Weaving Pathways for Justice with GPT

LLM-driven automated drafting of interactive legal applications

Quinten STEENHUIS [a] David COLARUSSO [b] and Bryce WILLEY [a]

[a] *Suffolk University Law School*

ORCiD ID: Quinten Steenhuis https://orcid.org/0009-0001-0110-064X, David Colarusso https://orcid.org/0009-0003-6287-9284, Bryce Willey https://orcid.org/0000-0003-1775-2869

Abstract. Can generative AI help us speed up the authoring of tools to help self-represented litigants?

In this paper, we describe 3 approaches to automating the completion of court forms: a generative AI approach that uses GPT-3 to iteratively prompt the user to answer questions, a constrained template-driven approach that uses GPT-4-turbo to generate a draft of questions that are subject to human review, and a hybrid method. We use the open source Docassemble platform in all 3 experiments, together with a tool created at Suffolk University Law School called the Assembly Line Weaver. We conclude that the hybrid model of constrained automated drafting with human review is best suited to the task of authoring guided interviews.

Keywords. document automation, generative AI, large language models, forms, form automation, guided interviews, interactive legal applications

1. Introduction

Can generative AI help us speed up the authoring of tools to help self-represented litigants?

The traditional way to help self-represented litigants with court forms is to create a hand-authored interactive legal application, or guided interview. Lauritsen and Steenhuis [1] describe a long list of tools that can be used in this traditional approach, with the most popular tools being HotDocs, A2J Author, and Docassemble. A recent entrant in this space focused on legal applications is JusticeBot [2].

When we say "forms," we mean the full range of documents that litigators and transactional attorneys might work with. These can include: legal complaints, answers to those complaints, deeds, wills, and demand letters. These documents often look like they only need simple fill in the blanks, but require the application of judgment to be used correctly. For example: wills and trusts have clauses that apply only to some classes of testators. Complaints and answers assert the violation of a law, which may or may not be true depending on individual facts. These rules are not always visible within the four corners of the form.

Creating interactive legal applications with traditional tools is slow and careful work. Authors add markup to templates in PDF or Word format, craft labels for each variable,

create pages that place the labeled variables in context, write logical rules to show and hide follow-up questions and text in the final output, and add instructions and help.

Suffolk Law School's Legal Innovation and Technology Lab created a tool to help speed up this task in 2020, during the early months of the COVID-19 pandemic [3]. This tool, the Assembly Line Weaver, scans templates for variables and uses a mix of pre-written questions and heuristics to generate a draft guided interview for the Docassemble [4] platform. The interview itself uses customizable templates for the YAML format that Docassemble uses, and can later be edited by hand. The tool was designed to allow volunteers to help scale the work of expert form automators, and during the pandemic more than 200 volunteers from around the world helped the lab automate about 30 key processes with it from start to finish [3].

Once the hundreds of volunteers left, the lab's work shifted. It didn't make sense for the lab to spend the many hours required to automate each of the almost 800 forms that remained in Massachusetts, even with the Weaver's help. We focused on improving the tools and helping others use them for the most important forms in their own states.

GPT-3 and 4 have inspired computer scientists to experiment with having an AI re-organize legal information. Some works have found success in using GPT-4 to convert other forms of legal knowledge, like legislation, into formal structures [5]. Having an AI do all of the work of forms (scanning a template for variables, asking the user questions, and following up until all variables are defined) is also being debated in hallways at AI and law conferences; touted on Reddit [6]; and discussed as an idea in general interest magazines [7]. We do not think that it is a good idea except for very simple forms. Instead, we suggest a hybrid approach, with a GPT assisting with the first draft and a human editor turning it into a usable interactive legal application. If it succeeds, this approach can lower the cost to automate court and similar forms dramatically, making it realistic for a state to automate hundreds of forms and reap the benefits for self-represented litigants, including a mobile-friendly interface and integration with electronic filing systems.

Some of our experiments are tentative and early. We acknowledge the limits of our approaches so far. But the promise is hard to deny.

1.1. Is the form the thing?

Before we describe our project, we acknowledge the centrality of forms in current approaches to serving self-represented litigants, **and** that forms may not be the best possible approach to serve that goal. Branting [8] has argued that narratives, not forms, are the tools that lawyers best use to structure effective legal claims. Whether forms or narratives are the best presentation, ultimately, what decision makers need are the facts to make a decision.

Despite that, we start our work with the form. We annotate it, add labels, and turn it into a guided interview in an automated way. There are thousands of existing legal forms (at least 25,000 in the US alone [9]) that represent codified legal systems and processes. The work that went into producing the paper form lives in the form's instructions, fields, and checkboxes, ready for the author of an interactive tool to interpret. Even if we want to abandon them eventually, forms are a great place to start.

2. Traditional interactive legal application authoring approaches

An interactive legal application that completes a form has three components:

1. An output **document**, turned into a template with fields.
2. A series of **questions**, organized into one or more screens.
3. A set of **logical rules** that decide which followup questions are needed and which boxes or paragraphs are included in the final output.

Human authors perform 4 tasks related to these components:

1. **Label** fields in the template.
2. Create a brief **prompt** for each field.
3. **Group** fields into a logical order.
4. Add any conditional logic needed.

This work can take **hundreds of hours**. One representative project, Massachusetts Defense for Eviction (MADE) [10], which took about a year to complete, contains:

1. 1,100 lines of Python code
2. 6,141 lines of YAML (or about 30,000 words)
3. 13 documents in Microsoft Word format, totalling 13,000 words

2.1. Interactive legal applications require thoughtful choices that affect usability

While it is tempting to think of the process of creating an interactive legal application from a form as simply letting the user "fill in the blanks," the process requires judgment in several areas. Jarrett and Gaffney's text Forms That Work [11] discusses some of those choices, including: the order of questions, how to provide help, and how to respect the user of your form. The UK's National Health Service [12] and SurveyMonkey [13] have similar guidance. Our lab developed a detailed style guide with guidance for the authors of legal apps [14]. We also describe these choices that affect burden in our paper RateMyPDF [9].

Some of these choices can be made once and repeated again and again. For example: we ran usability tests with our questions that ask for a participant's name. We researched and applied best practices to gather a litigant's pronouns and gender. Through practice, we determined an optimal structure for a guided interview that completes a court form. We built the Assembly Line Weaver tool to help us implement these and other choices consistently across each form that we turned into an interactive legal application.

3. The Assembly Line Weaver

The Assembly Line Weaver (Figure 1) is a template-driven guided interview that produces drafts of guided interviews in the YAML format. It strikes a balance between the ease of use of fully graphical authoring environments and the power and flexibility of Docassemble.

The Weaver:

1. Scans a pre-prepared Word or PDF template for field names.
2. Asks for metadata about the form.

Figure 1. The opening screen of the Assembly Line Weaver

3. Allows the author to customize the location, branding, and template used to author the interview.
4. Asks the author to provide "before and after" context to the form, including next steps the user will need to take.
5. Allows the author to add a prompt for each field.
6. Asks the author to assign fields to screens, with any extra context that is needed.

In the context of the Weaver, "template-driven" means that we have created a YAML file that has the basic structure of a Docassemble interview. The provided template includes the following screens:

1. A title screen.
2. A "before you start" screen.
3. Questions, both those created by the author and from the question library for information like party names, pronouns, addresses, and more.
4. A preview screen.
5. A "review" screen that allows the user to edit their responses.
6. A signature screen (with ability to text or email the document for signature by a third party).
7. A download screen.

The output of the Weaver is a .ZIP file that includes the customized YAML file, the output template, and a template representing the "Next steps" document that can all be further customized. The Weaver does not assist the author with adding logic (or rules) to the guided interview that they produce.

4. A vision for reducing human effort in form automation

At our lab, we were interested to see if generative AI could help reduce the human labor needed to create good guided interviews. Ultimately, we prototyped the automatic adding of labels to fields, drafting of prompts, and grouping questions with the assistance of GPT-3 and GPT-4 turbo. We experimented with using ChatGPT+ to convert laws into

Python code, but because identifying the statute often requires research outside of the form, we do not plan to integrate this with our automated authoring approach.

In order to maximize our team's productivity, we took two approaches to two unique document types: with Word documents, we focused on identifying and labeling the placeholder variables, which was a significant gap in our existing Weaver tool's capabilities. We left room for these labels to be fed directly into the Weaver in the future, which would take advantage of the structure of our template-driven guided interview authoring. With PDF documents, we prototyped a solution that directly authors a limited version of a Docassemble interview in YAML format. This let us quickly experiment with the interview flow that a large language model would produce. While this still allows for human editing and review, it doesn't take advantage of the existing work we have done with the Weaver to standardize the output. Ultimately, we expect to combine the two approaches.

5. Auto-labeling of Word documents

PDF and Word (DOCX) documents have some key differences: PDFs are usually final documents, meant to be printed and filled in by hand at least some of the time. That means that they almost always include lines, boxes, and circles where the user's answer goes. These features can be detected with traditional computer vision tools like OpenCV [15].

In contrast, Word documents can be edited in a word processor before they are printed and filed. Word documents are more likely to include placeholder text, like [CLIENT NAME], than blank spaces marked by lines or boxes. While one document might consistently use square brackets, another might mix in curly brackets, or leave out the placeholder marker altogether in favor of in-line text like "Your name" or "Dear Merchant." This means that totally rules-based systems, like regular expressions, do not perform well at identifying and labeling placeholders. Statistical approaches, enabled by large language models, can better handle the many variations.

Separately, it was important that whatever method we adopted for labeling the Word document did not disrupt the existing formatting of the document. These features include paragraphs, automatic numbering, and bold and italics, features that are not present in plain Unicode text. That meant the naive approach of sending just the text of the document wouldn't work. Internally, a Word document is an XML document, which **can** be represented in plain text. However: sending the full uncompressed XML of the document would almost always exceed GPT-4's context window of 4,096 tokens. Although GPT-4 turbo expanded the context window for inputs, the context window for replies remains 4,096 tokens.

We did not want totally free-form labels for variables in our documents. We wanted the automatic labeling to follow some basic rules:

- Variables have to be valid Python identifiers, and follow Python conventions like using underscores and lower case letters.
- We have a pre-defined set of nouns, like **users**, **other_parties**, **attorneys**, and more that we preferred to use when applicable, and we always use these nouns as members of a list.

- We have a set of predefined attributes for a person, like **name**, **address**, and so on that we wanted to use exactly the same way whenever collecting names, addresses, or similar common attributes.

We include these variable naming conventions in our prompt to the model.

A Python notebook with our code can be found in our GitHub repository: `https://github.com/SuffolkLITLab/FormFyxer/blob/docx-fields-experiment/explore_labeling_docx.ipynb`

5.1. Tokenizing the input document

To label our Word document, first we use the open source python-docx [16] library to extract the paragraphs and **runs** of formatted text in each paragraph. These indices in the Python representation can be referenced and used to modify the formatted document later.

Next, we turn the paragraphs and runs of text into a JSON structure, like this:

```
[
    [0, 1, "Dear John Smith:"],
    [1, 0, "I am writing to claim ..."],
    [2, 0, "[Optional: if ...]"],
]
```

Where each entry in the JSON list is a 3-tuple of paragraph number, run number, and the text of that run. Note: this approach does not yet handle tables, which sometimes appeared in our dataset.

5.2. Prompting the large language model

Our first attempt was to instruct the model to reply with a JSON structure that includes a tuple of the paragraph number, run number, the Jinja2 variable, and the starting and ending position of the text to be replaced. We also instructed the model to reply with only the modified text, rather than returning the full modified document. In GPT-3.5-turbo, this prompt ignored our instruction to return only the modified runs, instead returning all of the text of the modified document in the reply. In both GPT-3.5 and GPT-4, this prompt failed to accurately mark the start and end position of the text in a way that allowed us to insert Jinja2 variables in context. When we used the positions that were returned by the model, the Jinja2 variables would be inserted haphazardly in the middle of the existing placeholder text, leaving a messy output document that required a lot of manual cleanup. See Figure 2.

A more successful approach (that also used more input tokens) was to prompt the model to return the full text of each modified run. We then iterated through the return value to replace each run with its modified text. Because a run in the DOCX format all has the same formatting, we could safely replace the text of the original run with the modified run. This worked even with very idiosyncratic markup in the example document, as shown below.

Our revised prompt succeeding in identifying the placeholder text and replacing it in full. See Figure 4.

Superior Court of Washin{{ trial_court }}gton, County of {{ trial_court.address.county }}

In the Guardianship of:	
	No. _____
_____	Notice of Hearing about Emergency Minor Guardianship Petition
Respondent/s *(minors/children)*	(NTHG)
	Clerk's action required: **1**
	[ ] Interpreter required in: _____
	(language)

Notice of Hearing about Emergency Minor Guardianship Petition

To:{{ interested_parties[0] }}The {{ parents[0] }}arents, {{ children[0] }}hildren, {{ guardians[0] }}uardian, {{ person_with_court_ordered_custody[0] }}erson with court-ordered custody, {{ court_clerk[0] }}ourt clerk, and all people who must get notice:

Figure 2. Example of failed GPT-3.5 prompt, with Jinja2 variables inserted in the middle of placeholder phrases

Your name
Your address
Your telephone number

Date
Name of Merchant
Merchant's address

Dear Merchant:

Under the provisions of Massachusetts General Laws, Chapter 93A, Section 9, I hereby make written demand for relief as outlined in that statute.

On or about **{date}**, the following unfair or deceptive act occurred:
{EXPLAIN WHAT HAPPENED}

This unfair or deceptive act or practice is, in my opinion, declared unlawful by Section 2 of Chapter 93A, (you may want to give regulation number, if applicable) which reads as follows:
{Quote text or section. Remember: You are not required to quote written regulations or laws to support the assertion that the merchant's conduct was unfair or deceptive; it is, however, desirable. You will want
to include all the regulations which you believe were violated.}

Figure 3. An example document from the wild, demonstrating a wide variety of strategies to mark placeholder text within a single document

This approach appeared to perform well across a range of documents, a sample of which can be viewed in the GitHub repository. We did not, however, have a large set of pre-annotated documents to benchmark this performance.

5.3. Using the output to build an interactive legal application

Once the Word document is labeled, it can be uploaded to the Weaver. While the author can then assign questions manually for each field, the author can also choose to use our "auto drafting mode," which leverages heuristics and a traditional machine learning approach, but requires a lot of human editing. We describe this existing approach in our paper describing RateMyPDF.[9]

6. Building interactive legal apps automatically with PDF documents

Our experiment with PDF documents authors Docassemble interviews in YAML format without any human intervention. With important limits, our approach was successful.

Figure 4. Example of successful output with the revised prompt

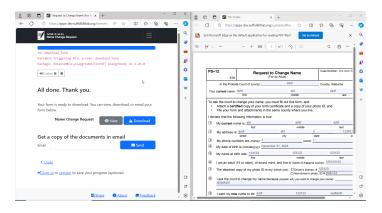**Figure 5.** The first page of an automatically generated name change interview

Figure 6. Final screen of the automatically generated name change interview, side by side with the completed PDF

We generated contextually appropriate labels, questions, and created screens for each question. (Figure 5) The interviews ran and filled in fields in the PDF document. (Figure 6) But the PDF's stream-based format made it difficult to accurately identify and label all fields, particularly checkboxes. We describe our success rate and the different ways we worked around this limit below.

We tested this approach with name change forms taken from 12 jurisdictions. We were able to automatically recognize and write questions for 62% to 69% of the existing

fields. On the best performing form, we were able to do so for 93% of the fields, compared to 27% on the worst performing form. For the fields that we could not identify, the problem stemmed from difficulties placing them in context as part of the text provided to the GPT.

All of the code, along with our input and output documents, for this section can be found in an interactive Python notebook, presented here: `https://colab.research.google.com/drive/1HRv5imbtHIVYt37h4liwFkW9YlPUNNKI?usp=sharing`.

6.1. Finding the fields in context

We wanted to feed the GPT model the full text of the form, with the field's placeholders in the right spot in the text, so that the model could come up with appropriate names, just like we did for DOCX files. Ultimately, we found an approach that succeeded most of the time, and we conclude this is a potential place for either further engineering work or adding a human in the loop.

First, we experimented with the native PDF feature of ChatGPT+. As of this writing, you can directly upload a PDF to ChatGPT+ and interact with its contents. We were able to ask ChatGPT+ what fields it thought a PDF of a court form had and get back an answer. However, this feature interacts only with the text, and not the form layer of the PDF. In some cases, it identified fields in the PDF that were not in the form layer, because they were designed to be completed by the court clerk (for example, a "case number" field). In addition, it cannot be used to add text to the PDF at all.

Next, we experimented with text-only approaches. It is possible to get the full text of a PDF with existing open source tools, but this text does not contain any representation of the form layer and its fields. The PDF file format is stream-based, with field elements occupying a stream independent of the PDF's text. Each field element has a defined position on the page, and so does the text, but understanding the relative position of the two requires rendering the PDF, particularly if, as is often the case, there is one large text object in the PDF, with only whitespace added to make room for the PDF fields.

The final, complex, approach that we found was the most reliable was to:

1. fill each field in the form layer with placeholder text (e.g, field_01)
2. convert each page of the PDF into a PNG image
3. apply optical character recognition (OCR) to the images to get the placeholders in context (e.g., "Your name: field_01").

This approach worked well for fields where the user was expected to write an answer, but it often failed for checkboxes. They were simply too small to add OCR-visible text to. To address this, we did not label small fields in step 1. We reintroduced checkboxes and other small fields when asking the large language model to generate questions, described below.

6.2. Leveraging context to create variable names, definitions, and questions

Given a string for the form's text with variable placeholders in-line, it is possible to have a large language model create appropriate variable names, along with definitions and questions. Building upon the output of step 3 from Section 6.1, we can:

1. use an LLM to generate a name and description for the document;

2. use an LLM to change placeholder names into semantically appropriate names (e.g., first_name) based on the output string;
3. use an LLM to write definitions for said variables, again based on the above string;
4. leverage these definitions and an LLM to author questions for users and guess at their data types; and
5. convert these questions into a Docassemble interview by writing a YAML file that takes user answers and fills them into a processed version of the original PDF.

Checkboxes are handled specially by our prompt. Because we skip checkboxes at our form filling step, at this stage we prompt the large language model to try to assign each checkbox a label and definition based on any text element adjacent to the field on the PDF. This approach could only pair checkbox fields with related text 28% of the time.

We note that checkbox fields can be important on court forms. The user can ask for relief, identify a legal claim in a complaint, or exercise their right to a jury trial by checking the appropriate box. Therefore, while the OCR method we describe has clear promise, an important future task is to improve our accuracy with checkbox fields.

6.3. Validating user interactions

We explored two methods to add input validation in our generation of interviews for PDF forms: a "large language model in the loop" when the user interacted with the form, and an approach that asked the model to classify the input's data type when it first parsed the form.

The "large language model in the loop" approach, while approaching a natural conversation, was resource intensive. After every response by a user, we asked GPT if the answer responded to the question. If not, the user was told why and asked to answer again. In addition to being expensive, it also risked annoying users by asking too many questions. In testing, it handled redirecting answers like "how is the weather today?" when presenting the user with the question "What is your full name?" well. But we were concerned that the model could be too rigid. For example, would it refuse valid but unusual address lines? What about unusual names, like X? Given the risk of offense and the cost, we abandoned this approach early on.

The up-front classification of answers into datatypes also had risks. For example, phone numbers were often assigned a numeric data type, which precluded users from entering phone numbers of the form 555-5555, accepting only 5555555. ZIP codes were assigned a numeric type too, which strips leading 0's from ZIP codes, causing Massachusetts's ZIP codes to appear incorrect when output. While human review could catch these validation failures, using standardized questions wherever possible, like the Weaver, would cut down the amount of review an author needed to perform.

6.4. Additional Limitations & Potential Points of Intervention

The section above does not address the handling of fields that were neither in the collection of potential checkboxes nor those for which a new name and definition could be made. Only 14% of the fields in our sample of forms couldn't be placed in-line with the OCR method or identified as a potential checkbox. On the best performing form, every

field was either placed in-line or identified as a check box. On the worst performing form, 28% of the fields remained unidentified.

The authors of Docassemble interviews can examine and edit the output of the final YAML file to correct any errors and improve the usability of the generated interview. But in the best-case scenario the full document can be completed. This was most likely to be true when the original PDF had few or no checkboxes.

We noticed that some errors could compound. For example: a missing or incorrectly labeled field would lead to an incorrect question and an incorrect series of screens. Early review, at multiple stages in the automation, could simplify the author's final editing task.

We identified 2 helpful intervention points:

- Before the the title, description, variable names and definitions are finalized
- Before questions and data types are finalized

We prototyped this review process by allowing the author to edit a JSON object in our Python notebook at each stage.

7. Reaching level 1

In our paper Digital Curb Cuts [3] we described a maturity model for guided interviews, with a level 1 form being at least as good as the experience of completing the original PDF, and level 4 representing a highly polished user experience. Many forms, now locked up in PDFs, would benefit simply from the increased usability of a responsive, mobile friendly design that can be integrated with electronic filing. Our experiments with GPT show that achieving this basic level 1 of automation is possible with a large language model and some Python code alone.

7.1. Weaving the two approaches together

We had better success with identifying and labeling fields in Word documents than PDFs, despite the more ambiguous markers that the Word documents used to label placeholders (e.g., see Figure 3). But we only used the GPT model to draft questions and put them in order with PDF documents. It's likely that the full interview authoring approach that we took with PDF documents would succeed with labeled Word documents. We could also continue to experiment with solutions to the context problem with PDF checkbox fields.

We think the best way to approach both experimental features is to integrate them with our existing Weaver tool. The unconstrained LLM authoring approach we took with the PDF automation would be enhanced with the template-driven and shared-question approach used in the Weaver. This would limit the need for auto-generated definitions and questions to only those field types not in the Weaver's library. We could get the best of both worlds: vetted, usability tested questions for common fields, and good drafts of the unique questions drafted by the large language model.

8. Conclusion

Generative AI can identify user inputs and create interview questions for a wide variety of forms. It can work with both DOCX and PDF formats. For the very simplest forms,

especially those without very much conditional logic, large language models can help to produce draft automations that require almost no human intervention at all. However, this AI cannot reliably automate the large number of more complex legal documents without human assistance. Human review and editing of these interviews remains essential. Integrating this human review flexibly at different points in the automation process allows humans to intervene as much or as little as needed, allowing humans to adjust to the needs of a wide range of input documents.

The hybrid approach of automated drafting of a traditional guided interview can get the time saving benefits of the large language model without the risks of fully automated question and answer driven form-filling. It has the potential to significantly expand the number and kind of forms that can benefit from document automation.

References

[1] Lauritsen M, Steenhuis Q. Substantive Legal Software Quality: A Gathering Storm? In: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law. Montreal QC Canada: ACM; 2019. p. 52-62. Available from: `https://dl.acm.org/doi/10.1145/3322640.3326706`.

[2] Westermann H, Benyekhlef K. JusticeBot: A Methodology for Building Augmented Intelligence Tools for Laypeople to Increase Access to Justice. In: Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law. ICAIL '23. New York, NY, USA: Association for Computing Machinery; 2023. p. 351-60. Available from: `https://doi.org/10.1145/3594536.3595166`.

[3] Steenhuis Q, Colarusso D. Digital Curb Cuts: Towards an Open Forms Ecosystem. Akron Law Review. 2021;54(4):2. Available from: `https://ideaexchange.uakron.edu/akronlawreview/vol54/iss4/2/`.

[4] Pyle J. Docassemble; 2021. Available from: `http://docassemble.org/`.

[5] Janatian S, Westermann H, Tan J, Savelka J, Benyekhlef K. From Text to Structure: Using Large Language Models to Support the Development of Legal Expert Systems; 2023.

[6] talktothelampa. How to use LLM in order to fill an intake form? [Reddit Post]; 2023. Available from: `www.reddit.com/r/LangChain/comments/140jhln/how_to_use_llm_in_order_to_fill_an_intake_form/`.

[7] Broadway M. Can ChatGPT Fill Out Forms? Yes, here's how; 2023. Available from: `https://www.pcguide.com/apps/can-chatgpt-fill-out-forms-yes-heres-how/`.

[8] Branting K, McLeod S. Narrative-Driven Case Elicitation. Workshop on Artificial Intelligence for Access to Justice (AI4AJ 2023). 2023 Jun.

[9] Steenhuis Q, Willey B, Colarusso D. Beyond Readability with RateMyPDF: A Combined Rule-based and Machine Learning Approach to Improving Court Forms. Proceedings of International Conference on Artificial Intelligence and Law (ICAIL 2023). 2023:287-96.

[10] Steenhuis Q. Making MADE: User-centered Design in Practice – Quinten Steenhuis; 2019. Available from: `https://www.nonprofittechy.com/2019/05/12/making-made-user-centered-design-in-practice/`.

[11] Jarrett C, Gaffney G, Krug S. Forms that Work: Designing Web Forms for Usability. 1st ed. Amsterdam ; Boston: Morgan Kaufmann; 2008.

[12] How to write good questions for forms - NHS digital service manual;. Available from: `https://service-manual.nhs.uk`.

[13] Survey Monkey. Smart Survey Design. Survey Monkey; 2008. Available from: `https://s3.amazonaws.com/SurveyMonkeyFiles/SmartSurvey.pdf`.

[14] Writing good questions | The Document Assembly Line Project;. Available from: `https://suffolklitlab.org/docassemble-AssemblyLine-documentation/docs/style_guide/question_overview`.

[15] Bradski G. The OpenCV Library. Dr Dobb's Journal of Software Tools. 2000.

[16] python-docx — python-docx 1.1.0 documentation;. Available from: `https://python-docx.readthedocs.io/en/latest/`.