Testing

- Acceptance Criteria
- Acceptance Test
- Test Cases
- Structural Testing
- Bug Testing
- Non-Functional Testing

Acceptance Criteria

Created based on updated user story.

ID	Feature	**Acceptance Criteria** *(Given/When/Then)*
1	Input Interface	Given the user is looking for a website to compare two different texts, when the user accesses the website, they then get greeted with a *user-friendly interface* where they can input a set of known texts and an unknown text for comparison
2	Security	Given the user is uploading documents that may be sensitive or personal, when the user uploads the documents to our website, our website processes the information then returns the result through the already pre-trained model without storing any of the documents
3	Multi-file type input	Given the user has documents of multiple types to upload, when the user uploads documents, they have the flexibility to upload .txt, .docx and .pdf, alongside textbox input. Then the user can start the comparison model
4	Results Explanation	Given that the user has uploaded and clicked compare, when the model has finished processing the texts, the results are then presented as a large visual circle with colour indicating similar percentage. The results is given alongside a set of data on the similarity information such as word similarity, punctuation percentage and more.
5	Instructions	Given that the user is not familiar with the website or has forgotten how to use it, when the user clicks at the bottom right of the website (the ? mark button), the instructions then pop up to help assist the user

Adequacy criterion = set of test obligations. E.g., cover all statements and all branches in the program under test

A test suite satisfies an adequacy criterion if

✔ all the tests succeed (pass)

✔ each test obligation is satisfied by >=1 test case(s)

Acceptance Criteria: set of predefined requirements that must be met in order to mark a user story complete.

- given [condition], when [something happens], then [result]

Acceptance Test

AC ID	AT ID	Acceptance Test	TC	Success?
1	1.1	User is able to see the home/welcome screen upon accessing website	**TC-5**	✓
	1.2	User is able to see the main features of website after successful load out of website	**TC-5**	✓
	1.3	User is able see all the navigation and buttons clearly	**TC-5**	✓
3	3.1	User is able to upload .pdf, .docx, .txt, or a combination of them	**TC-2**	✓
	3.2	User is able to use the text input box multiple times through converting them into files with the '+' button	**TC-3**	✓
4	4.1	User is able to successfully upload files/utilise the text boxes	**TC-2, TC-3**	✓
	4.2	User is able to successfully press the 'Compare' button	**TC-1**	✓
	4.3	User is able to see the animation and display of authorship comparison results.	**TC-1**	✓
	4.4	User is able to see the visual explanation bars on the details extracted from the comparison	**TC-1**	✓
5	5.1	User is able to click the '?' button at the bottom right of the website	**TC-4**	✓

AC - Acceptance Criteria, AT - Acceptance Test, TC - Test Case

Test Cases

Test Case [TC-1]

Test Type: Functional **Execution Type:** Manual

Objective: Test if results screen shows proper results after pressing compare button given correct input

Setup: Must have at least one known text and one unknown text

Pre-Conditions: Dynamic results bar shows resulting similarity percentage with colour depending on percentage

Notes:

1. Add known text(s) and unknown text through textbox or uploading files
2. Press compare button
   - results section should appear
   - results bar should have fill animation with colour based on percentage
   - explanation section should show evaluation metrics with numbers and similarity bars

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-2]

Test Type: Functional **Execution Type:** Manual

Objective: Uploading files on compare page

Setup: Be on compare page with files ready to upload

Pre-Conditions: Any files uploaded to either the known or unknown texts sections should show up as file cards under the text boxes while can be removed by clicking the 'x' button

Notes: Types of files accepted: PDF, docx and txt

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-3]

Test Type: Functional **Execution Type:** Manual

Objective: Utilize the '+' button in the text area to save text box text into a file - multiple inputs

Setup: Be on compare page with multiple texts ready to upload

Pre-Conditions: multiple texts of medium length separated for multiple input attempts

Notes: N/A

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-4]

Test Type: Functional **Execution Type:** Manual

Objective: Seeing if tutorial pop-up button works

Setup: Be on compare page

Pre-Conditions: After clicking the '?' button in the bottom right of the page, a pop-up should appear with images containing instructions on how to use the website

Notes: Images should be easily readable and instructions should be clear for a new user

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-5]

Test Type: Functional **Execution Type:** Manual

Objective: Fully functional load out of deployed product

Setup: Access to link - AuthoWrite

Pre-Conditions: Website is fully deployed, no crashes occurred and latests changes have been updated and passed through test cases

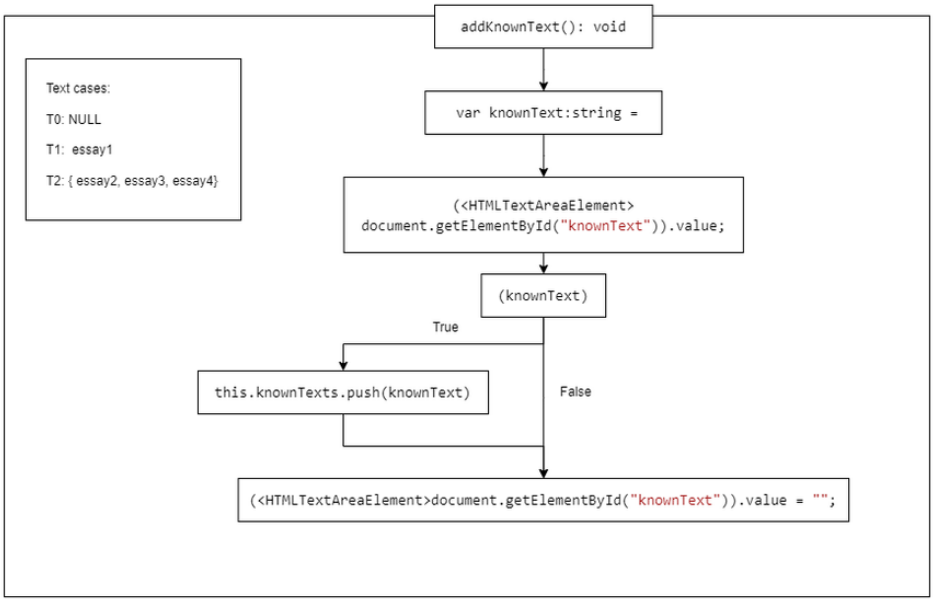
Notes: Check through that all things are loaded, buttons are all present

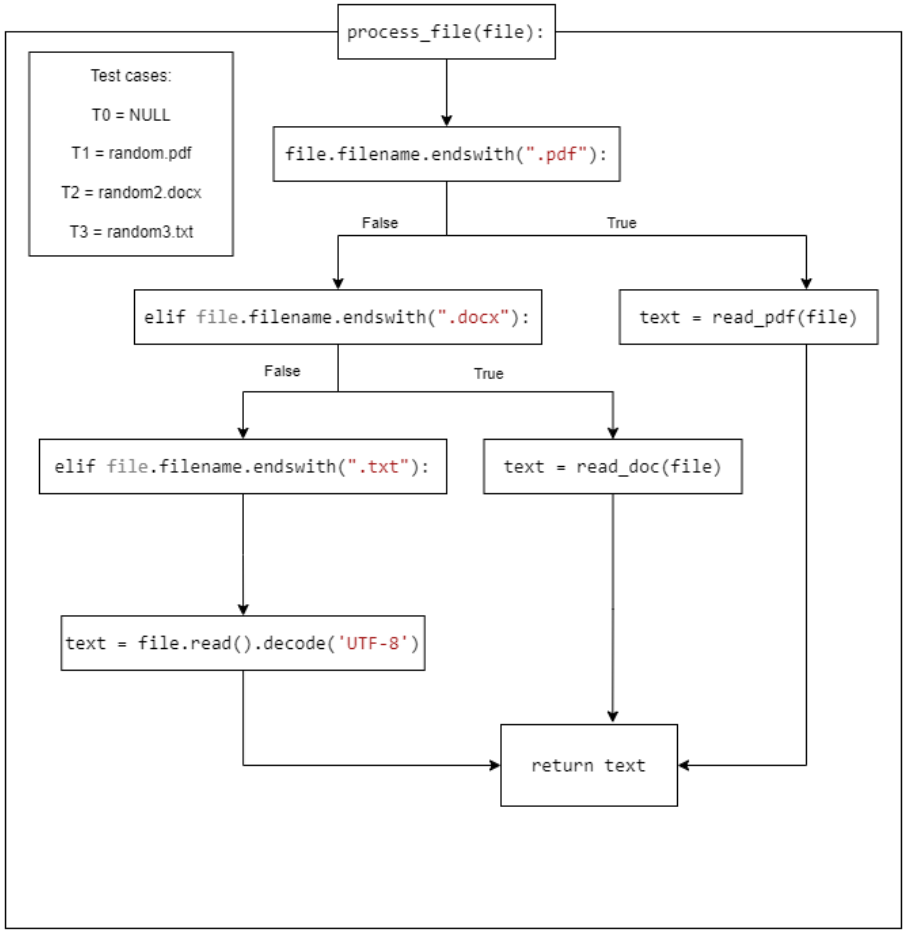
Time constraint:

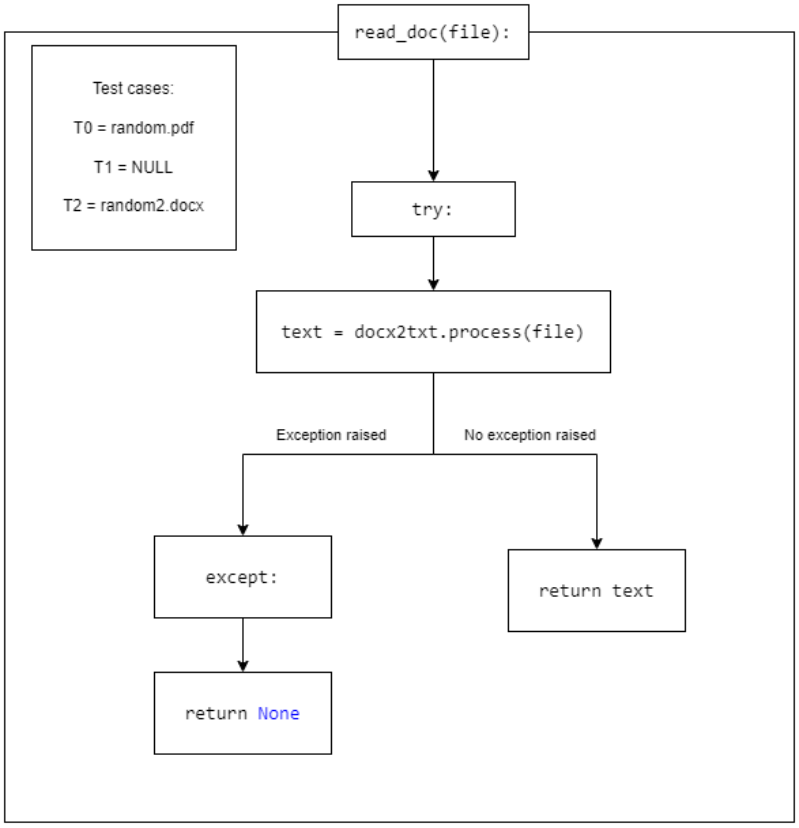
Minimum: 2 min

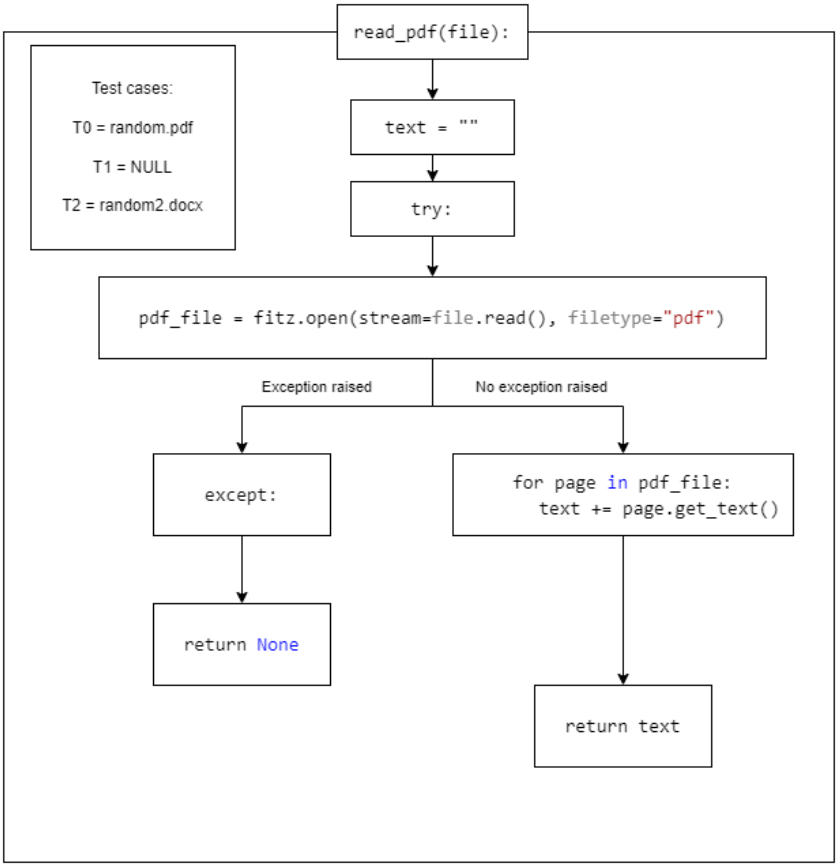
Maximum: 10 min

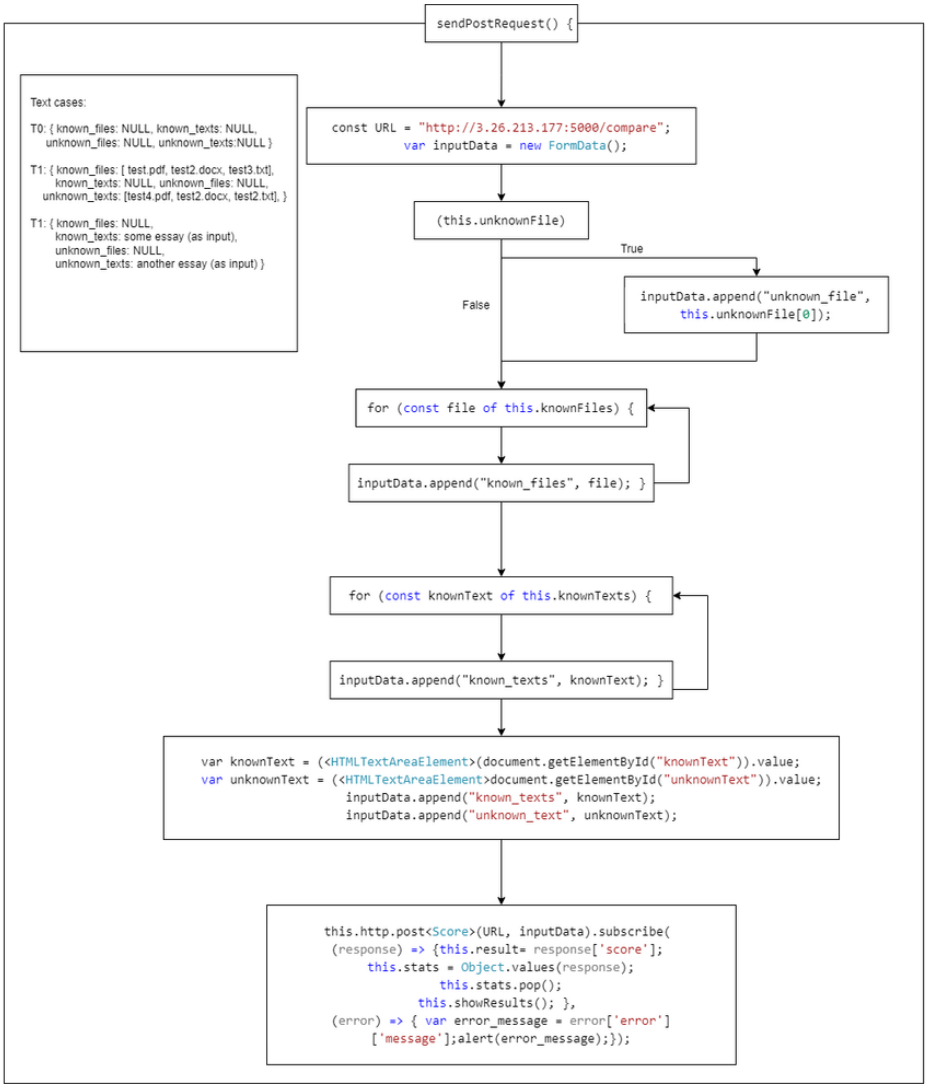
Structural Testing

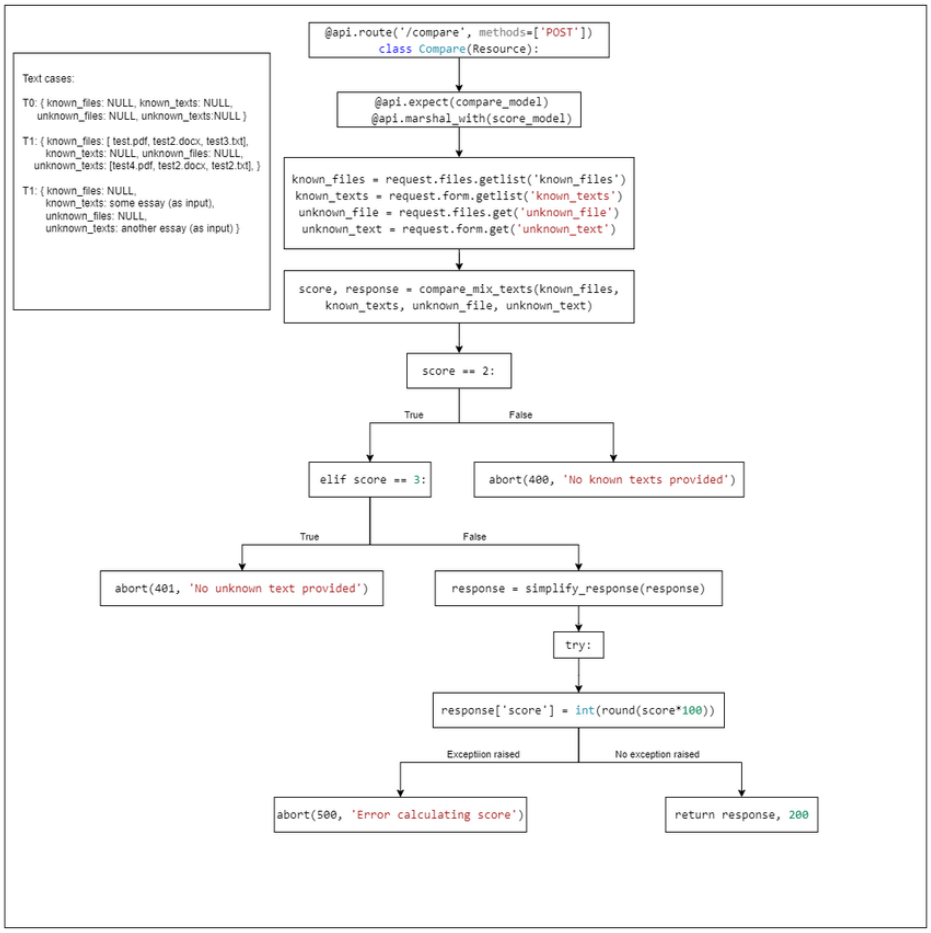



process_file(file):
Test cases:
T0 = NULL
T1 = random.pdf
T2 = random2.docx
T3 = random3.txt
file.filename.endswith(".pdf"):
False
True
elif file.filename.endswith(".docx"):
text = read_pdf(file)
False
True
elif file.filename.endswith(".txt"):
text = read_doc(file)
text = file.read().decode('UTF-8')
return text


read_doc(file):
Test cases:
T0 = random.pdf
T1 = NULL
T2 = random2.docx
try:
text = docx2txt.process(file)
Exception raised
No exception raised
except:
return text
return None


read_pdf(file):
Test cases:
T0 = random.pdf
T1 = NULL
T2 = random2.docx
text = ""
try:
pdf_file = fitz.open(stream=file.read(), filetype="pdf")
Exception raised
No exception raised
except:
for page in pdf_file:
text += page.get_text()
return None
return text


sendPostRequest() {
Text cases:
T0: { known_files: NULL, known_texts: NULL, unknown_files: NULL, unknown_texts:NULL }
T1: { known_files: [test.pdf, test2.docx, test3.txt], known_texts: NULL, unknown_files: NULL, unknown_texts: [test4.pdf, test2.docx, test2.txt], }
T1: { known_files: NULL, known_texts: some essay (as input), unknown_files: NULL, unknown_texts: another essay (as input) }
const URL = "http://3.26.213.177:5000/compare";
var inputData = new FormData();
(this.unknownFile)
True
False
inputData.append("unknown_file", this.unknownFile[0]);
for (const file of this.knownFiles) {
inputData.append("known_files", file); }
for (const knownText of this.knownTexts) {
inputData.append("known_texts", knownText); }
var knownText = (<HTMLTextAreaElement>(document.getElementById("knownText")).value;
var unknownText = (<HTMLTextAreaElement>document.getElementById("unknownText")).value;
inputData.append("known_texts", knownText);
inputData.append("unknown_text", unknownText);
this.http.post<Score>(URL, inputData).subscribe(
(response) => {this.result= response['score'];
this.stats = Object.values(response);
this.stats.pop();
this.showResults(); },
(error) => { var error_message = error['error']
['message'];alert(error_message);});


Text cases:
T0: { known_files: NULL, known_texts: NULL,
unknown_files: NULL, unknown_texts:NULL }
T1: { known_files: [test.pdf, test2.docx, test3.txt],
known_texts: NULL, unknown_files: NULL,
unknown_texts: [test4.pdf, test2.docx, test2.txt], }
T1: { known_files: NULL,
known_texts: some essay (as input),
unknown_files: NULL,
unknown_texts: another essay (as input) }
@api.route('/compare', methods=['POST'])
class Compare(Resource):
@api.expect(compare_model)
@api.marshal_with(score_model)
known_files = request.files.getlist('known_files')
known_texts = request.form.getlist('known_texts')
unknown_file = request.files.get('unknown_file')
unknown_text = request.form.get('unknown_text')
score, response = compare_mix_texts(known_files,
known_texts, unknown_file, unknown_text)
score == 2:
True
False
elif score == 3:
abort(400, 'No known texts provided')
True
False
abort(401, 'No unknown text provided')
response = simplify_response(response)
try:
response['score'] = int(round(score*100))
Exceptiion raised
No exception raised
abort(500, 'Error calculating score')
return response, 200

Bug Testing

Functional testing:

- best for missing logic faults
- A common problem: Some program logic was simply forgotten
- applies at all levels
  - unit - form module interface spec
  - integration - from API or subsystem spec
  - system - from system requirements spec

- tested corrupted files
- tested scroll bars - still need to change
  - include edge
- tested files which do not contain any text (eg. only images)

Mobile Testing

- Found an initial issue of a white background when zooming out of the page
  - Solution: needs to be changed to blue
- New issue after deployment of a failure in the responsive design
  - Solution: fix issues with flex and flex-grid

Non-Functional Testing

- **Visibility**
  - When we hit “compare”, the page automatically scrolls down to the results section. We implemented this feature because it was not intuitive as to where the results were.
  - We added a tutorial page that shows clearly how to use the website.
  - We have file icons whenever a user adds a file, and users can also press the “X” to delete them. We implemented this feature because using only alert boxes to notify the addition of a file was not very clear, and also this allows users to remove files easily.
- **Match**
  - Also for adding files, we use the universal “+” symbol to show users that they can add files. Furthermore, “X” is easily recognisable as removing files.
  - The tutorial symbol is “?” which is also a commonly used symbol for help.
  - We have basic instructions as placeholders in the textboxes to guide users.
- **Control**
  - The adding and removing of files allows users to freely add and remove files, so the users do not need to refresh the page to clear unwanted files.
- **Consistency**
  - We clearly label our textboxes with “known” and “unknown” to indicate to our users what exactly they should put into each textbox and we also show results consistently in this way.
- **Error Prevention**
  - If there are no files added or no text inputted, there will be an error message notifying users of this.
- **Recognition**
  - Most symbols such as “+”, “X”, “?” are easily recognisable, the compare button is a button with the “compare” text on it.
- **Flexibility**
  - Flexibility between adding text input or adding files. Can also mix and match between text / files. Furthermore, .txt, .docx and .pdf file types are all supported.
- **Minimalism**
  - Not much text or elements on screen before users click the “compare” button, only relevant information is needed. Tutorial is only shown if user decides to click “?”.
- **Recover**
  - Our website does not have many errors, the only one being the error message for no file/text input, written in plain English.
- **Help and Documentation**
  - The error message for no file/text input is written in plain language.

UX: how people feel about a product and their impression and how it feels to them and how they feel afterwards

Heuristic Evaluation

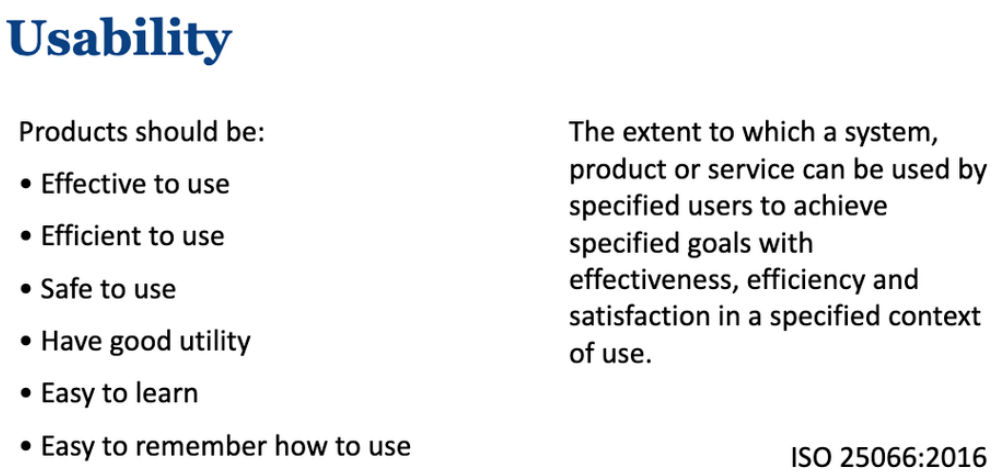
- **Visibility** - should always keep users informed about what is going on (ie appropriate feedback)
  - we should probably add a popup if their analysis runs over a certain amount of time
- **Match** - between system and real world (speak users' language, simple common language, follow real world conventions)
- **Control** - user control and freedom, support undo and redo

- Users often choose system functions by mistake and will need a clearly marked “emergency exit” to leave the unwanted state without having to go through an extended dialogue.

- **Consistency**
  - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
- Error Prevention
  - Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
- Recognition - Make objects, actions, and options visible.
  - (!!) add an ℹ️ button to pull up instructions again when users need it?
  - “Instructions for use of the system should be visible or easily retrievable whenever appropriate.”
- Flexibility - Accelerators - unseen by the novice user - may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
- Minimalism - Dialogues should not contain information which is irrelevant or rarely needed
- Recover - Help Users Recognise, Diagnose, and Recover from Errors
  - error messages should include plain language expression, precisely indicate the problem, construct/suggest a solution
- Help and Documentation
  - Troubleshooting: Make sure your system indicates errors and messages in plain language

In line with user stories, ask questions:

- Will the user know what to do?
- Will the user see how to do it?
- Will the user understand from the feedback whether the action was correct or not?



UX: how people feel about a product and their impression and how it feels to them and how they feel afterwards