes in Context

Figure 2 A change report

A small user study of MIT undergraduates was conducted to assess the usability of the CourseDiff configuration panel and asses the feasibility of students using such a system. Users mostly found configuration panel straight-forward and were able to add, rename and remove sites without difficulty. They expressed interest in using the system and confirmed that email was an appropriate way to deliver changes.

The rest of this thesis is organized as follows. Chapter 2 discusses related work. Chapter 3 details the course website study. Chapter 4 explains the design decisions that went into the CourseDiff backend. Chapter 5 describes the backend implementation. Chapter 6 explains the user interface components of CourseDiff. Chapter 7 evaluates the user interface. Chapter 8 is the Conclusion, and chapter 9 is Future Work.

2. RELATED WORK

2.1 RSS

RSS [11] is a markup language that can be used to create short, frequently updated snippets of an entire web page. The sequence of these snippets over time is collectively called an RSS feed. Users typically subscribe to a feed using an RSS aggregator, which may be a standalone application or built into a web browser. RSS allows a site's creators to actively deliver content to users instead of passively posting it on the web as they usually do and hoping that users will look at it.

Glotzbach, et al [12] examine the effectiveness of using Really Simple Syndication (RSS) as a delivery mechanism for course information. The study found that although using RSS increased students' general awareness of the technology, few took advantage of the course's announcements feed. The feed's low activity was cited by the researchers as the most likely reason for this.

This study is somewhat dated, as RSS has since grown in popularity [13]. That is, students may be more likely than before to be using an RSS aggregator and subscribing to many feeds already. The strength of RSS is that with many feeds coming in at once, low activity on a single one becomes a lesser concern. Also, because RSS feeds are created by the authors of the original web content, the identification of important changes is entirely decoupled from the aggregation of multiple feeds. Hence, the aggregators have a great deal of freedom about how to organize and filter the information they collect. RSS's simplicity in terms of both content and format makes it substantially easier to parse than web pages themselves. Many different aggregation schemes have been proposed and a host of products are available that make RSS aggregation very efficient even with the ever increasing number of feeds.

However, because RSS leaves the identification and reporting of changes entirely up to the site author, the content available for aggregation has limitations. An RSS feed may be a high fidelity representation of a website from the author's point of view, but not necessarily from that of the user. That is, the two may have different ideas about what changes to the site are relevant or important. More practically, certain web content (animations, forms) cannot be represented in RSS. The biggest issue, however, with leaving extraction to the authors is that they simply might not do it. Although in [12] a system automatically generates RSS notifications from course announcements, not every site can be expected to have such a system, and for more complex web content, such a system becomes less feasible. Therefore, RSS is ultimately limited by the willingness of any web author to provide a feed of the changes to their site.

2.2 Notification Collage

Notification Collage (NC) [8] takes an approach that solves the issue of author involvement. An NC user can visit any web page, select a rectangular region of that page and then place that region into a "collage" of other such clippings. A service then visits each clipped page, checks the rectangular region for visual changes and, if there are any, updates the user's collage with the new image. Each item also has a history, so the user can look through any previous changes the system has identified. No help from the site author is required, and since NC deals only with the final, rendered web content, any webpage no matter how complex can be captured this way. The extraction method captures exactly what users would see if they visited the page.

The downside is that the information is clipped from the sites by screen location alone. Users who do not know exactly where the content they want to track is going to be located are likely to have difficulties. For example, a webpage that lists course announcements might place new announcements at the top of the list or at the bottom. Which of these alternatives is actually in place may not be obvious until the site is actually updated. But, of course, by then an update might already be missed. A possible solution to this problem is to have NC track the entire page. This, however, will generate a collage image the size of the entire page. Identifying the actual change within that image (perhaps a few lines) could be quite difficult for the user.

A fundamental issue with NC is that since the collected data is an image, the identified changes are necessarily pixel-level changes. NC identifies all the pixel-level changes to a site and allows the user to see a history of those changes, but the user must ultimately figure out what the change is. For content that changes substantially (e.g. a webcam), this is not a problem. But for content with subtle changes (e.g. a course announcement page) this can be a considerable difficulty. The actual change to an announcement page – the textual change – is lost in converting the text of the webpage into an image.

2.3 Zoetrope

Zoetrope [10] is a utility that uses more sophisticated extraction techniques to get at both the visual and textual content contained in webpages. First, Zoetrope downloads a webpage at regular intervals over some period of time (e.g. daily for six months) giving it a history of that page's evolution over that time. After that, Zoetrope provides a series of tools (called lenses) for examining how regions of the page changed over the collected history. Zoetrope allows region selection by pixel location, by location in the HTML document, and as text.

Zoetrope has a rich repertoire for viewing the collected history. Lenses can be filtered by keywords. Changing numerical data can be graphed over time. Zoetrope can also aggregate multiple pages by linking lenses together. So, for instance, announcements from the history of two pages can be viewed side by side and examined for relationships.

Zoetrope is not, however, intended to view the latest changes to a page. Since the system is designed for viewing how pages evolve over a substantial length of time, the content of the page at any moment in time falls to the background, as visualizations of values and images changing over time take center stage. Also, the content of any lens must be viewed in the context of the original page and lenses cannot be organized on screen (as they might be in NC). Ultimately, the interface is intended for retrospective examination of a webpage, not for summarization of its latest changes.

2.4 X-Tree Diff

X-Tree Diff [6] is an algorithm for identifying changes between two XML trees (the old and the new). The algorithm decorates each node of both trees with an *index*, *operation* and *pointer*. The *index* is a unique number assigned to each node. The *operation* is one of:

- NOP – the node was unchanged
- ADD – the node was added
- DEL – the node was deleted
- UPD – the node's XML attributes were changed
- MOV – the node was moved from one parent to another

DEL can only occur in the old tree and ADD can only occur in the new tree. The *pointer* is the index of the node in the old (new) tree that corresponds to a node in the new (old) tree when the *operation* is NOP, UPD or MOV.

The algorithm works in 4 stages:

1. **Match tree nodes with a 1-to-1 correspondence.** For both trees, a hash is calculated recursively for every XML node based on its attribute-value pairs and the hashes of its children. Nodes in the old and new tree which have exactly matching hashes are matched with a NOP.
2. **Propagate matches up the tree.** For every match from stage 1, the two matched nodes' parents are compared. If they have the same XML node name, the two parent nodes are also matched with a NOP. If the two parent nodes do not have the same node name, the two nodes are marked with a MOV. So long as the parent node names match, the process continues up both trees until one of their roots is hit.
3. **Match unmatched nodes down the tree.** The tree is traversed, looking for matched nodes which have unmatched children. Children are matched between every two such nodes. First children that have the same hash values are matched, then children with the same node names and attribute-value pairs, and finally children with the same labels alone.
4. **Mark unmatched nodes with ADD and DEL.** At this point any nodes which have not been matched are assumed to be have been added or deleted. All unmatched nodes in the old tree are marked DEL and all unmatched nodes in the new tree are marked ADD.

Though the HTML that websites are written in is not necessarily XML, the main features necessary for this algorithm (tree structure, node names, attribute-value pairs) are present. Indeed, whatever language the page is written in, browsers will convert the page into a tree structure called the Document Object Model (DOM) as part of rendering it. It is not surprising that the testing discussed in [6] involved applying the algorithm to approximately sixty webpages.

A limitation of this algorithm is that it only deals with nodes. This can be problematic when dealing with textual content. In the DOM, text is a leaf node. This algorithm can identify that the text was changed (UPD), but it does not identify how. This means that whether the textual change is one character or one paragraph, the algorithm will report essentially the same information. That is, the tree will be decorated the same way. Another issue is that this algorithm does not, of course, explain how the change should actually be presented to a user. It can decorate a webpage with meta-information, but this does not actually change what the user sees.

2.5 DiffIE

DiffIE [9] is a browser extension specifically designed to visualize tree-level changes to the web pages that a user visits. DiffIE caches a copy of a webpage every time the user visits it. When the user returns to that page, the system compares the newly downloaded version against the one in the cache using a tree diff algorithm (similar to X-Tree Diff). DiffIE then highlights the added and updated portions of the webpage to bring them to the user's attention.

A user study of this system showed that making webpage changes visible introduced a number of new ways for users to interact with the web. Many users used DiffIE to monitor sites that they expected to change. However, [9] claims that DiffIE was particularly useful in cases where users did not expect change. Users found that some sites changed when they did not expect them to and many were drawn to content that they would not have been drawn to otherwise.

DiffIE is able to highlight surprising changes because it looks for changes in every site that a user visits. This does not become a burden on the user, since they only see changes to a site when they choose to visit it. However, if DiffIE were to report every change it identifies via a medium like RSS, users would likely be overwhelmed. Also, since the highlighting affects the user's view of every site they visit, it cannot be particularly distracting. For this reason, DiffIE does not display deletions, as this would require adding parts of the document that were removed.

DiffIE also (by virtue of the tree diff algorithm it uses) does not identify how textual content changes. Though [9] notes that this functionality could be added, this is not done because extra information would have to be stored for each visit to the page. Since DiffIE caches every page that a user visits, any extra data per visit can significantly increase the disk usage of the system. So although DiffIE is quite good at uncovering unexpected changes to sites, it does not provide very powerful tools for examining those changes.

3. COURSE SITE STUDY

3.1 Corpus Collection

MIT course websites come in many flavors. Some are hosted on MIT's official course management system, Stellar [14]. These sites have a common general layout (though they are customizable) and have many features (like RSS) built in automatically by the Stellar platform. Other course webpages are hosted on department specific sites. Some have their own independent web location, which they have had for many years. Some pages are created by course staff in personal web spaces that they own.

course.mit.edu [15] is a service that help students find the website for their classes. At the beginning of every semester it automatically checks the common locations of course websites. After that, users can search for a course by its number. The first person to do so is presented with the possible locations. They can then choose one as the correct site or provide another location instead. Visiting course.mit.edu/*course#* directs to the location that was most recently associated with the course. A few weeks into the semester, once students have had a chance to identify the websites of the courses in which they are enrolled, the web locations of many courses are available through this service.

A scrape of course.mit.edu 3 weeks into the Spring Semester of 2009 found websites for 1233 courses. Though 91% of the sites found this way were located on the Stellar course management system, this is likely an overestimate of the number of course websites that use the system since course.mit.edu adds locations on Stellar automatically. For some classes, these sites are generated but not used. These were culled to those which seemed to have recent activity and then, of those, 50 were chosen at random – 20 from Stellar and 30 not. Each of the selected 50 sites was visited manually and (1-4) pages on that site that appeared likely to change were added to the corpus. This resulted in a final corpus of 120 pages.

Though sites not hosted on Stellar were almost certainly oversampled, doing so captured a greater variety of course websites with the goal of designing a more general system. If the collected corpus consisted of 91% Stellar sites, any conclusions drawn from it would essentially be a referendum on Stellar, and not a statement about course websites in general.

WebPageDump [16] was used to download each webpage in the corpus hourly for 61 days. Each sample contains a single HTML file for the sampled page, plus any resources (images, style information) that this file needs. A screenshot of the page was also captured. The final corpus has over 150,000 samples and is approximately 30GB in size.

3.2 Identifying Changes

Once the corpus was collected, the contents of every sample's HTML file were compared against the contents of the sample that preceded it. In the first pass, contents were considered different so long as they were not exactly, byte-for-byte, the same. No examination of the HTML tree structure was considered at this point. Of over 150,000 hourly samples only 7,338 were different from the sample before. This suggests, not too surprisingly, that course webpages do not change very quickly (in Internet terms). The average rate of this type of change is roughly once per day.

An examination of the number of changes per page shows that the distribution is far from uniform. The mean number of changes over the sampling period is 67, and the standard deviation is 207. The maximum is 1443 and the minimum is 0 (exhibited by 10 pages). The two most frequently changing pages are actually different in *every* sample.

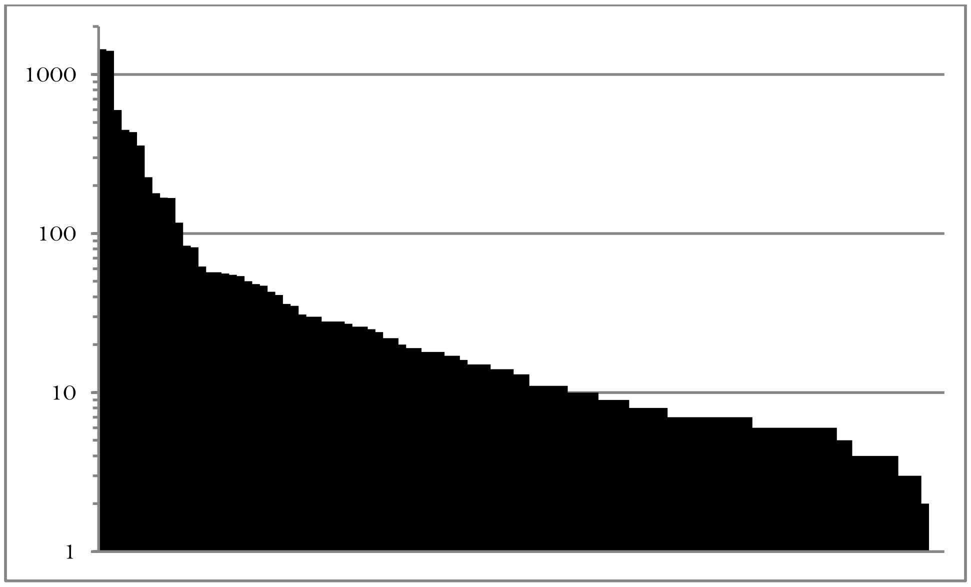


Figure 3 Number of Changes by Page

This suggested that some of the changes are not actually important to the course information of the site. It seems highly unlikely that the course staff is updating a website hourly with new information. Even the 1-change-per-day rate is intuitively somewhat high. Certainly, there are not assignments going out every day for every course. This intuition prompted a second pass over the data.

Each of the 7,338 identified changes was manually examined to determine whether the change to the file was important or trivial. Because CourseDiff is targeted at the course website domain, the definition of "important change" was tailored accordingly. A change to a course webpage is considered important if that change provides new information that might be valuable to a participant in the course. Changed due dates, new announcements and added readings are all important changes. Typo corrections, updated formatting and incrementing visitor counters are not, since they do not provide any new information about the content or administrative details of the course.

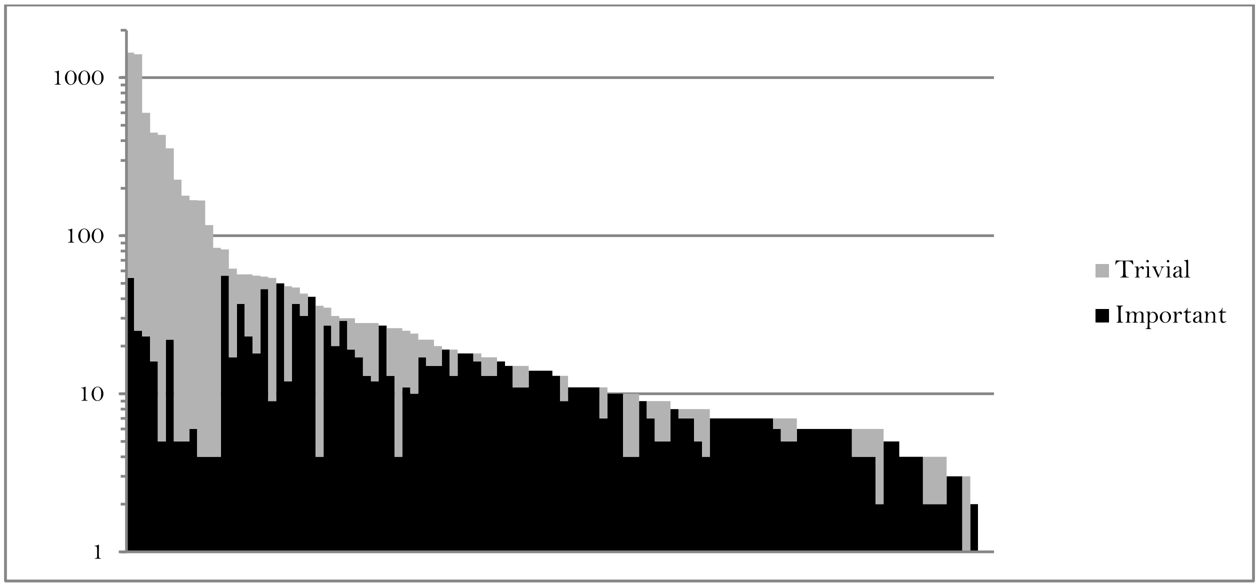
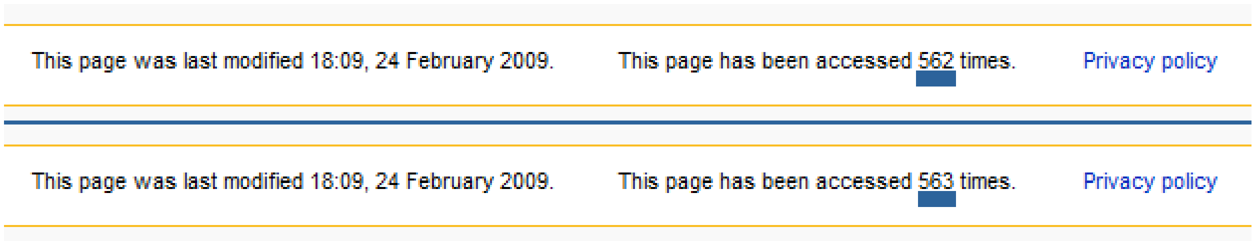


Figure 4 Trivial vs Important Changes by Page

Only 18% (1,333) of the changes were found to be non-trivial. This corresponds to an average of 1.3 changes per week, which matches intuition of how often material is released on a course website. The mean number of "important" changes over the sampling period is 12 and the standard deviation is 11, still showing a rather wide range in update rates.

Notably, the two pages with changes occurring every sample do not actually produce important changes on every sample. The reason these sites produce so many changes to their HTML content is that some of that content is automatically generated. One of the sites has a visitor counter. Every time the site is sampled, the counter increments and the content of the page changes. However, this change is likely of no interest to the user. Certainly, they do not want to be notified 1400 times that something has changed. The other site has a section of randomly chosen images. In this case, again, the changing content has nothing to do with course information. The change is largely an aesthetic measure and the user does not want to be notified about this kind of change.


This page was last modified 18:09, 24 February 2009. This page has been accessed 562 times. Privacy policy

This page was last modified 18:09, 24 February 2009. This page has been accessed 563 times. Privacy policy

Figure 5 A visitor counter. Example of every-sample change

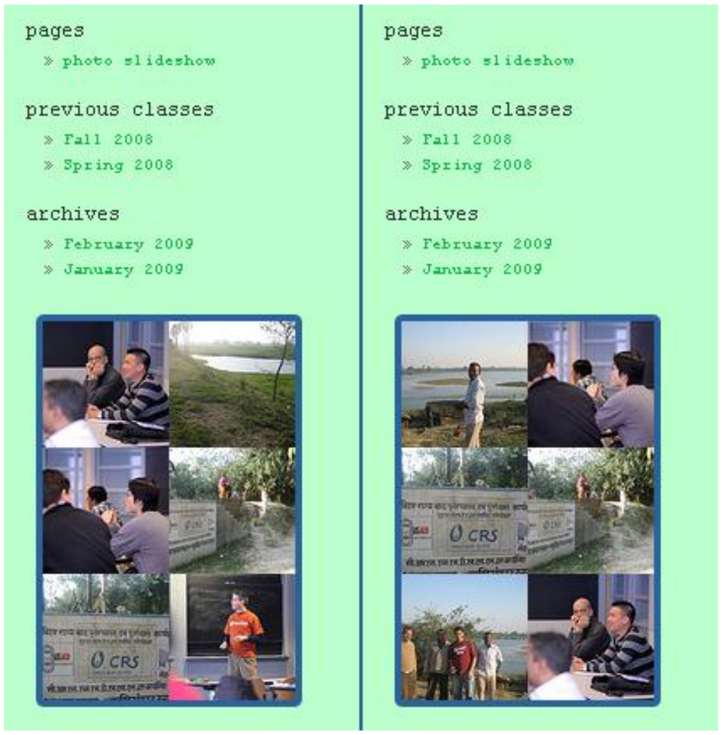


Figure 6 Randomly generated pictures. Example of every-sample change

The presence of every-sample changes accounts for a substantial amount (38%) of the trivial changes, but it does not nearly account for all of them. The next examined source of trivial changes was formatting. That is, a change might consist of modifications to the font or color of some web page content but not the underlying text. A concern here is that, perhaps, in some cases a change in formatting could be non-trivial. So if formatting changes produce not only trivial, but also important changes then ignoring them entirely is not an option.

To probe formatting changes, first, the every-sample changes were removed from the data set. After that, each remaining sample which contained a change was rendered in the browser and the document tree was traversed to extract only the textual content. This included not only the content of text nodes, but also other relevant textual information contained in attributes. Examples of the latter include the *href* attribute of anchor (link) elements, the *alt* and *src* attributes of image elements and the *title* attribute of any elements that have it. Also, the textual content not visible to the user was ignored. This included scripts, style information and comments. Once the textual content was extracted from each site, each of the changes in the corpus was examined again to see if the text content had also changed.

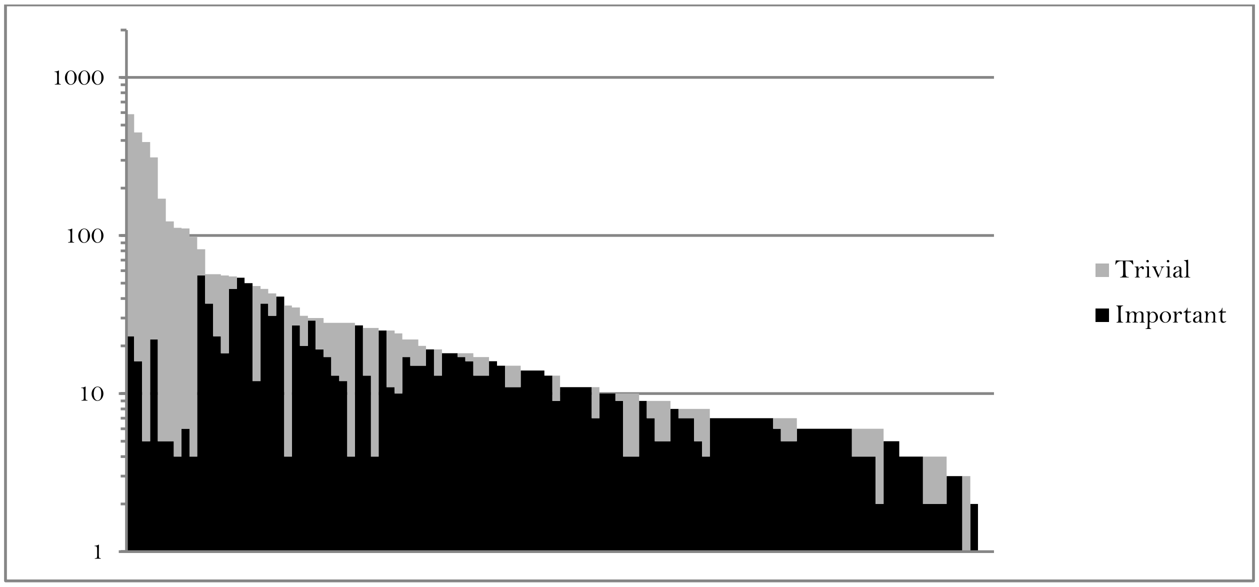


Figure 7 Textual Changes by Page

Some of the intuition behind ignoring formatting was confirmed by the fact that all important changes in the corpus were indeed textual changes. However, many of the textual changes are not important. Only 33% of the textual changes are important overall, and for the worst case page, only 1% of textual changes are important. On the other hand, for 47 of the pages 100% of the textual changes are important.

This final discrepancy indicated that some kinds of textual changes are clearly related to importance while others are not. A closer inspection of the changes where textual changes were not important revealed that in many cases these textual changes involve only whitespace – tabs, spaces, newlines and carriage returns. Since whitespace is largely ignored when HTML is rendered, these changes generally have no impact on what the user sees. However the HTML file is changed, and furthermore, the textual content is changed. This result suggested that a better metric of change is not textual change, but non-whitespace textual ("nw-textual") change.

Much the same process that was carried out for identifying textual change was applied to identifying nw-textual change. Each sample with a change (excluding every-sample changes) was rendered in the browser and the document tree was traversed to extract the textual content. This time, however, all textual content was stripped of whitespace. Again the changes were evaluated to see if the nw-textual content also changed.

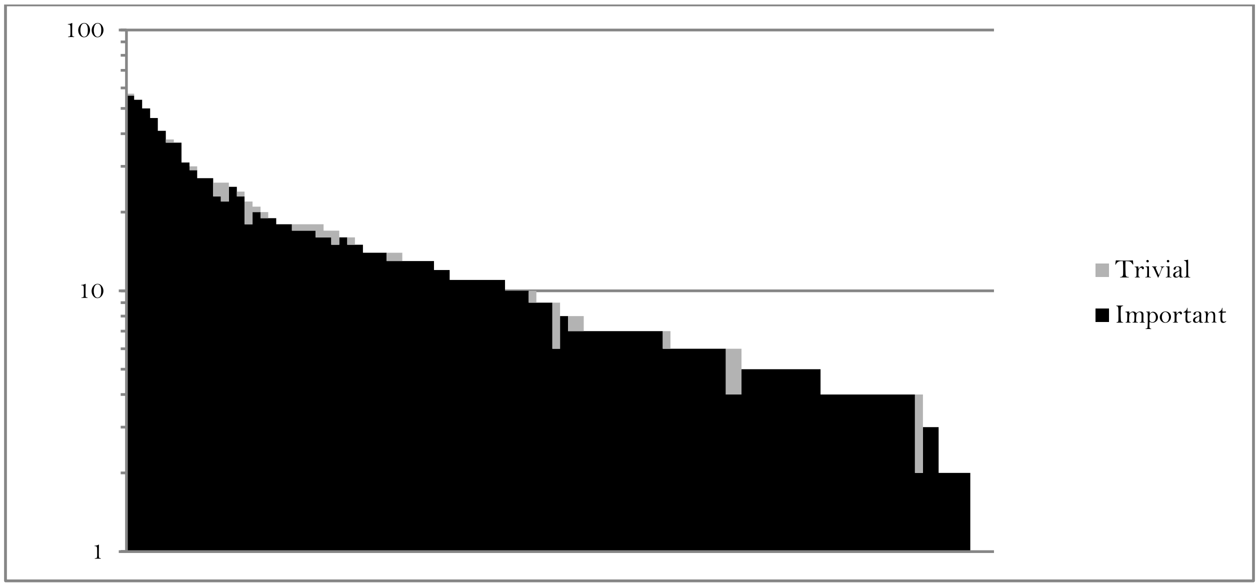


Figure 8 NW-Textual Changes by Page

The results show a very close relationship between nw-textual changes and important changes. Overall, 97% of nw-textual changes are important. For 84 of the pages in the corpus, 100% of the nw-textual changes are important. Equally significant, all important changes in the corpus are nw-textual. This means that trying to filter the corpus for important changes by looking for nw-textual change produces no false negatives.

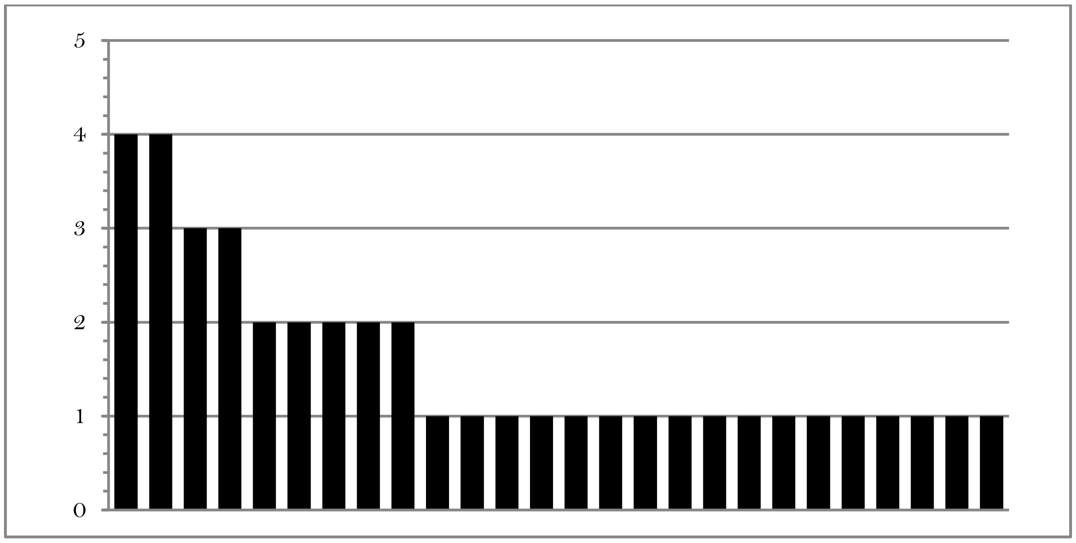
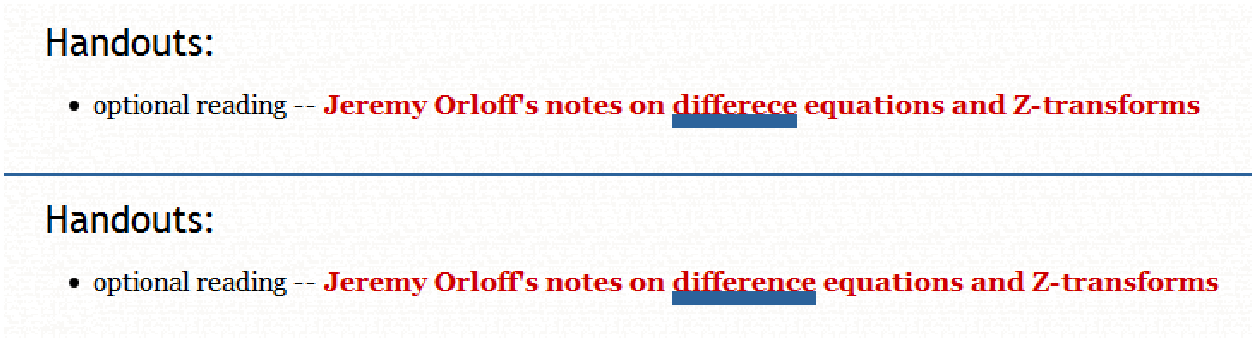


Figure 9 Trivial NW-Textual Changes by Page

The fact that nw-textual changes are much closer to important changes than plain textual changes indicates that many sites have changes that are purely whitespace. The source of this whitespace is not entirely clear. Some of it is likely introduced by HTML editors. Opening and then saving a file may add some extra whitespace characters without changing any other content. However these changes occur often enough to suggest that they cannot all be created by site authors. Some of the change may come from the web platform that serves the page. Though the author might not change any of the content, the platform might automatically add some whitespace to the boilerplate of the site depending on the date or time. Since this whitespace does not generally cause changes to how the rendered page looks, these sorts of changes are easy for platform designers to overlook.

3.3 Change Identification Limitations

By throwing out every-sample changes and then only considering nw-textual changes a very large portion of trivial changes was filtered out from the corpus. Only 41 of the 6005 (<1%) of trivial changes make it through the process. However, the filter does have its limitations.



Handouts:

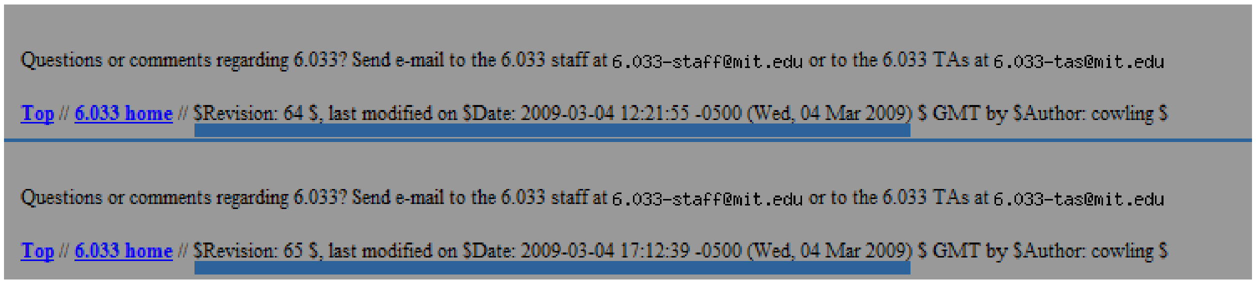
- optional reading -- Jeremy Orloff's notes on differece equations and Z-transforms

Handouts:

- optional reading -- Jeremy Orloff's notes on difference equations and Z-transforms

Figure 10 A fixed typo – a trivial one character change

The fundamental problem with identifying nw-textual changes is that even a single character is enough to generate one. When that that change is something like fixing a typo or adding a period, it could easily not be important. However, attempting to set a minimum bar on the length of a change is not a good solution. For example, a changed due date might be a one character change, but is nonetheless important. Indeed, the corpus contains 20 one character nw-textual changes, 17 of which are important.



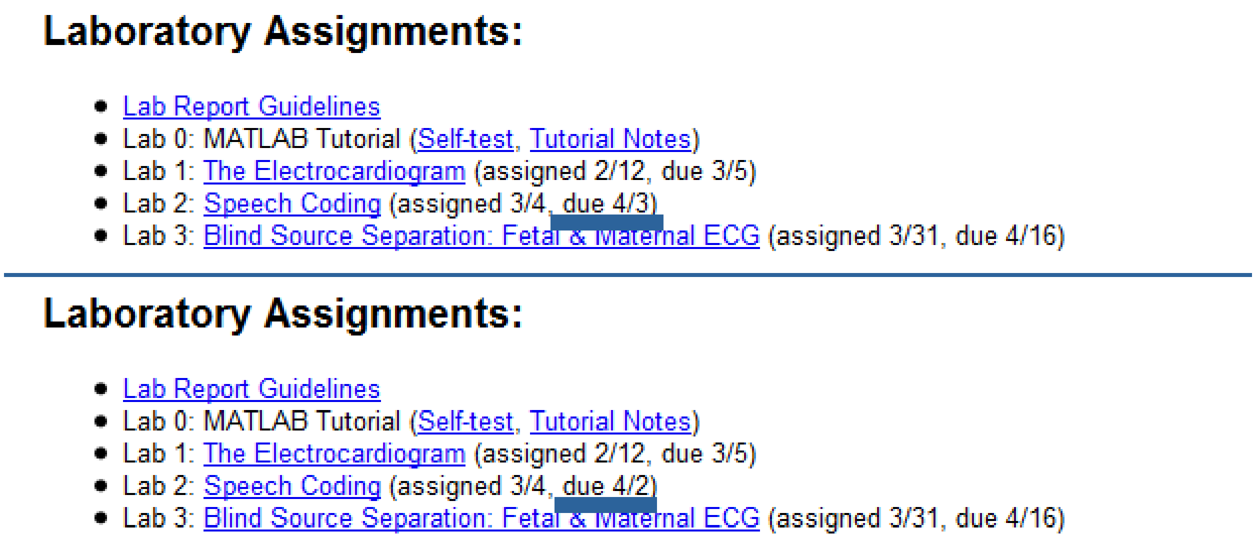
Questions or comments regarding 6.033? Send e-mail to the 6.033 staff at 6.033-staff@mit.edu or to the 6.033 TAs at 6.033-tas@mit.edu

Top // 6.033 home // $Revision: 64 $, last modified on $Date: 2009-03-04 12:21:55 -0500 (Wed, 04 Mar 2009) $ GMT by $Author: cowling $

Questions or comments regarding 6.033? Send e-mail to the 6.033 staff at 6.033-staff@mit.edu or to the 6.033 TAs at 6.033-tas@mit.edu

Top // 6.033 home // $Revision: 65 $, last modified on $Date: 2009-03-04 17:12:39 -0500 (Wed, 04 Mar 2009) $ GMT by $Author: cowling $

Figure 11 A revision change – a trivial nw-textual change



Laboratory Assignments:

- Lab Report Guidelines
- Lab 0: MATLAB Tutorial (Self-test, Tutorial Notes)
- Lab 1: The Electrocardiogram (assigned 2/12, due 3/5)
- Lab 2: Speech Coding (assigned 3/4, due 4/3)
- Lab 3: Blind Source Separation: Fetal & Maternal ECG (assigned 3/31, due 4/16)

Laboratory Assignments:

- Lab Report Guidelines
- Lab 0: MATLAB Tutorial (Self-test, Tutorial Notes)
- Lab 1: The Electrocardiogram (assigned 2/12, due 3/5)
- Lab 2: Speech Coding (assigned 3/4, due 4/2)
- Lab 3: Blind Source Separation: Fetal & Maternal ECG (assigned 3/31, due 4/16)

Figure 12 A due date change – an important one character change

Another type of nw-textual change that is not important is revision numbers. A page may contain a line of text to indicate the current version of the document. This text is usually automatically generated, but since it does not happen on every sample, it is not filtered out. Usually when the revision changes, this indicates that some other part of the page has also changed. So although the time stamp itself might not be important the overall change to the page is. Unfortunately, in some cases, the time stamp can update even when the rest of the page is unchanged, creating an nw-textual change that is not important.

4. System Design

Having found an effective algorithmic tool for identifying important changes to webpages, the next step in developing CourseDiff was to design a tool that used this tool to create reports of changes to course websites.

The first major decision in the design process was how to sample the site. That is, how to download the versions of the page to compare for changes. One approach would be to have a central server that users could request to track a site. The server would download and store samples of the site, compare them for changes, and generate reports when they were found. The server would be able to merge requests for the same site, downloading it only once for all of the users who want to track it and sending a change report to each one. The server would scale with the number of courses, not the number of students. If the system became popular the load on the course website server might even decrease as users visit the page less and less.

Unfortunately, though such a design has many nice features, it is not practicable because some course websites require the user to have certificates in order to view some of their pages. What's worse is that course pages that are openly available at the beginning of the term can become closed to those not enrolled in the course later on. This means that unless users are willing to give the central server their certificates (a potentially large security risk) the server cannot, in general, download the content of course webpages.

A potential compromise to this limitation would be to say that the system will only deal with sites that do not require certificates. However, this approach would fragment the pages that the user is tracking into two groups: the ones that are being tracked by CourseDiff, and the ones that have to be tracked manually. This can become a tricky distinction to keep track of, especially when some pages on a course site are certificate-protected and some are not. Overall, the system would not be as complete or cohesive if it did not allow users to track all of their course sites.

To give the user this ability, CourseDiff downloads webpages from the user's browser. This approach gets around the certificate problem, since the user's browser already has certificates. If the user can visit a page in their browser, then so can CourseDiff. There are, however, some drawbacks of this approach compared to the central server. The user must have the space to store the collected samples. Even though hundreds of students might be in the same class and tracking the same page, each student must download their own copy. This sampling also increases the burden on the course site, as hundreds of students may begin requesting pages every hour. This load is likely larger than the site would have experienced otherwise, but certainly not so large that it should overload the site. However, if the system had to handle sites with many more users or sites that needed higher sampling rates, then this issue would become a concern.

Also, certificates can pose a different problem for sampling in the browser. As the browser samples pages, it loads each one in the background. If it tries to sample a page that requires certificates, the user may be presented with a dialog asking for confirmation. This could be quite unexpected because the page that requested the certificate is not visible and the user never directed navigation to it. To avoid this kind of surprise, CourseDiff makes it clear when a certificate dialog pops up because of sampling. Also the user is given the option to automatically confirm these dialogs in future sampling.

Another issue with sampling in the browser is that the browser might not always be on. A central server can be set up to run constantly. A user's browser, however, might be opened and closed. The computer might even be turned off. For sampling in the browser to work, users must have their browser turned on often enough for changes to be caught in a timely fashion. Of course, if users were to just visit the site (i.e. sample manually) they would have to turn their browsers on. So, in the worst case, sampling in the browser could still save the user from having to visit the sampled page, though identified changes might not be caught in a timely fashion.

A big benefit of downloading samples in the browser is that the browser has built in mechanisms for generating, modifying and traversing the DOM. For example, on a server, some tool would be needed to parse the downloaded HTML document into the DOM. But since the browser is designed to display downloaded webpages, it already has this tool built in. By working directly in the browser, tools become available that would need to be installed separately on a server.

The second major design decision was to choose where to store the samples. If the browser were to store all of the samples, then the user would have to worry about disk space. This would require some mechanism to manage the amount of space that sampling is allowed to take up. DiffIE runs into this issue and does provide an interface to manage disk space. Another drawback of storing samples on the user's computer is that they are inaccessible from other devices. In other words, any identified changes can only be viewed on the machine where they were identified. This is a big disadvantage over the central server model, which could, in principle, deliver changes to any device with an internet connection.

CourseDiff takes a hybrid approach when it comes to storing samples. Samples are stored both on the user's machine and on a server. At most two samples of any page are ever present on the user's machine – the current sample and the one before it. This could be thought of as a queue with two elements. When a sample is downloaded, the one that came before it is shifted over and the one before that is deleted. The current sample is compared with the one before it for changes. If a change is found, a report is created and the entire sample and the report is uploaded to the server. In other words, the server only stores changes. If no change is found, the sample will just be replaced with the next one when the time comes.

This approach keeps a constant disk space cost on tracking each page. This cost is only twice that required for keeping the page in cache, which the browser does automatically anyway. The low cost allows CourseDiff to avoid forcing the user to manage space on their machine. The disk space required on the server does, on the other hand, increase with every change. Still, at an average of 1.3 changes per week and with an average sample size of 200KB, one user tracking one course would generate 3.9MB a semester. 1GB of server hard drive space could support over 50 users with 5 classes each.

The final design decision was how to report identified changes to the user. Because change reports are stored on the server, they can be delivered to any device, not just the machine that did the sampling. There were essentially three possibilities for delivering the content: a custom application, RSS and email. A custom application would be both time-consuming to build and would require users to learn a new system, so the decision was primarily between RSS and email.

RSS is flexible with respect to the content it allows in feeds. Though not all HTML content is allowed, much of it is. Users could integrate the feed generated by CourseDiff in with the rest of the feeds they read in their RSS aggregator, making it part of their routine. On the other hand, [12] does indicate that students may not be interested in following feeds from course sites. For people who do not use RSS already, using CourseDiff would require learning another new interface as well. Also, since RSS aggregators work by polling feeds for changes, using an

RSS aggregator on content which was generated by sampling another site indicates a certain redundancy.

Email does not have the flexibility of RSS. Emails can (in practice [17]) only contain a limited subset of the content found in HTML pages. Organization ability varies greatly depending on the email client, though more powerful tools are available to users who want them. Email's chief strength is that it is very popular among students and faculty for communication. Students typically receive many email messages already and could almost certainly incorporate an emailed change report into their routine.

Ultimately, CourseDiff uses email in favor of RSS. The main motivation is that more users are likely to be using email already. The fact that email cannot display the content of the original HTML page is not particularly troubling because important changes are textual changes. In other words, as long as the textual change can be conveyed, the change report is likely to make sense. As a precaution, in case the textual change is sensitive to the context it appeared in on the page, the emailed report provides a link to see the change in context. Unfortunately, this requires leaving the email client and visiting a webpage, perhaps breaking the email workflow. The hope is that most of the time the textual change alone is enough.

5. SYSTEM IMPLEMENTATION

CourseDiff is implemented as two high-level pieces: a browser extension for Firefox that is responsible for sampling webpages, identifying changes and creating a report of those changes; and a server that is responsible for storing the reports generated by the browser extension and emailing them to the user.

5.1 Browser Extension

The browser extension has two main functional components:

- **Dispatcher** an XPCOM service that interacts with the database and initiates sampling
- **DiffWindow** a Firefox window that actually downloads each page and identifies changes

The Dispatcher is a singleton, created when the Firefox starts. Every Firefox window, when it loads, registers itself with the Dispatcher. The first window to do so becomes the DiffWindow. The Dispatcher has no direct access to the DOM, so it uses the DiffWindow to perform operations that require it. If the DiffWindow is closed, one of the other registered windows becomes the DiffWindow. If all registered windows are closed, the Dispatcher cannot initiate sampling and is forced to wait until another window opens and registers itself.

Data about which sites to sample is stored in a SQLite database in the user's Firefox profile directory. The database contains two tables.

1. `page (id, url, name, lastsample, lastattempt, samplecount)`

`page` stores the pages which are meant to be tracked by CourseDiff. A unique id, the `url` and a user specified `name` are in the table for each page. The table also stores the last time a page was sampled, the last time a sample was attempted, and the number of times the page was sampled.

2. `skipcert (pageid, certsite, certname, certindex)`

`skipcert` stores information about certificate dialogs that the user requested to skip when they appear as a result of sampling. The table stores the `pageid` of the page that was sampled when the dialog came up. `certsite` identifies the site that requested the certificate and `certname` identifies the name of that site. `certindex` identifies which certificate the user selected to access the site.

The sampling period and the email address to which the report is sent are not stored in the database, but as Firefox Preferences. This eliminates the need for another table (e.g, `preferences`) and allows these settings to be more easily integrated into the UI.

Sampling Pages

The Dispatcher runs a query to the database every 10 seconds, to select one page which is overdue for sampling. A page is overdue when it was last sampled longer ago than the sampling period (i.e. when `lastsample ≤ now - period`). This query is ordered by the last attempted sample time, so that if a sample fails, other pages will have a chance to be sampled before the failed page gets another chance. If the query returns a result, the Dispatcher calls on the DiffWindow to sample the selected page and waits for a callback. If not, the Dispatcher tries again 10 seconds later.

In order to handle sampling requests the DiffWindow calls out to WebPageDump (WPD) [16], which is embedded inside the CourseDiff extension. The DiffWindow first creates an invisible internal frame (iframe) element and adds this frame to its DOM. The frame is then set to load the requested page. When the page loads, WPD is used to store the page to disk.

Samples are stored in a 'samples' folder in the user's Firefox profile directory. Every page has a numbered subfolder in 'samples' corresponding to its `id` in the database. The current version of the page is stored in a subfolder of that ('cur'), as is the previous version ('prev'). When the DiffWindow samples a page, if there is a current version of that page it is moved over ('cur' becomes 'prev'). The page that is loaded in the iframe is then saved by WPD as the current version ('cur').

WPD saves a 'flattened' version of the page. This means that all images and other file resources used in the page are downloaded to a single folder. Any references to those resources in the HTML and CSS style files are rewritten to point to the files in that folder. Finally, the main HTML file is renamed to 'index.html'. The result of this process is that the webpage is contained entirely in the folder and can be viewed by loading 'index.html'. The page does not need any resources from the web and can be moved around without worrying about breaking it.

Identifying Changes

After a page is sampled, the DiffWindow compares successive versions of the page. If there is no previous sample ('prev' does not exist), then this cannot be done, and control is given back to the Dispatcher. However if the two samples both exist, the DiffWindow performs the following steps:

1. **Create Invisible IFrames**. Three invisible iframes are created. The first loads the current sample of the page, 'cur/index.html'. The second loads the previous sample of the page, 'prev/index.html'. The third loads 'prev/index-diff.html', a decorated version of the previous sample that was created the last time this page was checked for changes. This version has update count information needed for step 3. If this is only the second time the page has been sampled, this version will not exist.

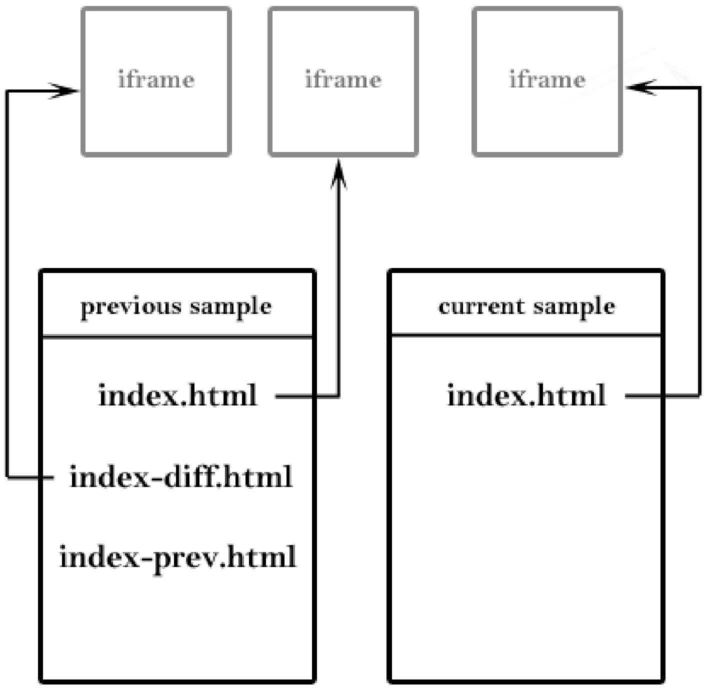


Figure 13 Step 1 – load samples into iframes

2. **Run X-Tree Diff**. Once the frames load, a Javascript implementation of X-Tree Diff is run on the body elements of the two 'index.html' DOMs. The algorithm decorates the DOM with attributes *xt:index*, *xt:pointer* and *xt:op*. Since text nodes in the DOM cannot have attributes, this algorithm replaces text nodes with 'xt:text' nodes. The 'xt:text' node is decorated with the appropriate attributes and the removed text node is made its child. This algorithm also goes a step further and does a word-by-word String comparison on the text of updated (UPD) text nodes using the method described in [2]. Text that has changed as a result of the update is wrapped in an 'xt:textupd' node.

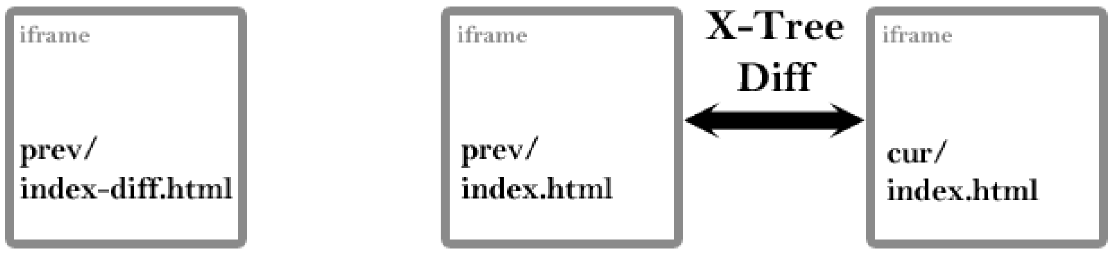


Figure 14 Step 2 – X-Tree Diff the current sample against the previous

3. **Transfer Update Counts**. If the previously decorated sample, 'prev/index-diff.html', exists, then it may contain update counts. An update count is another tree decoration (the attribute *xt:updatecount*). It indicates how many samples in a row a node has updated or had one of its children updated or added. If a node has this happen again in the current sample then its update count is incremented. If not, the update count drops back to zero. Once the update count hits a critical threshold (3) it stays there and the node is then considered a source of every-sample changes. The update count is first transferred from the DOM of the initially decorated previous sample, 'prev/index-diff.html', to the previous sample DOM that was just decorated, 'prev/index.html'. The two have matching *xt:index* attributes on their nodes and are mapped using these. Next, the update counts are transferred over to the current DOM, 'cur/index.html', using the *xt:pointer* attribute generated by the X-Tree Diff. After that, the update count of each node is incremented or reset depending on whether or not it has changed.

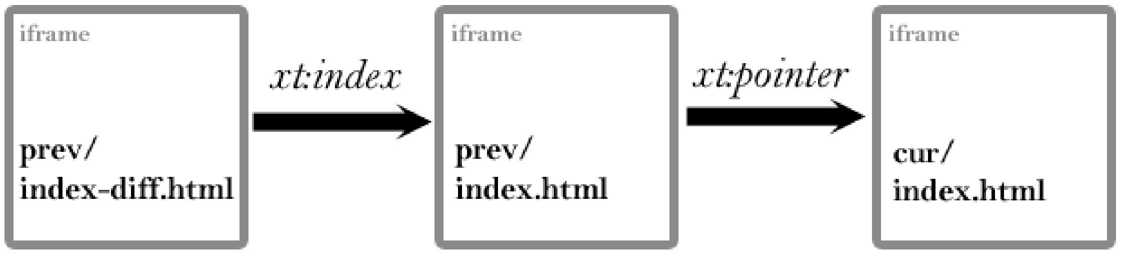


Figure 15 Step 3 – transfer update counts to the current sample

4. **Extract NW-Textual Content**. Now that all of the nodes in the current DOM are decorated with update counts, every-sample changes can be ignored by ignoring the nodes whose counts are at the threshold. To get the nw-textual content of each sample, its DOM is traversed. Every-sample nodes are ignored. The content of text nodes (and textual attributes) is stripped entirely of whitespace and concatenated into a single string. If the

string generated from the previous sample is not equal to the string generated from the current sample, then the change is considered important.

5. **Upload the Change.** If the DiffWindow determines that no important change has occurred, then the two newly decorated DOMs are rendered into HTML and saved to the 'cur' folder. (The saved decorated DOM of the current sample will function as the initially decorated previous sample when the page is sampled the next time.) If the change is important, however, the DOMs are traversed and styled to highlight the changes (see 6.4 Change Viewer) before being saved. Then the DOM is traversed again to generate the email report (see 6.3 Change Report), which is also saved to the 'cur' folder as 'digest.html'. After that, the entire contents of the 'cur' folder are uploaded to the server along with the user-specified email address.

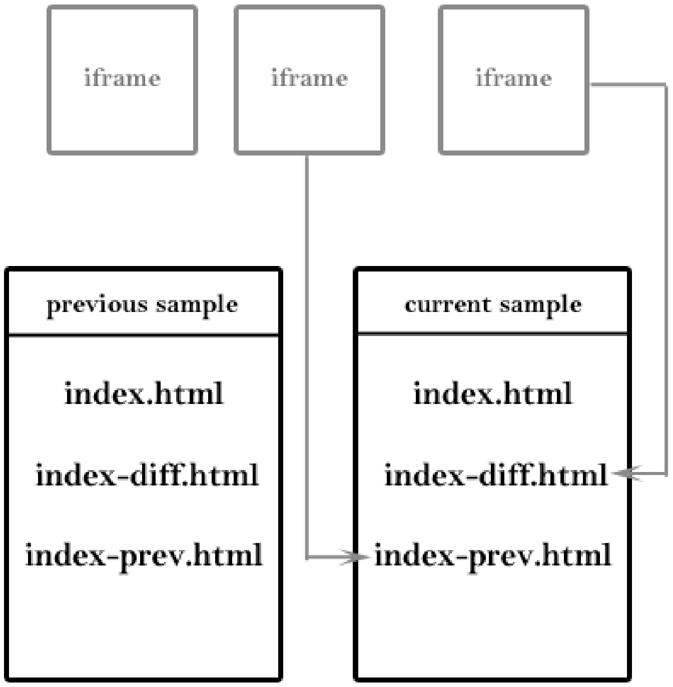


Figure 16 Step 5 – write decorated DOMs back to file

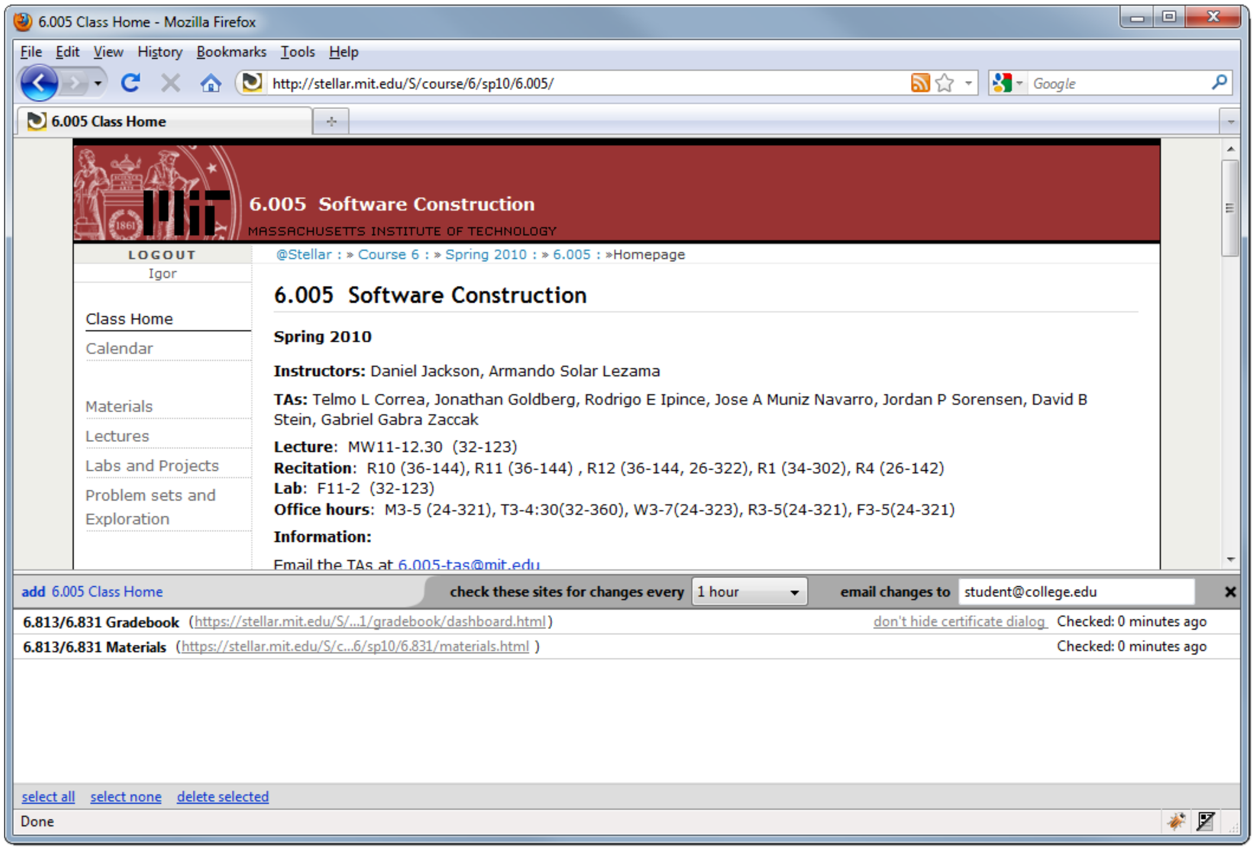
5.2 Server

The CourseDiff server is responsible for storing each uploaded change and emailing a report to the user. The server works very simply. When it receives an upload request, it creates a new folder, named by applying a hash to a random number, and stores all of the uploaded files to it. It looks for 'digest.html' and sends its contents to the email address it receives as part of the request. At the bottom of the email, it appends a link to a change-viewer page that references the generated folder. The change-viewer page takes the folder name as a URL parameter and loads the highlighted versions of the changed pages in that folder. Deletions can be seen by looking at the previous page and additions/updates by looking at the current.

Notably, the server does not keep track of where any particular site is stored. It just gives out space for any uploaded change. The organization of changes happens in the user's email client. Whether the user wants to hold on to those changes for future reference, or delete them from their inbox is up to them. Once the email is deleted, however, there is no way to recover which folder contains any given change. This information is not stored in the server or in the browser extension. This policy might be problematic in some cases, but, in general, since change reports are intended to be timely notifications, not records of a page's history, users are not particularly likely to miss them if they are deleted.

6. User Interface Design

6.1 Configuration Panel


6.005 Class Home - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://stellar.mit.edu/S/course/6/sp10/6.005/
Google
6.005 Class Home
LOGOUT
Igor
Class Home
Calendar
Materials
Lectures
Labs and Projects
Problem sets and Exploration
select all
select none
delete selected
Done

6.005 Software Construction
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

@Stellar : » Course 6 : » Spring 2010 : » 6.005 : »Homepage

6.005 Software Construction

Spring 2010

Instructors: Daniel Jackson, Armando Solar Lezama

TAs: Telmo L Correa, Jonathan Goldberg, Rodrigo E Ipince, Jose A Muniz Navarro, Jordan P Sorensen, David B Stein, Gabriel Gabra Zaccak

Lecture: MW11-12.30 (32-123)
Recitation: R10 (36-144), R11 (36-144) , R12 (36-144, 26-322), R1 (34-302), R4 (26-142)
Lab: F11-2 (32-123)
Office hours: M3-5 (24-321), T3-4:30(32-360), W3-7(24-323), R3-5(24-321), F3-5(24-321)
Information:
Email the TAs at 6.005-tas@mit.edu

add 6.005 Class Home | check these sites for changes every 1 hour | email changes to student@college.edu

6.813/6.831 Gradebook (https://stellar.mit.edu/S/...1/gradebook/dashboard.html)	don't hide certificate dialog	Checked: 0 minutes ago
6.813/6.831 Materials (https://stellar.mit.edu/S/c...6/sp10/6.831/materials.html)		Checked: 0 minutes ago

Figure 17 The CourseDiff configuration panel

The main goal of the configuration panel is to give users quick access to all of CourseDiff's settings. The panel can be opened by pressing the CourseDiff icon in the bottom left corner of the browser or by selecting CourseDiff from the tools menu. The panel can be closed by repeating either of the previous two actions or by clicking the 'x' icon in the panel's top right corner. Also, the panel can be resized by dragging the splitter separating the panel from the page content.

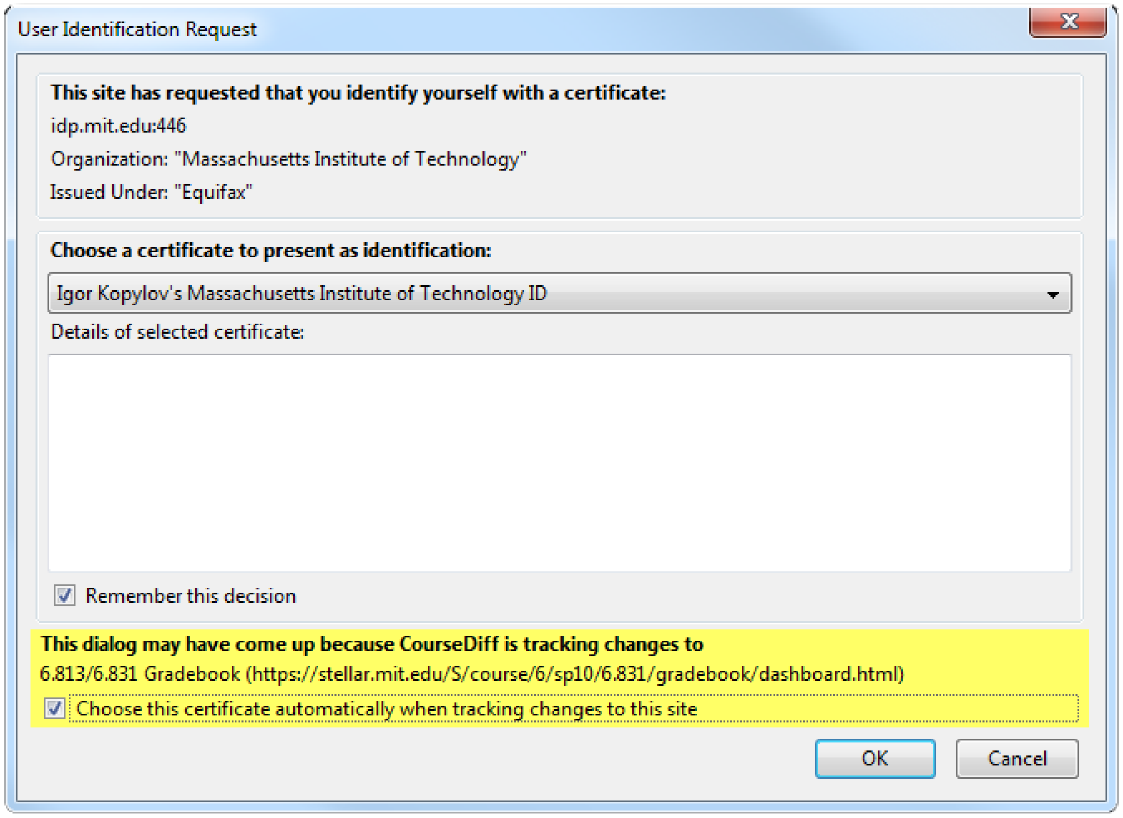
Pages are added for tracking by clicking the link in the top left corner of the panel. Only the currently displayed site can be added. Though this does constrain the user somewhat, this constraint is motivated by error prevention. If users were allowed to enter an arbitrary URL and start tracking it, they may not realize that they entered an incorrect URL until the system misses an important change. By forcing the user to actually see the page that they add, this interface helps ensure that the user is tracking the right one.

A page can be removed from tracking by selecting it from the list and then pressing the 'delete' link at the bottom of the panel. The list allows multiple selection, and all of the pages can be selected at once by clicking the 'select all' link. Pages can be renamed by double clicking on the item in the list. This turns the site name into a textbox. The user can change the name and then press enter or change focus to confirm, or press the escape key to cancel. Rename and delete are also available from a right-click context menu.

The period with which to check for changes can be set by choosing an option from the dropdown at the top of the panel. The options are 30 minutes, 1hour, 2 hours, 5 hours and 10 hours. The email address to which to send changes can be set by filling in the textbox in the top right of the panel. While editing, the box background turns yellow and then becomes white again when the email address is confirmed, either by pressing the enter key or by moving focus to another part of the interface.

If a page requires certificates and CourseDiff was configured to close certificate dialogs automatically when sampling that page, then a 'don't hide certificate dialog' link appears for the item. Clicking it will allow the dialog to appear as usual the next time the page is sampled.

6.2 Certificate Dialog



User Identification Request

This site has requested that you identify yourself with a certificate:
idp.mit.edu:446
Organization: "Massachusetts Institute of Technology"
Issued Under: "Equifax"

Choose a certificate to present as identification:
Igor Kopylov's Massachusetts Institute of Technology ID
Details of selected certificate:

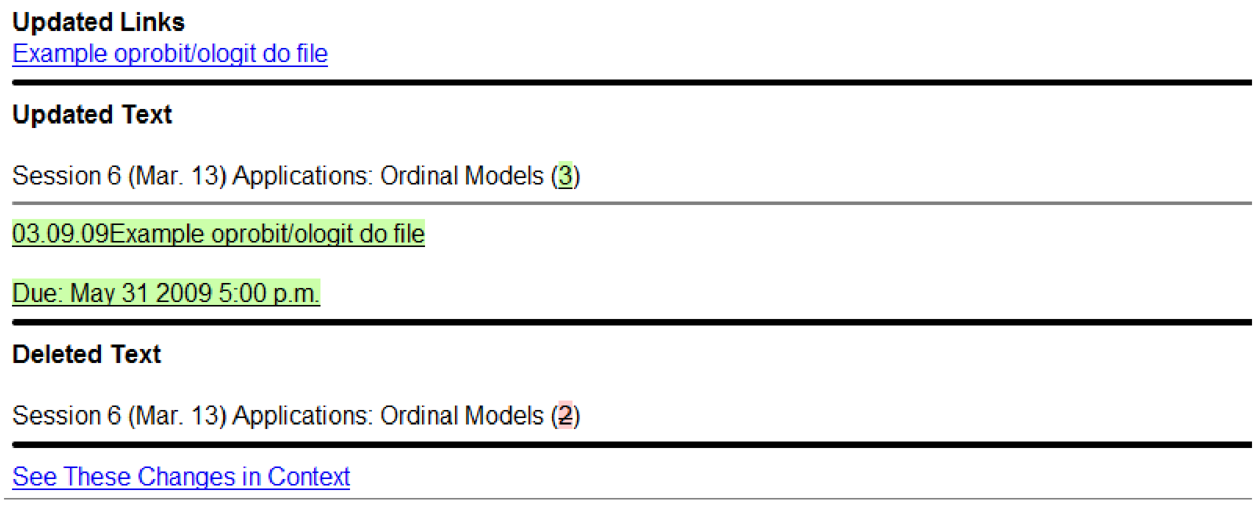
☑ Remember this decision

This dialog may have come up because CourseDiff is tracking changes to
6.813/6.831 Gradebook (https://stellar.mit.edu/S/course/6/sp10/6.831/gradebook/dashboard.html)
☑ Choose this certificate automatically when tracking changes to this site

Figure 18 The Firefox certificate dialog with CourseDiff notification

The certificate dialog is a standard dialog in Firefox. It appears whenever a site requests a user to identify themselves with certificates. When the dialog opens during the sampling of a page, CourseDiff adds a notification to it, explaining why the dialog opened. Without this warning, the dialog would seem to appear for no reason. That is, the user would likely not be trying to access a certificate protected site, and yet the dialog would appear. The notification also provides the option of automatically using the choice the user makes this time on all future attempts to sample this page. If the user selects this option, then their certificate choice will be stored in the database and chosen automatically the next time.

6.3 Change Report



Updated Links
Example oprobit/ologit do file

Updated Text

Session 6 (Mar. 13) Applications: Ordinal Models (3)

03.09.09Example oprobit/ologit do file

Due: May 31 2009 5:00 p.m.

Deleted Text

Session 6 (Mar. 13) Applications: Ordinal Models (~~2~~)

See These Changes in Context

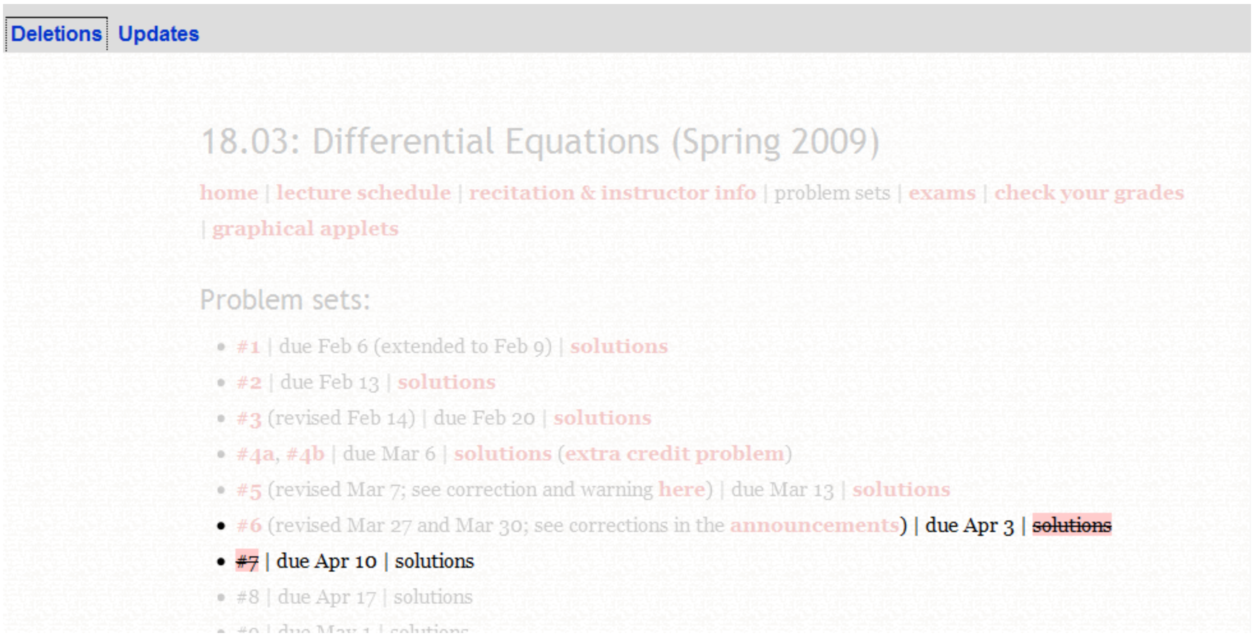
Figure 19 A change report

The change report extracts just the changed textual data from a sample in a format suitable for email. The change report has four sections: Updated Links, Updated Text, Deleted Links and Deleted Text. Each section is only displayed if it has content. Both updated and added text is displayed in the Updated Text section. The text that was actually added is highlighted in green and underlined. Similarly, both updated and deleted text is displayed in the Deleted Text section, but the text that was actually deleted is highlighted in red and crossed out.

Links are separated out from the textual content because links are often the focal point of a change. An added link might be a new problem set, an article or a test solution. Separating them out can, in some cases, allow the user to ignore the rest of the textual content of the change and just focus on whatever document the change was meant to direct them towards.

6.4 Change Viewer

In some cases, textual changes may not make sense on their own and would be better viewed in the context of the entire page. In case this is necessary, the change report provides a link at the bottom that takes the user to the change viewer. The change viewer has two tabs: Deletions and Updates. The highlighting and colors parallel the change report, but the text is shown in context. The change viewer makes all unchanged parts of the website semitransparent so that changes pop out of the page.



18.03: Differential Equations (Spring 2009)

home | lecture schedule | recitation & instructor info | problem sets | exams | check your grades | graphical applets

Problem sets:

- #1 | due Feb 6 (extended to Feb 9) | solutions
- #2 | due Feb 13 | solutions
- #3 (revised Feb 14) | due Feb 20 | solutions
- #4a, #4b | due Mar 6 | solutions (extra credit problem)
- #5 (revised Mar 7; see correction and warning here) | due Mar 13 | solutions
- #6 (revised Mar 27 and Mar 30; see corrections in the announcements) | due Apr 3 | ~~solutions~~
- ~~#7~~ | due Apr 10 | solutions
- #8 | due Apr 17 | solutions

Figure 20 A change view showing the deletions tab

This view is very similar to the approach taken by DiffIE. However in DiffIE, identifying changes plays a secondary role to being able to view and use the visited page. The change viewer's priorities are exactly reversed, so it is able to make more drastic graphic design decisions in order to make the changes stand out. Also, the change viewer is able to show both updates and deletions by providing two tabs. Since DiffIE was designed to be unobtrusive, such extra features would have been inappropriate. It is important to note, however, that although the change viewer is a nice fallback, in the ideal case, the change report would always provide the necessary context, and users would never have to break their email workflow to understand a change.

7. User Interface Evaluation

7.1 Configuration Panel

The configuration panel was presented to 4 MIT undergraduates, preceded by a brief explanation of CourseDiff. The participants were asked to add a few of their own course webpages to the list, rename the sites to their liking, and choose an email address to send the changed to. After that the users were asked to delete all of pages from the list.

All four users were able to figure out that adding a page required visiting it. Two of the users were unsure about this at first, but figured it out within seconds. The other two immediately began navigating to their course websites seeing that as a natural way to add them. Users were able rename the pages without trouble, though two of them did express a concern that they would not have discovered this feature if it had not been explicitly mentioned. All users were able to delete pages from the list without trouble. Three used the 'select all' command followed by delete. One used the context menu to delete each item individually.

When entering the email address one user was unclear of whether the change had taken effect. She indicated that an explicit button to confirm setting the address would make this clearer. Another user was unsure of whether the email address applied to all of the sites or only the currently selected one. She expressed interest in sending certain updates to one address and certain updates to another.

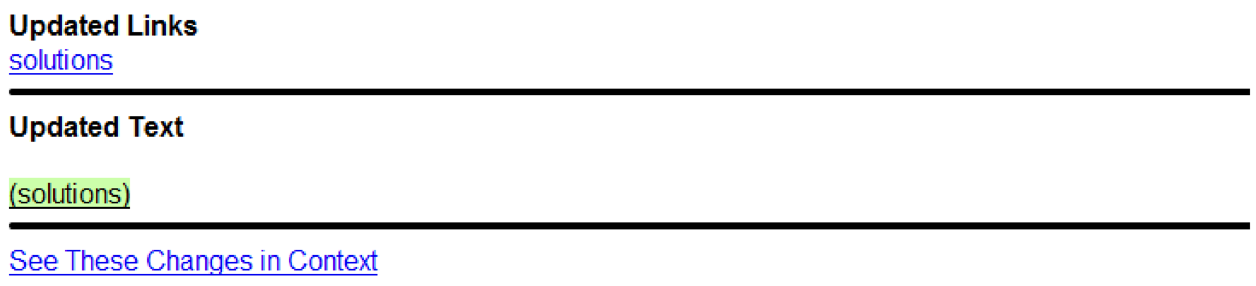
Though they were not explicitly instructed to, two of the participants modified the sampling period. One expressed a desire for longer periods than 10 hours. When asked for the reason, the participant expressed some concern over receiving spurious updates from sites that she knows change rarely. Part of that concern was also the thought that checking for changes too often might use up too many of the browser's or computer's resources. The other participant asked if the sampling could be temporarily disabled (an infinite period) in case, for example, the machine has a poor or no connection to the web.

All four participants indicated that they would be interested in using a system like CourseDiff to receive updates via email. None of them said that they use RSS as part of their routine.

7.2 Change Report

Because textual and important changes are so closely related, the change report is able to adequately represent changes in the DOM despite being almost purely textual. However, there are two problem cases in which the change report fails to adequately capture the change it represents.

Text Out of Context


Updated Links

solutions

Updated Text

(solutions)

See These Changes in Context

Figure 21 Change report showing out of context text

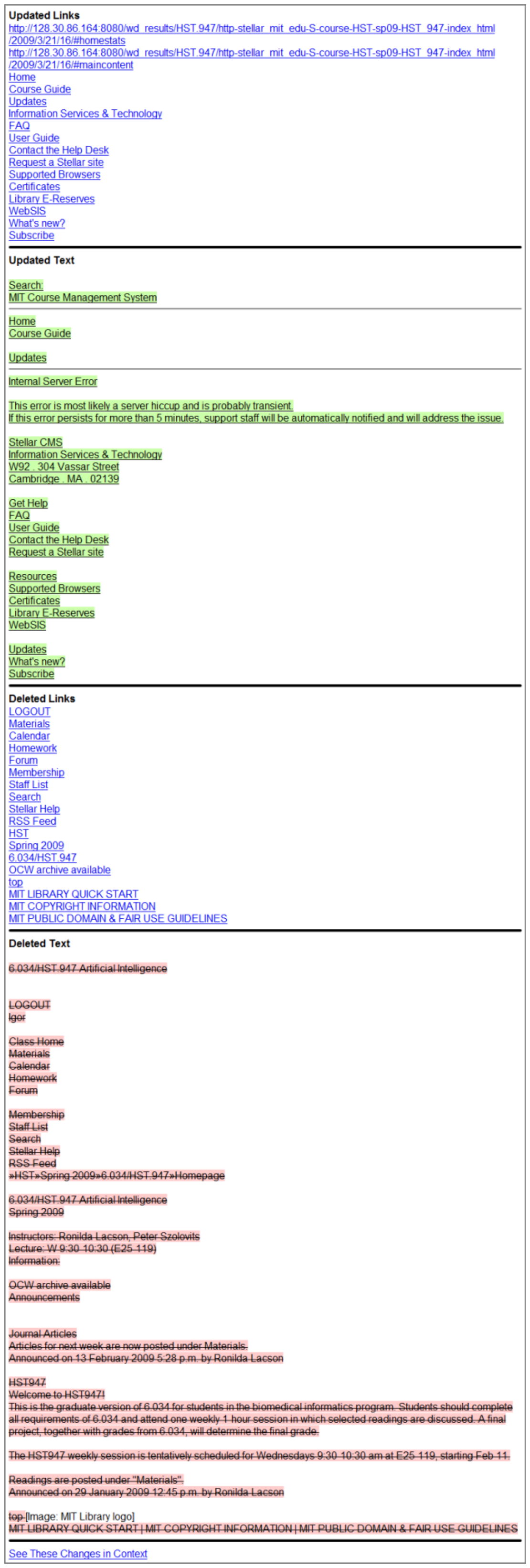
Some textual changes need the context of surrounding text to mean something to the user. If a "solutions" link was added to the document, its placement next to a given problem set or test would indicate what the solutions are for. Unfortunately, the change report does not always capture this information. The change viewer can help provide context, but in this case it would be faster to just click on the link and see what the solutions are for. It is also possible that the user would know what they are for, depending on what assignments have gone out recently.

This sort of additional knowledge cannot help out-of-context changes in general, however. So, change reports could certainly benefit from grabbing some extra context.

Site Error

Another situation in which a change report fails to provide a good representation of the underlying change is when a site responds with an error. This can mean that the entire DOM is replaced. That is, every node in the previous sample is deleted and every node in the new sample is added. This creates an extremely large change report that says very little. Even worse, when the site recovers to its original state, the opposite change will also be reported.

One possible way to deal with this issue is to store a longer history of samples. If the current sample is very different from the previous one, then the next sample after that can be compared against both to see if the site just had a momentary problem or if it has indeed changed dramatically. A cheaper solution might be to simply throw away changes that are too drastic. This approach, however, would never



Updated Links
http://128.30.86.164:8080/wd_results/HST.947/http-stellar_mit_edu-S-course-HST-sp09-HST_947-index_html/2009/3/21/16/#homestats
http://128.30.86.164:8080/wd_results/HST.947/http-stellar_mit_edu-S-course-HST-sp09-HST_947-index_html/2009/3/21/16/#maincontent
Home
Course Guide
Updates
Information Services & Technology
FAQ
User Guide
Contact the Help Desk
Request a Stellar site
Supported Browsers
Certificates
Library E-Reserves
WebSIS
What's new?
Subscribe

Updated Text

Search:
MIT Course Management System

Home
Course Guide

Updates

Internal Server Error

This error is most likely a server hiccup and is probably transient.
If this error persists for more than 5 minutes, support staff will be automatically notified and will address the issue.

Stellar CMS
Information Services & Technology
W92 . 304 Vassar Street
Cambridge . MA . 02139

Get Help
FAQ
User Guide
Contact the Help Desk
Request a Stellar site

Resources
Supported Browsers
Certificates
Library E-Reserves
WebSIS

Updates
What's new?
Subscribe

Deleted Links
LOGOUT
Materials
Calendar
Homework
Forum
Membership
Staff List
Search
Stellar Help
RSS Feed
HST
Spring 2009
6.034/HST.947
OCW archive available
top
MIT LIBRARY QUICK START
MIT COPYRIGHT INFORMATION
MIT PUBLIC DOMAIN & FAIR USE GUIDELINES

Deleted Text

~~6.034/HST.947 Artificial Intelligence~~

~~LOGOUT~~
~~Igor~~

~~Class Home~~
~~Materials~~
~~Calendar~~
~~Homework~~
~~Forum~~

~~Membership~~
~~Staff List~~
~~Search~~
~~Stellar Help~~
~~RSS Feed~~
~~»HST»Spring 2009»6.034/HST.947»Homepage~~

~~6.034/HST.947 Artificial Intelligence~~
~~Spring 2009~~

~~Instructors: Ronilda Lacson, Peter Szolovits~~
~~Lecture: W 9:30-10:30 (E25-119)~~
~~Information:~~

~~OCW archive available~~
~~Announcements~~

~~Journal Articles~~
~~Articles for next week are now posted under Materials.~~
~~Announced on 13 February 2009 5:28 p.m. by Ronilda Lacson~~

~~HST947~~
~~Welcome to HST947!~~
~~This is the graduate version of 6.034 for students in the biomedical informatics program. Students should complete all requirements of 6.034 and attend one weekly 1-hour session in which selected readings are discussed. A final project, together with grades from 6.034, will determine the final grade.~~

~~The HST947 weekly session is tentatively scheduled for Wednesdays 9:30-10:30 am at E25-119, starting Feb 11.~~

~~Readings are posted under "Materials".~~
~~Announced on 29 January 2009 12:45 p.m. by Ronilda Lacson~~

~~top~~ [Image: MIT Library logo]
~~MIT LIBRARY QUICK START | MIT COPYRIGHT INFORMATION | MIT PUBLIC DOMAIN & FAIR USE GUIDELINES~~

See These Changes in Context

Figure 22 Change report for an error page sample

recognize that a site has legitimately had a dramatic change.

In principle, a site going down could very well be an important change. If the course staff needs to act to remedy the situation, then the sooner the change is identified the better. So perhaps a better solution would be for the report to say something more meaningful instead of presenting the enormous change.

8. CONCLUSION

CourseDiff is a prototype system for identifying and reporting changes to course websites. CourseDiff's change identification system was designed based on a study of 120 course webpages sampled during the spring semester of 2009. This study found that although many of the changes to course webpages are trivial, there are two main causes of trivial changes: 1.) automatically-generated content that changes every time a page is visited and 2.) formatting and whitespace changes that do not affect the underlying textual content. These two types of trivial change combined make up over 99% of the trivial changes found in the corpus.

CourseDiff is designed to filter out both types of trivial changes from the DOM of sampled pages. When CourseDiff detects a change that passes through the filter, it generates a change report and sends it to the user as an email. In many cases, the change report is a good indication of how the site changed. When it is not, the user has the option of seeing the reported changes in the context of the original page. A small user study of CourseDiff's browser-embedded configuration panel showed that users can comfortably add and then manage pages for tracking.

This work makes the following contributions:

- Collection of a large (> 100,000 sample) corpus of course webpages.
- Identification of two key sources of trivial changes in the collected corpus.
- A method for filtering out the majority of the trivial changes.
- The development of a tool for reporting non-trivial changes.

9. Future Work

One clear direction of future work is to gauge CourseDiff's performance on a new corpus of course websites. One good way to do this would be to deploy the extension to some pilot users and then have them rate each change report as it comes in. It would also be interesting to see how having a change tracker would affect the users' browsing habits. For example, if the users become confident in CourseDiff they might stop visiting the actual site entirely. On the other hand, if they have doubts, they might continue to visit the page and end up seeing changes both on the site and in their email.

One other possibility is to expand the role of the server to allow users to share changes, so that they do not have to necessarily scrape pages themselves. In other words, before scraping a page CourseDiff could check to see if someone else had already sampled that page recently. Furthermore, when a change is identified, the server could notify all of the users that are tracking changes to that page. Such an approach would put less stress on the course websites since all of the users would not have to sample the site regularly.

Another possible direction is to experiment with different types of change reports. Perhaps, more than textual content could be included in the main report. Images might be added to it, as well as layout components like tables. Leaving some of these elements in might preserve more of the original look of the site, but it may also make for an awkward email. Even trickier, if those elements' style is not preserved, then the generated content may not even look like the original site content. Finding a good way to translate more of the structure of the original site into an email message could be a very interesting area of research.

A final possible direction is to see how CourseDiff's methods might be applied to webpages in other domains. Perhaps there are other classes of webpages with the same sources of trivial changes. Equally important, there may be classes of webpages that have entirely different sources of trivial changes. Determining which kinds of sites change in similar ways and which kinds do not could lead to new ways to think about how the web evolves over time.

[1] R. A. Wagner and M. J. Fischer, "The String-to-String Correlation Problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168-173, Jan. 1974.

[2] P. Heckel, "A Technique for Isolating Differences Between Files," *Communications of the ACM*, vol. 21, no. 4, pp. 264-268, Apr. 1978.

[3] W. Miller and E. W. Myers, "A File Comparison Program," *Software – Practice and Experience*, vol. 15, pp. 1025-1040, Nov. 1985.

[4] K. Tai, "The Tree-to-Tree Correction Problem", *Journal of the ACM*, vol.26, no.3, pp. 422-433, July 1979.

[5] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol.337, no.1-3, pp. 217-239, June 2005.

[6] D. A. Kim and S. K. Lee, "Efficient Change Detection in Tree-Structured Data," in *Web Communication Technologies and Internet-Related Social Issues - HSI 2003, Second International Conference on Human Society@Internet*, 2003, pp. 675-681.

[7] S. Rönnau, G. Philipp, U. M. Borghoff, "Efficient change control of XML documents", in *Proceedings of the 9th ACM symposium on Document engineering*, 2009, pp. 3-12.

[8] S. Greenberg and M. Boyle, "Generating custom notification histories by tracking visual differences between web page visits," in *Proceedings of Graphics interface 2006*, 2006, pp. 227-234.

[9] J. Teevan, S. T. Dumais, D. J. Liebling and R. L. Hughes, "Changing how people view changes on the web," in *Proceedings of the 22nd Annual ACM Symposium on User interface Software and Technology*, 2009, pp. 237-246.

[10] E. Adar, M. Dontcheva, J. Fogarty, and D. S. Weld, "Zoetrope: interacting with the ephemeral web," in *Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology*, 2008, pp. 239-248.

[11] RSS Advisory Board, "RSS 2.0 Specification" *RSS Advisory Board.* [Online]. Available: http://www.rssboard.org/rss-specification#whatIsRss

[12] R. J. Glotzbach, J. L. Mohler and J. E. Radwan, "RSS as a course information delivery method," in *ACM SIGGRAPH 2007 Educators Program.* SIGGRAPH '07.

[13] H.Liu, V. Ramasubramanian and E. G. Sirer, "Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews," in *Proceedings of the 5th ACM SIGCOMM Conference on internet Measurement.* Internet Measurement Conference.

[14] "MIT Course Management System," [Online]. Available: http://stellar.mit.edu/.

[15] "About course.mit.edu," [Online]. Available: http://course.mit.edu/about.html.

[16] B. Pollak and W. Gatterbauer, "Creating Permanent Test Collections of Web Pages for Information Extraction Research," in *Proceedings of SOFSEM 2007: Theory and Practice of Computer Science*, 2007, pp. 103-115.

[17] Campaign Monitor, “Guide to CSS support in email clients,” *Campaign Monitor*. [Online]. Available: http://www.campaignmonitor.com/css/