MIDDLESEX Community College

Tools and Technologies for Tech Writers 2024

Week 5

Agile (and HTML)

Notices

This document was prepared as a handout for the Middlesex Community College Tools and Technologies for Technical Writers class, Winter semester 2024.

Prepared by Zoë Lawson, course instructor.

Contents

Agile introduction

The Agile development process is meant to be adaptive and responsive to customer needs.

I've also heard agile referred to as "Developers gone wild".

When the tenants are followed, an agile development process can be a productive way to develop software.

The learning curve to really understand how to plan and work within an agile process so that you don't try to do too much at once can take a while and be a bit bumpy.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

***Individuals and interactions** over processes and tools*

***Working software** over comprehensive documentation*

***Customer collaboration** over contract negotiation*

***Responding to change** over following a plan*

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	*James Grenning*	*Robert C. Martin*
Mike Beedle	*Jim Highsmith*	*Steve Mellor*
Arie van Bennekum	*Andrew Hunt*	*Ken Schwaber*
Alistair Cockburn	*Ron Jeffries*	*Jeff Sutherland*
Ward Cunningham	*Jon Kern*	*Dave Thomas*
Martin Fowler	*Brian Marick*	

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.

From https://agilemanifesto.org/.

Agile Resources

When you get to a workplace, look at the Agile resources they used, so that you understand how they are implementing Agile. While the concepts are similar, implementations can be very, very different.

Agile 101	https://www.agilealliance.org/agile101/

	One of the many web sites dedicated to Agile methodology.
Agile Coach	https://www.atlassian.com/agile This guide is provided by Atlassian, the company that makes JIRA. The JIRA ticketing system has been designed around Agile development, so they want you to understand Agile and succeed with their software.
IBM Design Thinking	https://www.ibm.com/design/thinking/page/framework The Keys section gives an overview of the concept of hills.

Agile process

One method of following the Agile Development method. Again, my summary based on experience.

1. Product Managers meet with customers and understand what they need.
2. Product Managers write user stories.

   These are called *epics*, *hills*, or *wedding cakes*.
3. Product Managers decide which epics to work on when and assign epics to a scrum team.
4. Scrum team sits with Product Manager and designs a software solution to the user story.
5. The epic is broken down into *stories* (or *cupcakes*).

   A story should be a smaller, understandable unit of work that should be completable within a *sprint*. You assign each story *story points* to identify how much work you think it will take to do.

   Sometimes you need to break stories into even smaller stories.

   All of these stories are added to the *backlog* of tasks to do.

   Steps 4 and 5 can take a full sprint to complete.
6. At the start of each sprint, you pick some number of stories to work on. For each story, you break it down into specific tasks, and assign time estimates to each task.
7. If you complete the work within the sprint, close the task. If you are concerned you won't complete the work within the sprint, raise it as a concern in scrum.

   You can split tasks to continue working on them in the next sprint.
8. When all tasks are closed (or moved to the next sprint), you can close the sprint.
9. After the sprint is closed, perform a retrospective to determine what you might need to do better next sprint.
10. Repeat steps 6 through 9 until you complete all the stories in the epic.

Once an epic is complete, you can release the software. That may be actually releasing to customers, or "releasing it to main" to be handed to customers when the correct number of items are in main.

The goal of a sprint is to complete all the work assigned to the sprint in the sprint. However, that doesn't always happen, and that's to be expected.

You hold a retrospective to figure out why you may not have completed all the work in the sprint. It could be something outside of your control, such as internal testing servers being down. It could be a major design flaw that needed to be reworked. It could be that you needed to learn how to estimate better.

Agile is meant to be iterative. You learn how your team works, you learn what goes right, you learn what goes wrong, and you work to fix it.

Agile terms

Some of the terms you hear about with Agile development. These are my definitions based on experience and web searches.

Backlog	A list of stories, tasks, and bugs that are incomplete.
Blocker	Some problem that cannot be fixed by whomever it's assigned to. You may need a few minutes of help, or a component may need to be redesigned, or you may need IT help to make a server start working again.
Epic	A large task defined by a user story. The task is eventually broken down into stories.
Grooming	Reviewing an epic and designing a solution. This includes breaking down the epic into stories.
Iteration	The unit of time you try to complete your work in. Usually 1 to 3 weeks long. Another term for Sprint.
Research spike	A task to look into something, where you schedule time to do research. This can include researching different tools, new software, speaking with experts, etc.
Scrum master	The person who leads scrum meetings. They are supposed to keep the meeting on task, and make sure you're following whatever agile processes you're using.
Scrum meeting	Also known as a stand-up. This is a short daily meeting. Each member of the team should list what they did yesterday, what they plan to do today, and if there are any blockers. If there needs to be additional conversations about design flaws, major bugs, etc, those discussions should be scheduled outside of the scrum meeting.
Scrum team	A group of people who work together. Usually consists of developers, QA, a product manager, and a tech writer. If you're very lucky, you have a trained scrum master or project manager. Often the scrum master is just one of the team members.
Sprint	The unit of time you try to complete your work in. Usually 1 to 3 weeks long. Another term for Iteration.
Sprint retrospective	A review of the sprint that just closed where you determine what went well, what didn't go well, and what you can do to improve.
Story	A component of an epic that needs to be completed. Stories are usually broken down into tasks. Epics are made up of stories. A story might be

	"Add widget button to sprocket GUI," "Write API to create widget object in sprocket," or "Make widget button call createWidgetObject API".
Story points	A numeric representation of how long it takes to complete a story. It is not an exact time, it's more along the lines of "small," "medium," or "large." See https://en.wikipedia.org/wiki/Planning_poker.
Task	A thing that needs to be completed. Stories are made up of tasks. A task might be "Dev - Add widget button," "QA - Write test plans for widget button," or "QA - test widget button"
User story	A description of some task a user wants to complete using the software to be developed.

HTML overview

This is brief because there are thousands of HTML how-tos on the internet.

HTML is a markup or tagging language.

HTML tags are marked by angle brackets. < starts a tag and > ends the demarcation of the tag. You must include the trailing >.

HTML elements have a start tag and a closing or end tag. You open a tag with `<tagname>` and close it with `</tagname>`. Your content goes inside of the tags. For example:

```
<tagname>Your content goes here.</tagname>
```

Your tags can be nested, meaning you can put tags inside of other tags. There are many rules about which tags need to go where. For example, `<li>` elements must always be inside of an `<ul>` or `<ol>` element.

If your tag does not contain any content, you can create a *singleton* by including the / before the closing >, such as `<tagname />`

Technically, in some older forms of HTML you didn't *have* to close elements. That is a horrible practice. Even if it's allowed, don't do it. Browsers then need to guess at what to do. They will often guess wrong, and debugging the issue becomes impossible. While it may seem to be a bit harder, you really should make sure you open and close elements properly.

Elements may contain additional information in the form of *attributes*. Attributes must be inside the angle brackets of the tag and take the format `attributename="attribute value"`. Attributes must come after the tag name. For example:

```
<tagname attributename="attribute value"/>
```

Attributes contain a lot of additional information:

- To help the tag work, such as for a tag that includes an image or video.
- Information to help the content display, such as the `@class` attribute to reference styles in a cascading style sheet.
- Information that can be used by other things that use the HTML, such as the `@id` attribute.
- References to scripting languages that make HTML do more than just display text and pictures, such as the `@onclick` attribute that contains JavaScript information.

There are three reserved characters in HTML, <, >, and &. If you want to use these characters in an HTML file, you have to use their *entity*. An entity is a special text string that represents a character. See https://www.w3schools.com/html/html_entities.asp.

Character	Entity
<	&lt;
>	&gt;
&	&amp;
"	&quot;

The " is not always reserved. Generally, it's okay to use in the content part of tags, but it can sometimes be problematic. There are many other entities used, such as `&nbsp;` for a Non-Breaking SPace. It's worth learning a few to have better luck including special characters such as ® and ™.

The following is a sample HTML5 page, with all required elements.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title in the browser tab</title>
    </head>
    <body>
        <h1>Heading 1 Title in Topic</h1>
        <p>This is a paragraph</p>
    </body>
</html>
```

The `<head>` tag contains all sorts of stuff the browser needs to display and interact with the HTML file. This is where you link to a Cascading Style Sheet (CSS) to control the display of the content. You can also reference other files and languages to make the HTML page do things, such as JavaScript files. Metadata that helps with search results can also go here. Minimally, you have to have a `<title>` element. That is what displays in the tab of the browser window.

The `<body>` tag contains what actually shows up in the browser. This is where the majority of your content goes. Most of the time, you're writing in this area.

Content tags have two major types, *block* tags and *inline* tags. Blocks make up structural items that make the layout work, and major chunks of contents. The basic tag of `<p>`, for paragraphs, is a block tag. Inline tags are for items that appear in the line of text. The basic tag of `<b>`, for bold, is an inline tag. There are some tags that can be both, such as `<img>`, for image, which sometimes you want to stand alone (such as for a diagram), or sometimes to be inline (such as for an icon).

Some of the most common block elements:

- `<p>` — Paragraph
- `<ul>` — Unordered list, i.e. bullets. Must contain an `<li>`.
- `<ol>` — Ordered list, i.e. a numbered list. Must contain an `<li>`.
- `<li>` — List item. These are bullets if a child of an `<ul>`. These are numbered items if a child of an `<ol>` element.
- `<h1>`, `<h2>`, `<h3>` — Headings of various levels. Goes up to `<h6>`, but formatting often gets weird after `<h3>`.
- `<img>` — Images. Must contain a `@src` (source) attribute with the path to the image file to display.
- `<div>` — a generic division. Often used for formatting purposes.

The `<table>` element is also a block element, but it's easier to show example simple table markup.

```
<table>
  <tr>
    <th>Column 1 Table Head</th>
    <th>Column 2 Table Head</th>
  </tr>
  <tr>
    <td>Column 1 Row 1 Table Data</td>
    <td>Column 2 Row 1 Table Data</td>
  </tr>
  <tr>
    <td>Column 1 Row 2 Table Data</td>
    <td>Column 2 Row 2 Table Data</td>
  </tr>
</table>
```

- `<table>` defines the table.
- `<tr>` a table row.
- `<th>` a table head. Used to identify cells in a table that are a heading. Must be inside a `<tr>`.
- `<td>` short for table data, identifies a cell in the table. Must be inside a `<tr>`. Can contain most of the standard block elements, to have additional paragraphs, lists, images, etc.

There are additional elements that can be used in tables, such as `<thead>`, `<tbody>`, and `<colgroup>`. They allow for more precise formatting.

Before CSS really took off, a lot of HTML page layout was done with tables (including nested tables). This was super complicated to troubleshoot. Table layouts are now frowned upon because they are problematic for screen readers and cause issues with accessibility.

Some of the most common inline elements:

- `<b>` or `<strong>` — Bold (or strong) text
- `<i>` or `<em>` — Italic (or emphasized) text
- `<a>` — An anchor or link. Anchors require an `@id` attribute, links require an `@href` attribute.
- `<span>` — A generic span of text. Often used for formatting purposes.

The HTML5 specification defines 110 elements, so there are a lot more to learn about. There are elements for creating forms, there are elements for fancy display, elements for providing extra information to help with search, interacting with servers, and who knows what. However, these basic block and inline elements probably cover 90% of what you'll use with HTML.

Images and other file references

When you want to insert an image, what you're really doing is inserting a reference to an image file. The browser then displays the image you point it at.

The least you need to include an image is using the `<img>` element with the `@src` attribute.

```
<img src="myimage.png"/>
```

The `@src` attribute needs to be a path to the image file relative to where the HTML file is.

- Technically, the path is a *URI (Uniform Resource Identifier)*. Google it to learn more.
- Always use forward slashes (/)
- Use relative paths, not fully qualified paths.

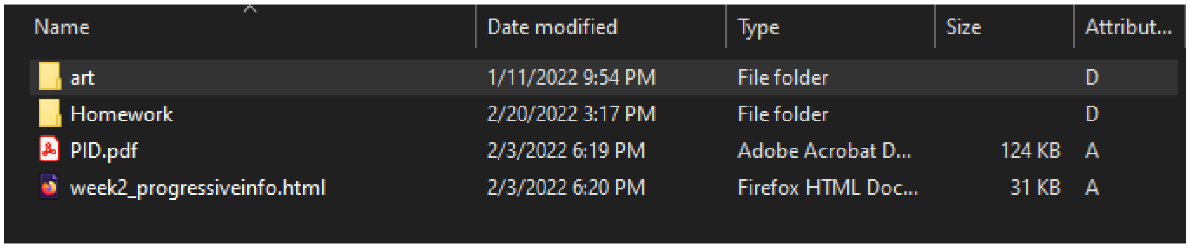
  HTML files are usually moved around, so you need to make sure the HTML file can find the image wherever it's posted.

- Avoid special characters as much as possible. If you have to have them, they most likely will need to be *URL encoded*.

  If you don't remember any other encodings, remember that `%20` is for a space.

  See https://www.w3schools.com/tags/ref_urlencode.asp for more details.

You can look at the Week02-ProgressiveInfo presentation.



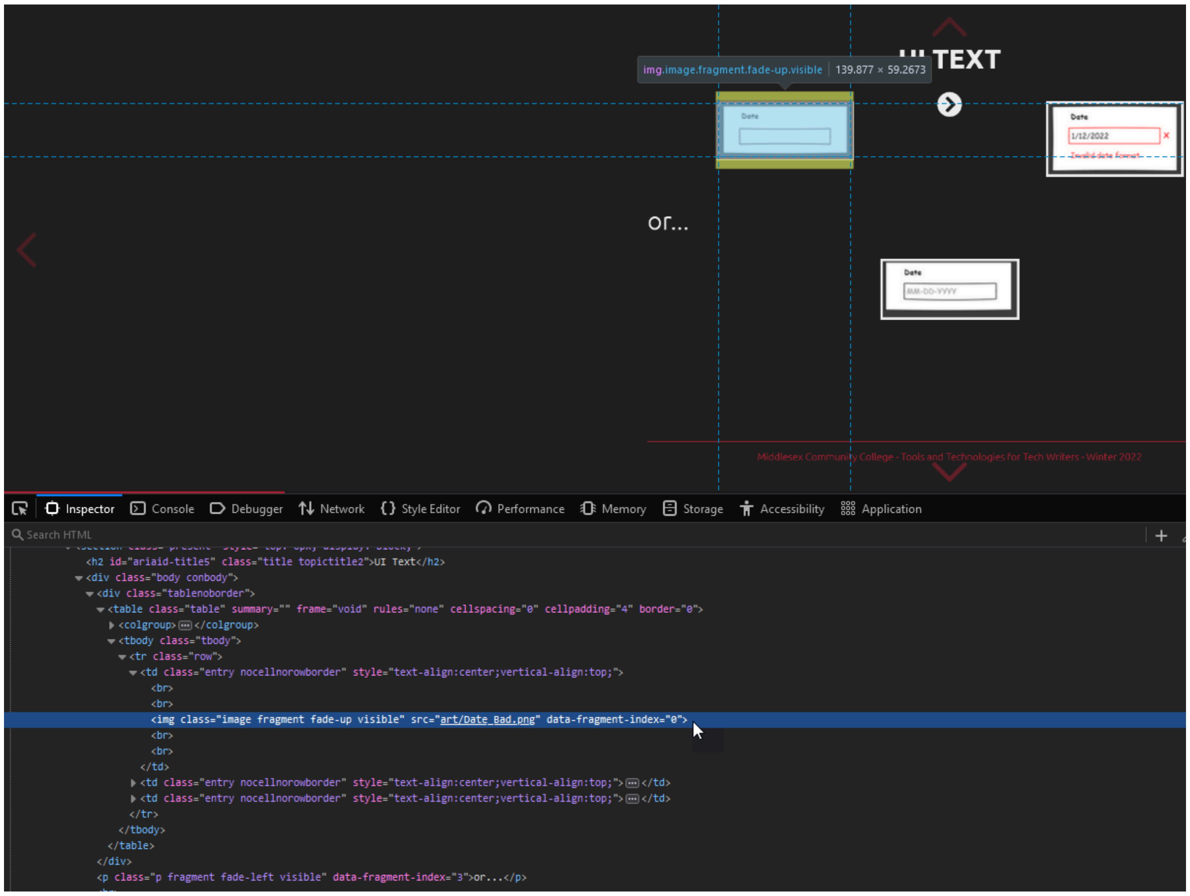
Name	Date modified	Type	Size	Attribut...
art	1/11/2022 9:54 PM	File folder		D
Homework	2/20/2022 3:17 PM	File folder		D
PID.pdf	2/3/2022 6:19 PM	Adobe Acrobat D...	124 KB	A
week2_progressiveinfo.html	2/3/2022 6:20 PM	Firefox HTML Doc...	31 KB	A

Figure 1: File and folder structure

The HTML file (`week2_progressiveinfo.html`) is in the `Week02-ProgressiveInfo` folder.

All images are in the `art` folder.

HTML display and source



```
<h2 id="ariaid-title5" class="title topictitle2">UI Text</h2>
<div class="body conbody">
  <div class="tablenoborder">
    <table class="table" summary="" frame="void" rules="none" cellspacing="0" cellpadding="4" border="0">
      <colgroup>…</colgroup>
      <tbody class="tbody">
        <tr class="row">
          <td class="entry nocellnorowborder" style="text-align:center;vertical-align:top;">
            <br>
            <br>
            <img class="image fragment fade-up visible" src="art/Date_Bad.png" data-fragment-index="0">
            <br>
            <br>
          </td>
          <td class="entry nocellnorowborder" style="text-align:center;vertical-align:top;">…</td>
          <td class="entry nocellnorowborder" style="text-align:center;vertical-align:top;">…</td>
        </tr>
      </tbody>
    </table>
  </div>
  <p class="p fragment fade-left visible" data-fragment-index="3">or...</p>
```

The selected image has the following source:

```
<img class="image fragment fade-up visible" src="art/Date_Bad.png"
 data-fragment-index="0">
```

- The element is `<img>`.
- The `@class` attribute supplies information to link how it displays to the CSS files used.
- The `@src` attribute give the relative path to the image, `art/Date_Bad.png`.
- The `@data-fragment-index` is a special attribute defined for `reveal.js`, the tool I use to make my presentations.

  (It's nifty, see https://revealjs.com/.)

For more on relative paths, see https://www.w3schools.com/html/html_filepaths.asp.

Brief CSS primer

Again, go forth unto the internet and learn CSS there.

Cascading style sheets (CSS) is a language to provide formatting for HTML tags.

Originally HTML had a whole bunch of style elements and attributes. The `@style` attribute still remains, and you can put CSS in the attribute. However, maintaining all those formatting items in the HTML itself is very difficult to maintain. Therefore people reference sets of styles using the `@class` attribute. These styles are also inherited. If you set a font and color for the `<body>` element, all the elements contained in the `<body>` use the same font and color, unless the element overrides the settings.

You can put raw CSS inside a `<style>` element in the `<head>` of the HTML file. Or you can put your CSS in a separate file and reference it with a `<link>` in the `<head>` of the HTML file. Most web sites use `<link>` elements to pull in multiple CSS files. Many generated HTML pages use the `<style>` element.

You can reference HTML tags by name, by a `@class` attribute, or by a combination of the two.

```
tagname {
  css_property: css_value;
}
.classname {
  css_property: css_value;
}
tagname.classname{
  css_property: css_value;
}
```

Again, there are over a hundred of CSS properties. It's just easier to look them up, I always do. A few to keep in mind:

- color — change the color of the font
- background-color — change the background of the whole block. This is super useful when you're arguing with getting blocks to line up.
- border — add borders to a block. Also super useful when you're arguing with getting layout correct.

- padding — space around the content of the block. Related to but different from margins.
- margin — space around the block. Related to but different from padding.

There is some arcane voodoo about how padding and margins interact to define how different elements are spaced around each other. Understanding the *box model* is fundamental to understanding CSS. And sometimes your browser actually follows the rules.

Whenever you are arguing with CSS, the **inspect element** option in your browser is your friend.

By the way, browsers still interpret and implement HTML and CSS uniquely. This is why we still have tools and web pages that state "only works in Internet Explorer". You may be programming the HTML and CSS according to the specification, and the browser may still interpret it in a different way. Always test with multiple browsers, and if possible, different types of devices, such as laptops, tablets, and phones.

Why HTML is still important

No one writes HTML pages anymore, they write applications that write HTML to display content.

For example, many web sites are written in PHP or Ruby. When writing your PHP or Ruby, you're making scripts that produce HTML as the output. Therefore, in order to be able to really use PHP or Ruby, you have to understand the HTML you're trying to create.

A lot of software is either a web application that runs in a browser, or ends up having to emit HTML somewhere at some point. Even if you're not writing pure HTML, a lot of tools let you use HTML tagging to control the formatting. Therefore, knowing basic HTML is something you'll end up using relatively often.

HTML Resources

A few web sites to take a peek at.

W3 Recommendation	https://html.spec.whatwg.org/multipage/ This is the official spec for HTML5. However, reading a spec is hard, but you can find exactly what an element is supposed to do. The history section gives an interesting read on the development of HTML.
W3 Schools HTML Tutorial	https://www.w3schools.com/html/ My go to for remembering tagging details and learning new features.
W3 Schools CSS Tutorial	https://www.w3schools.com/css/default.asp My go to for looking up HTML formatting. I usually search and find something on StackOverflow for specific questions, but if I can almost remember the formatting feature and I just need a refresher, this is where I go.

Document Object Model	Brief intro: https://www.w3schools.com/js/js_htmldom.asp More details: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
Cool color picker	http://paletton.com I am not a graphic designer and my understanding of color theory is limited. But this site is fun to play with.
Font Awesome	https://fontawesome.com/ The world has realized that everyone needs icons, and some folks are providing them for free. See https://www.w3schools.com/css/css_icons.asp for how to use them.
Web Design in a Nutshell	https://www.oreilly.com/library/view/web-design-in/0596009879/ A smidge out of date, but I appreciated having all the bits and bobs in one place.

Week 5 Homework

Practice making HTML pages

See `Week05-AgileHTML\Homework\Sample_HTML_File.html` for an example. You just can't copy it for your own.

Week 5 Homework

Make an HTML file that contains:

- Headings
- A list
- A table
- An image
- Paragraphs
- Various inline formatting

More details are in `Week05-AgileHTML/Agile_and_HTML.pdf`

Style the page

Use CSS to style your page.

- Change the font
- Change Heading styles
- Change the background color

To make things interesting

None of these are required, but they're a lot more interesting to play with.

- Make two different sets of CSS and show how you can style one page in different ways.
- Make your page two columns (without using an invisible table)
- Try to use Font Awesome