

# Development and Use of An Advanced Methodology for Performing Accessibility Audits in the Federal Government

Karl Groves
Senior Accessibility Consultant, SSB BART Group



# Agenda

- Introduction
- Typical Auditing Methods
  - Pitfalls
  - Automated Tools
  - Manual Review
  - Use Case Testing
- Developing a Methodology
- Reporting results



# **SSB BART Group**

A bit about us...

#### SSB Technologies

- Founded in 1999 by technologists with disabilities
- First commercial company in the testing software space
- Focus on IT Manufacturers and private organizations

#### BART Group

- Founded in 1998 by individuals with Visual Impairments
- Focus on East coast and federal market

#### Customer Base

- Over 500 commercial and government customers
- Over 800 projects successfully completed
- Accessibility Management Platform
  - Assessments and Audits
  - Standards
  - User Testing
  - Training and eLearning



# **Customer Experience**



































# Typical Accessibility Audit Techniques



# **Pitfalls**

- Typical methods are often haphazard and seem to be made up on the spot:
  - Running the system through an automated test (in the case of websites)
  - Or, going through the list of technical provisions and taking a cursory glance at the product to see if it complies in an ad hoc test of each provision



# **Pitfalls**

#### Testing Methods Are Often Incomplete, Inaccurate, Inconsistent

- Performing an ad hoc set of tests is more likely than not to result in test results that are incomplete at best
- The test results may not touch on every possible problem a disabled user might face.
- Automated tests may remain unable to notice some of the more egregious errors in today's modern web sites



# **Pitfalls**

#### Testing Methods Are Often Not Repeatable

- Any test performed on an ad hoc basis may net results that are not repeatable throughout multiple regression tests.
- When it comes to perform a regression test, the "make it up as you go" approach will be unable to determine whether the issues uncovered in previous tests were sufficiently remediated.



# **Automated Tools**



#### **Automated Tools – Introduction**

#### What is it?

- Use of desktop or web-based tool to parse document markup to check for potential areas of accessibility problems.
- May or may not involve the use of spiders to crawl multiple pages.
- May or may not involve ability to schedule repeat tests and/ or automate reports.



# **Automated Tools – Strengths**

- Ability to scan large volumes of code.
  - On a single page, site wide, and anything in between
- Ability to automatically generate reports
- Ability to catch errors which do not need humans to review
- Configurable to include/ exclude specific guidelines.
  - Checking method for specific guidelines often also configurable



#### **Automated Tools - Flaws**

#### Notoriously prone to inaccurate results:

- Passing items which should fail, i.e. insufficient alt attribute values.
- Failing items which should pass, i.e.:
  - missing <label> for <input> element which has 'hidden' or 'submit' as
     value for type attribute.
  - Missing <meta> for language, when language defined via lang attribute of <html>



# **Automated Tools - Flaws (cont'd)**

- The bulk of tools utilize spiders.
- Spiders tend not to do well with:
  - Form driven authentication
  - Form driven workflows
  - Pages that utilize JavaScript to render content.
  - The bulk of enterprise class web-enabled applications contain all of these elements.



# **Automated Tools - Flaws (cont'd)**

- Questionable checking rules
  - "Failing" a document for items which have no real-world impact on access.
- The tools test rendered HTML, sometimes CSS, but not JavaScript or non-text formats (i.e. Java Applets, Flash, etc.)
- Markup may look good, but page may use DOM Scripting/ AJAX which makes it inaccessible.
- Tools often test only the markup as a string without assessing DOM structure
  - Analogy: PHP's file\_get\_contents vs. DOMDocument



### **Automated Tools - Flaws (cont'd)**

- Unable to test the functional standards (§1194.31)
- Automated tool may be unable to access the site to test it.
  - Security restrictions may disallow installation of automated tool on client system or may disallow the running of spiders



# **Manual Review**



#### **Manual Review - Introduction**

#### What is it?

- Code-level review of the generated HTML/ CSS markup, specifically oriented toward finding potential areas of accessibility problems.
- Methods meant to mimic coping mechanisms and/or uncover errors
  - Manipulation of software or hardware settings



## **Manual Review - Strengths**

- Much higher level of accuracy (for individual violations) than any method.\*
- Reviewer likely to be capable of not only finding the error but can also recommend the necessary repair at the same time.



#### **Manual Review - Flaws**

- Relies on extensive knowledge on the part of the tester.
- Reviewing large volumes of code far too time intensive.
- The more code/ the more complicated the code, the greater chance the reviewer will miss something.
- Mostly limited to inspection of HTML & CSS



# Manual Review - Flaws

There are just some things that don't require human eyes to catch!



# **Use Case Testing**



# **Use Case Testing - Introduction**

#### What is it?

 Similar to use case testing/ acceptance testing for QA: the actual use of a system by users with assistive technology performing typical system tasks.



# **Use Case Testing - Strengths**

- The true measure of a system's level of accessibility is whether or not disabled users can use it effectively.
- Provides ability to catch issues which may have gone unnoticed by other methods.
- Provides a much more 'real' impression of the severity and volume of problems uncovered.
- Particularly useful in finding failures of 1194.21(b)
   provisions which cannot be uncovered any other way.



#### **Use Case Testing - Flaws**

- Dependent upon proper authoring of use cases
  - Too broadly worded, testing may take too long to be economical vs. results returned
  - Too narrowly worded may 'lead' the tester too much to be realistic.
- Time & budget constraints may leave large portions of system untested.



# **Use Case Testing – Flaws (cont'd)**

- Less accurate when testing is performed by non-disabled user.
- Tester may be unrepresentative of common user.
- Results can vary widely based on not only the AT type but also the brand and even the version.
  - Success with one specific AT does not correlate to success with all AT.
  - Success with one specific AT is not indicative of compliance



# **Use Case Testing – Flaws (cont'd)**

There are just some things that don't require user-based testing to catch!



# Toward A Better Methodology

# Requirements

- Accuracy
- Efficiency
- Reliability
- Repeatability
- Actionability (is that a word?)

#### Requirements

#### Accuracy

No singular method is sufficiently accurate on a large scale project.

#### Efficiency

 The more efficient methods are inaccurate, the more accurate methods are inefficient.

#### Reliability

No singular method can be reliable for predicting real world accessibility by all
users.

#### Repeatability

- Any assessment should be structured in a way in which it can be repeated accurately during subsequent regression tests.
  - The goal of testing isn't to generate reports, it is to work toward resolution of problems

#### Actionable

Results must be reported in a fashion that makes the results actionable.



# **A Trident Approach**

#### Unit-based Testing

- Tested via Automated and Manual means
- Automated tests reserved only for checking what automated tests can check effectively.
- Manual means validate & verify automated tests
- Will also determine at a low level if code is written in a compliant fashion

#### Use Case Testing

- Tested with multiple assistive technologies
- Will determine from a high level if the application is usable for people with disabilities
- Further validates and verifies results from automated & manual tests

#### Actionable Results Reported, Repairs Prioritized



# **Unit-Based Testing**



#### What We Know About Web Production

- Enterprise-level websites and web-based applications are mostly generated server-side.
- Backend programming libraries are pre-processed at time of request (or cached) to assemble front-end interface.
- This (usually) means all interface elements of a specific type (forms, tables, templates, etc.) will be written to screen using essentially the same code.



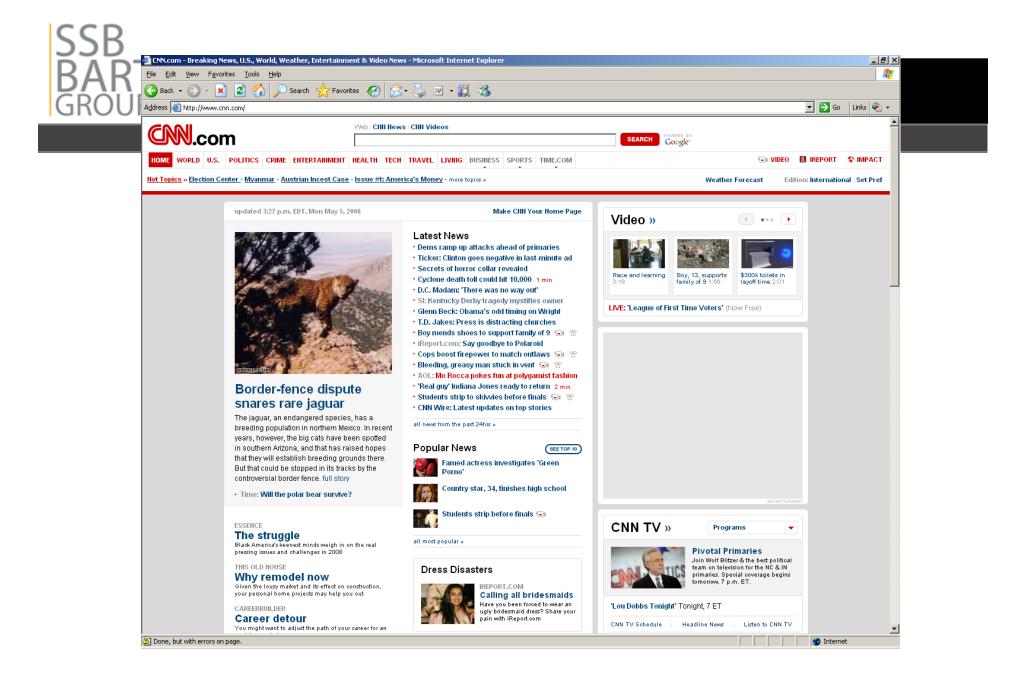
#### What We Know About Web Production

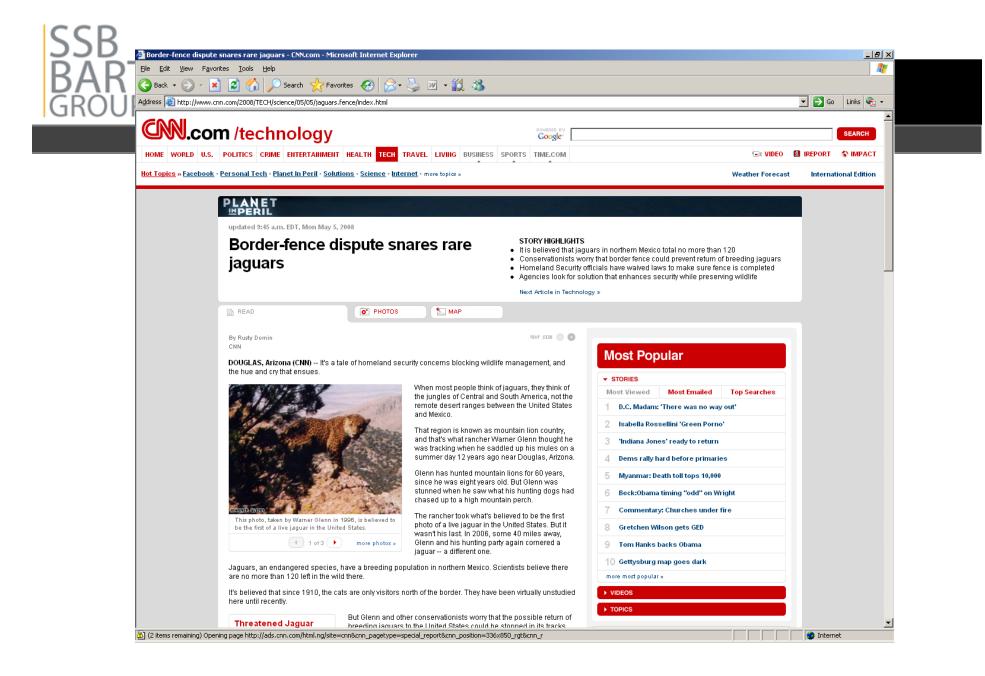
- Interfaces often driven by templates which look mostly identical on all pages and only the unique content changes
- Even where templates vary, variances are few and are also driven from server side code.

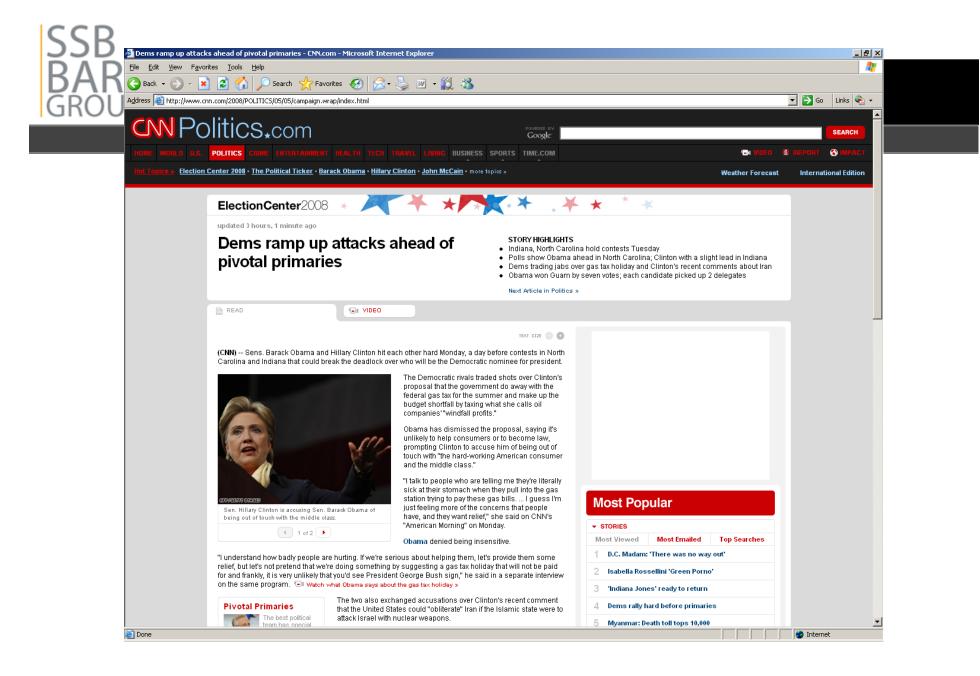


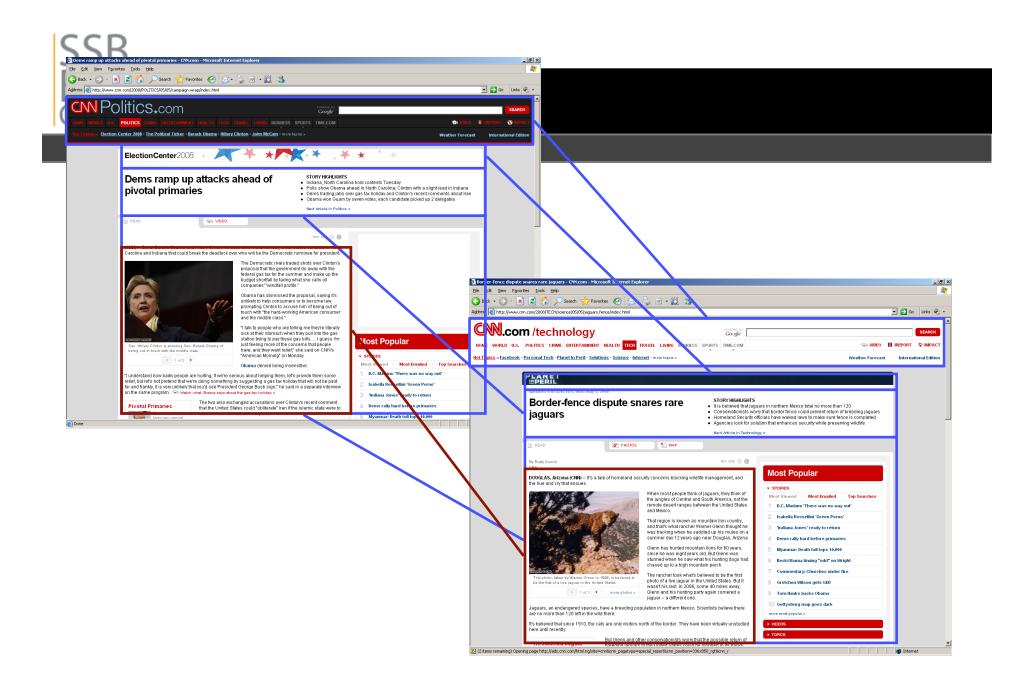
#### What We Know About Web Production

- This approach decreases production time for new content, increases quality, decreases maintenance debugging time.
  - For our purpose it also tends to let accessibility problems propagate themselves throughout the whole system.
  - Fortunately, it makes them easier to fix, too.











# Determining What & How To Test



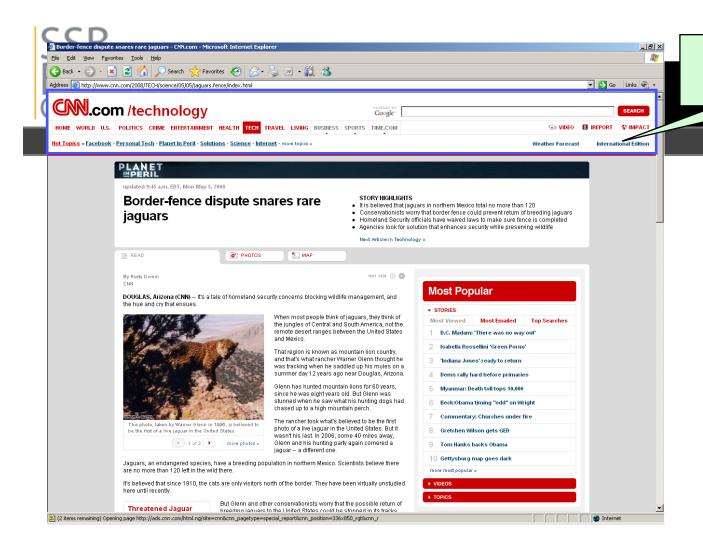
## **Producing our Component List**

- Test coverage = Entire UI of the application
- Test set is a list of all unique UI components in an application
- Prioritize testing efforts based on frequency of these components and potential impact on users

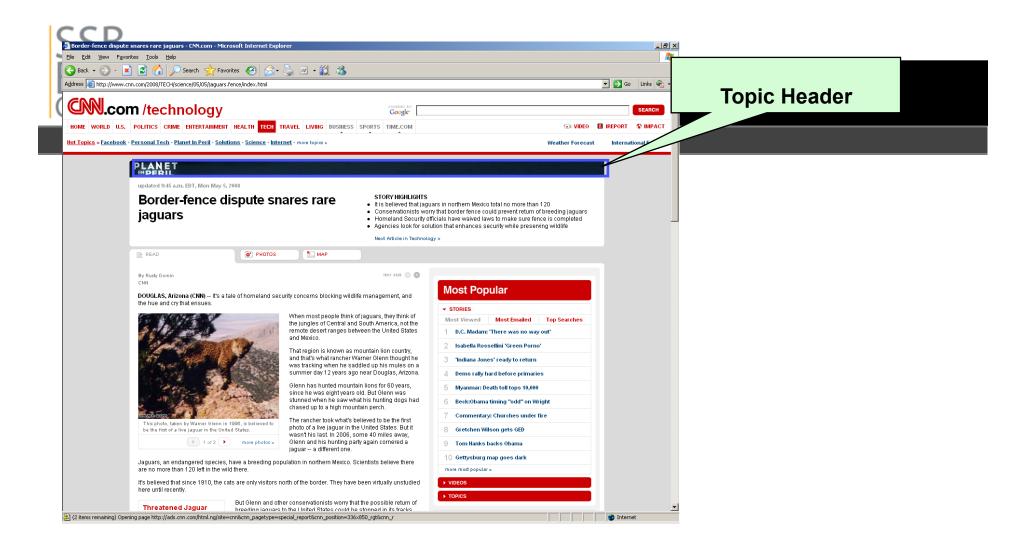


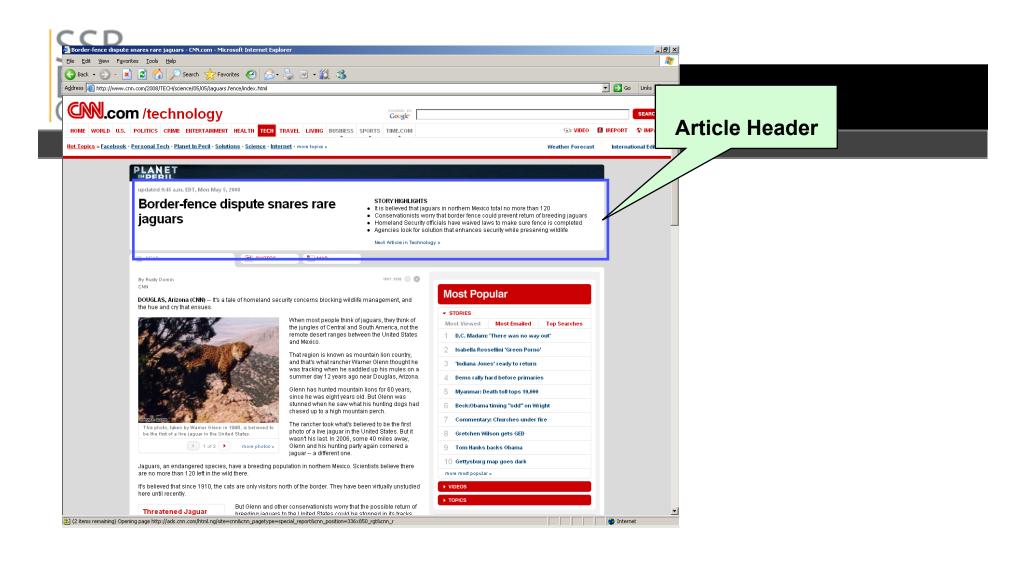
# **Producing our Component List**

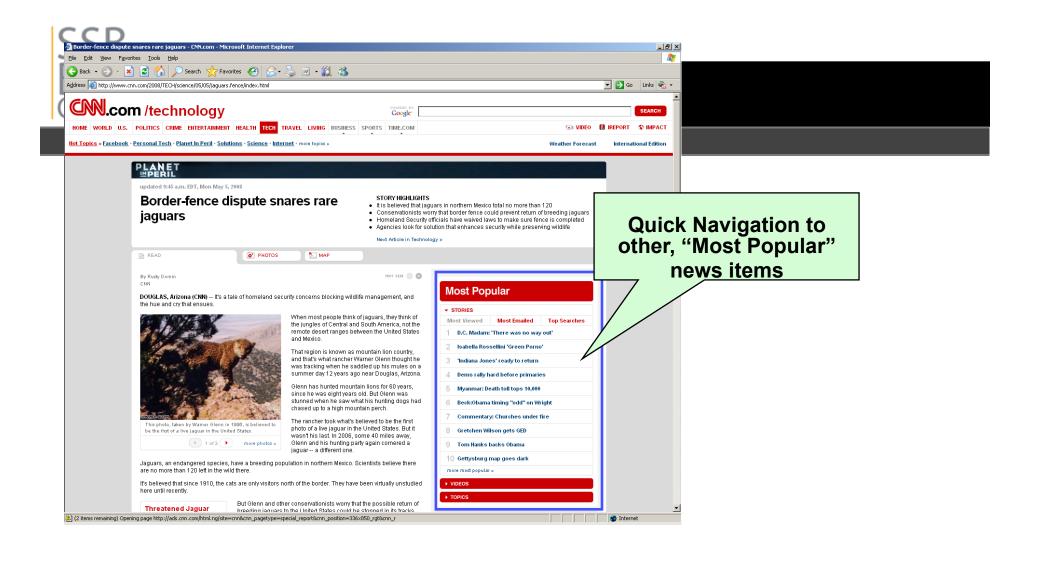
- Attempt to include all interface component types historically found to cause challenges for disabled users:
  - Images & other non-text formats
  - Forms
  - Tables
  - Interface elements relying on client-side scripting
  - Frames and i-frames
- Always include the overall template

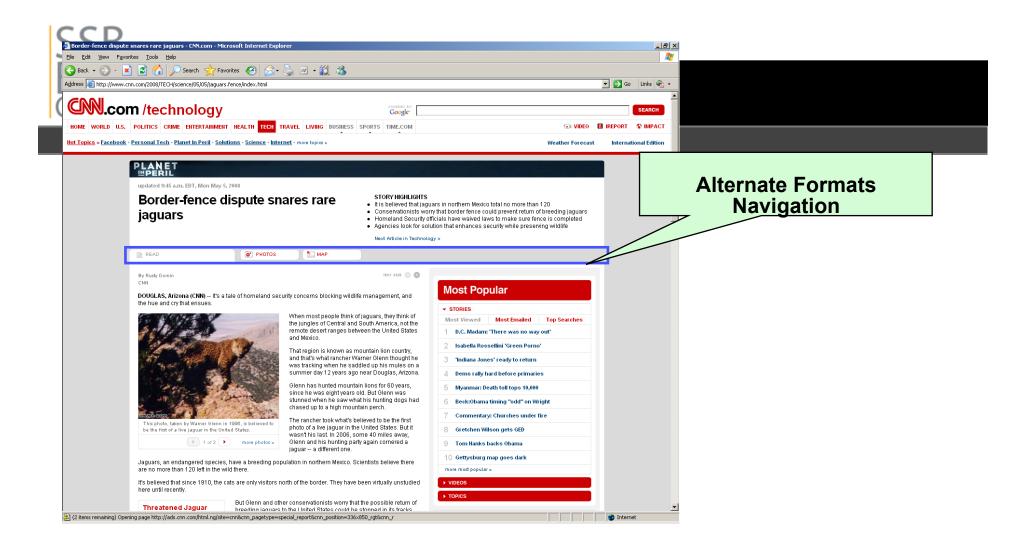


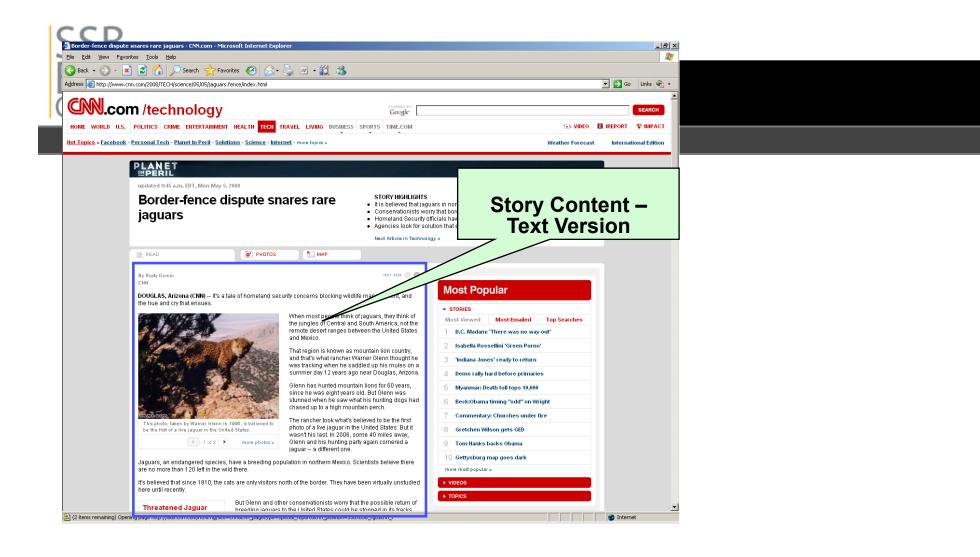
**Global Header** 

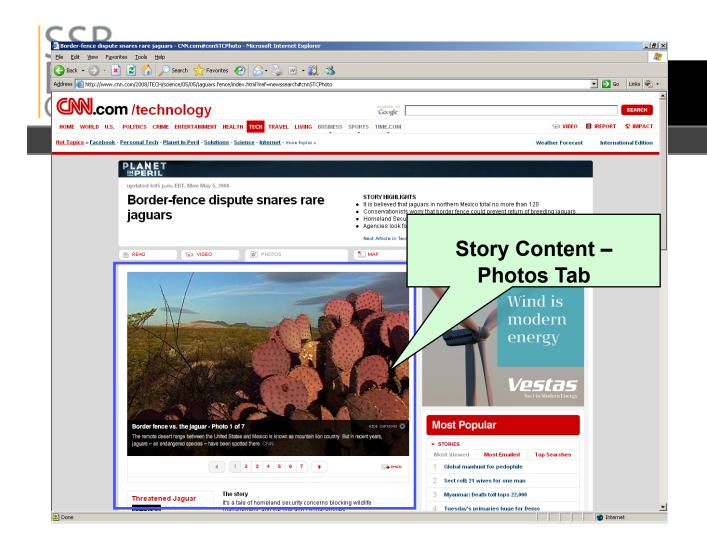














## **Producing our checklist**

- Based on technologies in use, determine a set of tests to be performed.
- Produce a checklist which touches:
  - Each component
  - Each container
  - The entire application
- Checklist validity enhanced by being based on industry standards.
- Checklist composed of Best Practices



# **Best Practices**

- Ensure the standards are met by performing a thorough and complete audit.
- Dissection of each Industry Standard you've committed to support.
- Each provision separated into their conformance criteria



# **Best Practices (cont'd)**

- Establish a defined set of conformance criteria for the standards the system needs to meet.
- Establish a checklist which can be used not only for the initial test but also any subsequent regression tests as well.
- Means the review not hampered by the reviewer's memory or knowledge of the standards or inconsistent interpretation of the standards by team members
  - One best practice = One test = One result



# **Best Practices (cont'd)**

#### Each Best Practice Should Contain

- Title
- Description
- Compliant Code Example
- Non-Compliant Example
- Recommended Changes
- Unit Test for Checking Compliance
- This information is essential for ensuring accuracy and repeatability!



# **Producing our checklist**

# Checklist Creation - Component Level Example (Sample Best Practices based on 1194.22(a))

Provide alt attributes for all images				
Ensure images which convey meaning are not defined in CSS				
Ensure image text and alternative text are equivalent				
Ensure alt text is sufficiently informative				
Ensure alt text is not descriptive of the appearance but rather the content				
Ensure complex images provide long descriptions				
Avoid redundant alt attributes				
Provide mathematical formulas in appropriate markup				
Ensure alternate text for linked images is descriptive of destination				



#### **Unit-based Test Execution**

- Following your checklist:
- Test each component for each applicable best practice in your checklist
- Run automated tests on those which can be graded automatically
  - i.e. presence of alt text, presence of device-dependent event handlers
- Each best practice graded as pass/ fail
- Mark each violation in your checklist and/ or in your bug tracking system
- Take note of patterns which reveal themselves during testing
- Take special note of patterns that exist throughout the entire set of components.



# **Use Case Testing**



## **Use-Case Methodology**

- Create use cases to be performed with application
- Define a list of assistive technologies to execute the use cases with
  - Should include more than one type of AT



#### **Use-Case Methodology**

#### Usage case consists of

- Actor The individual that is performing the task
- Goal A clear definition of what the actor is attempting to accomplish
- Main Success Scenario A list of the ideal steps the actor must take to accomplish a task
- Extensions Alterations to the task that may occur during execution
  - Particularly errors and alternate navigation possibilities



# **Use-Case Methodology**

#### **Usage Case – Example Web Site**

(simplified for this presentation)

Goal	Access Story Photos					
Operator	User					
Main Success Case	<ol> <li>Navigate to http://www.cnn.com/</li> <li>Navigate to link labeled story headline</li> <li>Navigate to 'Photos tab'</li> <li>View Photos</li> </ol>					
Extensions	2.a. Navigate to 'Most Popular Menu' 2.a.1. Navigate to link labeled story headline					



#### **Use Case Test Execution**

- User(s) perform exact same test cases for each assistive technology.
- Take note of any & all difficulties in performing task.
- Grade test case based on:
  - Failure
  - Completed with difficulty
  - Improvements needed
  - Success



# Reporting the Data



## **Reporting The Data**

- Automated test results validated by manual test
- Manual & automatic test results form the bulk of report
  - Because they'll be the bulk of the findings
- Use case results incorporated in report
  - Because they provide a 'face' to the real world effect
- All together, they form basis for further regression testing



# Reporting The Data

- Organize results based on content type rather than by provision/ guideline
  - i.e.: Data Tables, Forms, Images, etc.
  - Doing so provides context for both stakeholders and is actionable by developers.
- Prioritize list of violations based on:
  - Severity
  - Frequency
  - Noticeability
  - Tractability





- Some portions of prioritization should be worked out on a per-best-practice basis prior to evaluation, as part of the overall methodology.
- Each best practice contains a predetermined:
  - Severity
  - Noticeability
  - Tractability
- Each of these is also given a weight used as coefficient during calculation of priority.



#### **Prioritization**

#### "Severity":

- A measure of how large an impact on the disabled user experience a violation of the best practice will have.
- Violation severity is inferred from our cumulative knowledge gained from observing disabled users.





#### "Noticeability"

- The likelihood that a given violation will be detected by users of a document.
- Certain best practice violations are more easily detected than others, such as violations that can be detected with automated tools.
- Other violations, such as those that can only be detected through manual review techniques, are more difficult to find in a document.
- Violations that are more difficult to detect generally pose a lower overall risk for enforcement than violations which can be detected in a trivial fashion.



#### **Prioritization**

### "Tractability"

- The estimated costs associated with ensuring that all instances of the violation are fixed.
- The cost is designed to give an estimate of the number of hours of effort required to ensure compliance with a given best practice.





#### Frequency

- How often a particular violation occurs within a document.
- Violation Frequency is calculated based on the number of pages that exhibit a violation divided by the total number of pages, multiplied by ten
  - Example: Violation on 54% of components would be frequency of 5.4



# **Prioritization**

Name	Severity	Frequency	Noticeability	Tractability	Priority
Provide alternative text for images	10	6.8	10	2	10
Ensure images provide informative alternative text	10	8.5	7	1	10
Provide explicit labels for form fields	10	9.5	6	2	9.6
Provide a mechanism for skipping past repetitive navigation links	10	10	6	3	9.4
Ensure headers and cells are properly associated	10	9.5	7	4	9.1
Ensure row header cells define scope	10	8.3	7	4	8.8
Provide alternative text for image buttons	6	6.8	8	2	7.6
Ensure alternative text for image links is informative	8	3.8	7	2	7.5
Ensure table headers are used in a valid fashion	10	6.3	4	4	7.5
Ensure documents are readable without style sheets	10	10	3	6	7.4
Ensure the sole use of device dependent event handlers is avoided	8	3.3	7	2	7.3
Ensure link text is meaningful	8	4.3	6	2	7.3



## Conclusion

- Unit-based approach facilitates greater accuracy, efficiency, and reliability.
- Reserving automated testing for things it is best suited to find reduces inaccurate results.
- Performing manual review on components rather than whole documents is more efficient.
- Unit-based approach provides repeatability
- Use case testing validates findings and provides context.
- Prioritization will allow for findings to be actionable.



# For more information...

#### Contact Karl Groves

- karl.groves@ssbbartgroup.com
- 703.637.8961 (o)
- 443.889.8763 (c)

#### SSB BART Group

- Silicon Valley Office
  - 415.975.8000
- Washington DC
  - 703.637.8955

sales@ssbbartgroup.com