INTERNATIONAL COMPUTATIONAL LAW JOURNAL

CODEX SOURCE ARCHIVE

Software is Scholarship

Houman B. Shadab

Computational Law Report · Release 2.0 · 2020-11-20

Reader edition prepared by The Research University under CC BY 4.0. This cover was added; the original PDF pages follow unchanged. Original authorship, publication identity and source links are preserved. This independent republication is not sponsored or endorsed by Stanford University, Stanford Law School, CodeX or MIT.

Volume 1 · Issue 2 · Systems, Rules and Public Infrastructure

Original source and license evidence are recorded on the article landing page.

MIT Computational Law Report

Software is Scholarship

Houman B. Shadab

Published on: Nov 20, 2020

License: Creative Commons Attribution 4.0 International License (CC-BY 4.0)

0. ABSTRACT

This Article provides the first systematic account and justification of software applications as works of scholarship. Software is scholarship to the extent that software functionality is derived from scholarly research, software is used as a means to develop scholarship, or software is used as a medium to communicate scholarly ideas. Software applications are superior to articles and books for communicating scholarly ideas because software is not limited by the constraints of traditional written works. Software can communicate using a wide variety of textual components, graphical elements, and programmable interactivity that significantly enhance the ability to communicate scholarly concepts, arguments, and findings.

This Article identifies four methods for software applications to enhance scholarly communication: app-ified argumentation that provides theoretical clarity, interactive toolkits that create rich qualitative studies, data visualizations that persuade using data, and policy tech that improves the ability to enact social change. Interactive software applications can enhance research agendas in the humanities and social sciences by making traditional, prose scholarship more thorough, persuasive, and analytically precise. Due to recent innovations, developing software for scholarly purposes is accessible to those that work in the humanities. Platforms for developing software have grown so sophisticated that they no longer require creators to write code to develop powerful, data rich, and well-designed interactive applications. Scholars should accordingly use and develop software to better communicate their ideas. By providing a framework for developing software as works of scholarship, this Article contributes to the field of digital humanities.

To better understand this Article's concept of scholarly software, I apply my conceptualization of scholarly software to legal scholarship and legal technology and discuss three case studies: LegalTech toolkits, voice recognition for automated contract drafting, and court data visualizations. Law is a fertile ground for the development of scholarly software because the core of legal reasoning consists of a formalistic, computational structure that is well-expressed through programmable applications. This Article contributes to legal scholarship by identifying how it can be enhanced through the creation of software applications.

1. Introduction

This Article provides the first systematic account and justification of software applications as works of scholarship. Software is scholarship to the extent that software functionality is derived from scholarly research, software is used as a means to develop scholarship, or software is used as a medium to

communicate scholarly ideas. This Article focuses on software applications as works that communicate scholarship.

Expressing scholarly ideas in the form of software applications can enhance research agendas in the humanities and social sciences by making traditional, prose scholarship more robust, persuasive, and analytically precise.[1] Interactive software applications are superior to articles and books for communicating scholarly ideas because software is not limited by the constraints of traditional written works and can communicate using a wide variety of textual components, graphical elements, and programmable interactivity that significantly enhance the ability to communicate scholarly concepts, arguments, and findings. Scholars should accordingly use and develop software to better communicate their ideas.

I identify four methods for software applications to enhance scholarly communication:

- App-ified argumentation that provides theoretical clarity,
- Interactive toolkits that create rich qualitative studies,
- Data visualizations to persuade using data, and
- Policy tech that improves the ability to enact social change.

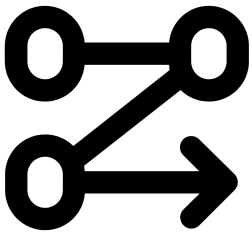
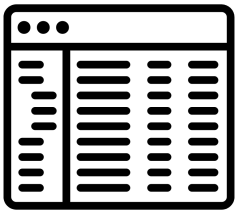
By providing a framework for developing software as works of scholarship, this Article contributes to the field of digital humanities. Digital humanities as a field includes the application and study of digital technology in the humanities, examples of which include creating digital art, digitizing scholarly archives, and making available the entire corpus of Shakespeare's works to be analyzed programmatically.[2] Using software as a tool to digitize scholarly materials, apply quantitative methods, create data visualizations, and produce other data-oriented scholarship is an essential and well-recognized activity within the humanities and social sciences.[3]

However, using software applications as a medium to communicate scholarly insights is generally unrecognized and insufficiently practiced.[4] On the one hand, works in the digital humanities are typically presented as stand-alone projects without a full scholarly treatment.[5] On the other hand, traditional scholarly publications are only able to present software-driven graphical elements as static, non-interactive images and must do so within the confines of fixed margins and a linear exposition structure. The benefit of scholarly software applications is that they can have the best of both worlds. Software applications can combine software-generated graphical and interactive components with detailed scholarly prose to create the most effective means of scholarly communication possible. In advocating for the use of software to communicate, this Article follows the tradition of literate programming. Literate programming views programmers as essayists that communicate with code and the other tools that software makes available.[6]

To better understand this Article's concept of scholarly software applications, I apply my conceptualization of scholarly software to legal scholarship and legal technology and discuss three case studies: LegalTech toolkits, voice recognition for automated contract drafting, and court data visualizations. Law is a fertile ground for the development of scholarly software because the core of legal reasoning consists of a formalistic, computational structure that is well-expressed through programmable applications. This Article contributes to legal scholarship by identifying several ways in which LegalTech applications can qualify as legal scholarship.

As represented in Figure 1, software applications that are used for scholarly purposes should be conceptualized as consisting of three primary components:

- a user interface (UI) that displays text, graphics, and other visual media and enables users to interact with an application,
- programmable logic that enables the application to perform computable tasks, including tasks initiated through the UI, and
- a database that contains the application's data.[7]

> ***Figure 1: Primary Components of Software Applications***
>
> | User Interface | Programmable Logic | Database |
> |---|---|---|
> |  |  |  |

An example of incorporating interactive software elements into a traditional work of scholarship comes from online chapters of Stéfan Sinclair and Geoffrey Rockwell's book Hermeneutica. The screen on the left side of Figure 2 shows a data visualization within the authors' textual analysis of David Hume's Dialogues Concerning Natural Religion.[8] The screen on the right side displays additional data analysis of the word "natural" in Hume's text after a user clicks on the word "natural" in the data visualization within the chapter.[9] These software elements enable the authors to communicate more precisely about how Hume uses the concept of skepticism throughout his work.

> ***Figure 2: Example of Scholarly Software***

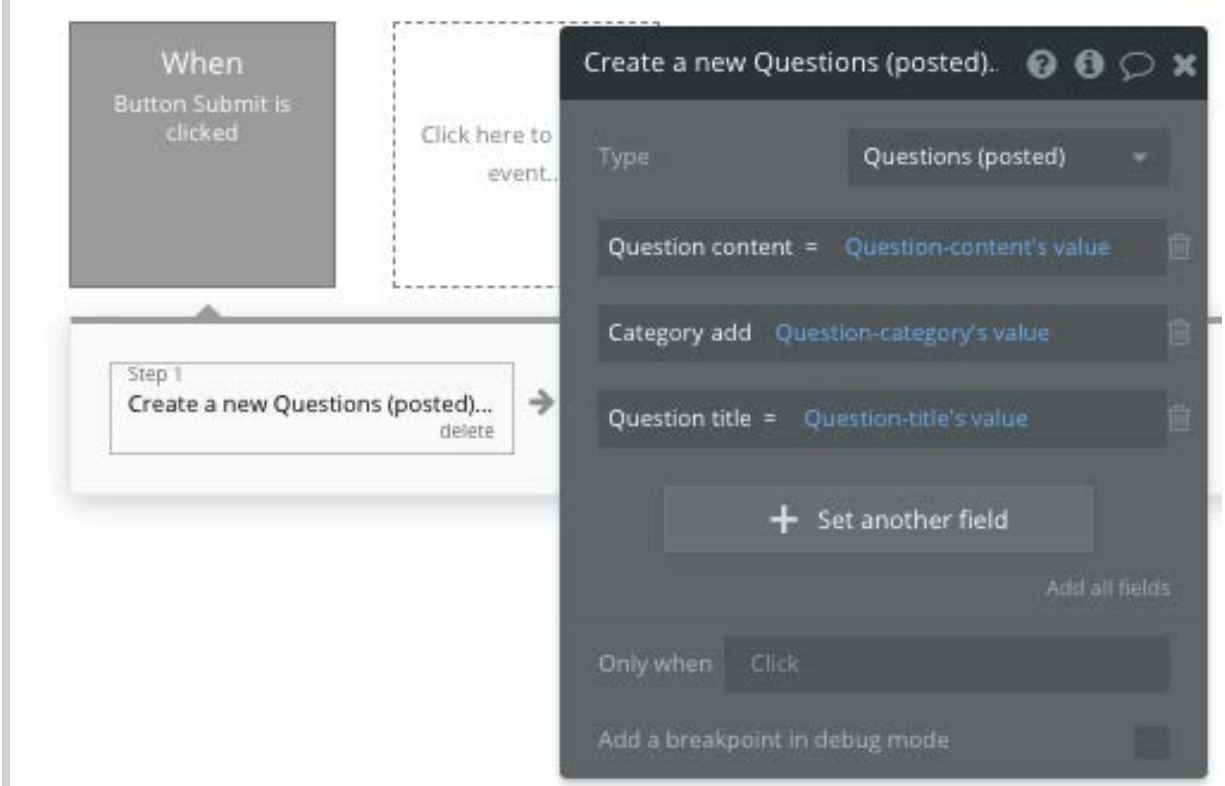
Interactive Data Visualization in Book Chapter	Analysis of "Natural" in Hume's Text
	

As discussed below, scholarly software applications can do much more than incorporate data visualizations into otherwise traditional text.

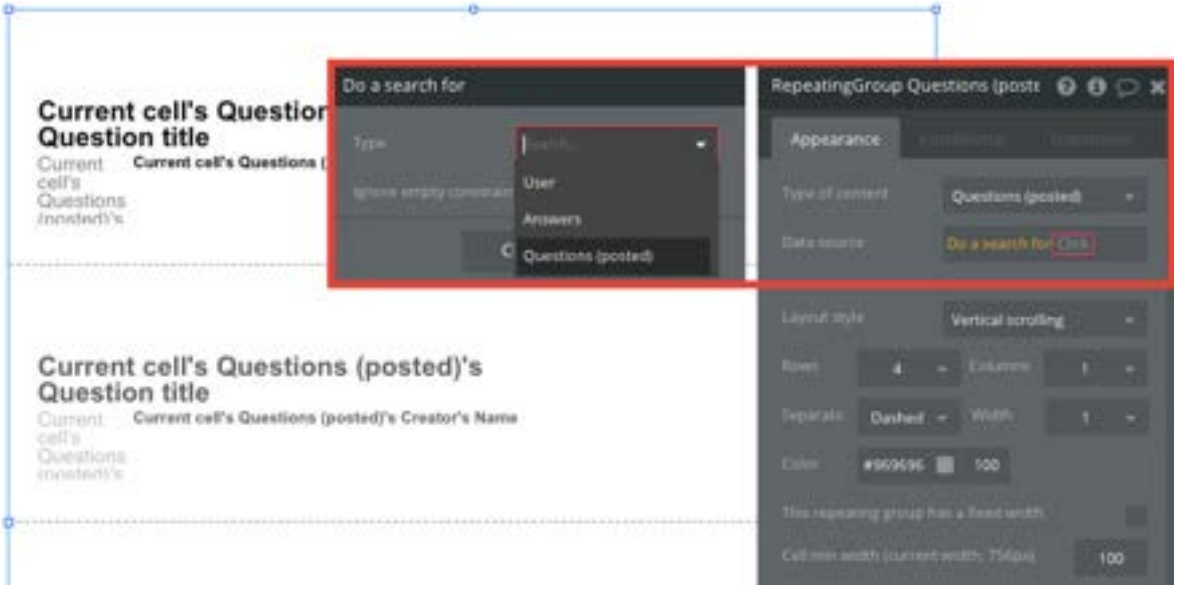
Historically, programming to create scholarly software would require significant expertise in writing software code. However, due to recent innovations, platforms for developing software applications have become so accessible and sophisticated that they no longer require creators to write code in order to create powerful, customized, well-designed, and interactive applications—including those that further scholarship. Visual software development uses drag-and-drop and other graphical elements to build an application's user interface, programming logic, and underlying database. Powerful machine learning, data science, and data visualization tools are also becoming increasingly accessible without being required to write code.[10]

Figure 3 shows a very simple example of programming using the Bubble visual development platform.[11] The top part of Figure 3 shows how to visually program software to create a new database entry when the "Submit" button is clicked. The bottom part of Figure 3 shows how to visually program software to dynamically display the data from the database in a user interface with completely customizable graphical elements. The elements shown in Figure 3 replace the need to write code.

> ***Figure 3: Visual Software Development***
>
> Programming the Logic of a Database Entry



Displaying Dynamic Data in a User Interface



Bolstering the significance of widely accessible visual software development is the fact that publicly available data has also since proliferated; moreover, technology that enables data platforms and software applications to connect with each other is now standardized.[12] Accordingly, due to highly accessible visual development, data sets, and software integrations, scholars with no ability to write

software code and without any specialized technical training can create a wide variety of very powerful software applications and tools to further their scholarship.

The dramatic increase in the accessibility of powerful software development is poised to unleash a wave of technology-enabled creativity in the humanities and social sciences. As noted by The No Code Revolution white paper published by Webflow:

> Those with backgrounds in the humanities tend to think of problems differently. Efficiency is rarely much of a concern. Quality, affect, and experience tend to dominate their thinking. So you have to wonder: what would a comp lit major build, if they were to build software? How would a philosopher, or environmental lawyer, tackle technological challenges?[13]

By removing code, visual programming narrows the gap between natural language and programming language.

This Article proceeds as follows: Section II explains the essential aspects of computer science, software engineering, databases, and web applications that are relevant to the creation and functioning of scholarly software. Section III develops a theory of software applications as scholarship and explains four methods that can be used by scholarly software. Section IV explains the development and significance of visual software development platforms, open data, and software interconnectivity. Section IV also explores how to use visual development to create scholarship based upon the core components of user interface, programmable workflow logic, and database systems. Section V applies this Article's arguments to legal scholarship and legal technology. Section VI concludes.

2. Software Foundations

This Section explains the foundational components of software that are relevant to understanding the creation and operation of scholarly software. It also highlights implications of particular importance for software that is used to communicate. These foundational components of software are drawn from computer science, software engineering, database technology, and web applications.

A. Computer Science Concepts

In developing scholarly software, a scholar must program software to instruct it to perform various computational tasks. These tasks include displaying text and graphics, changing what is displayed in response to user inputs, and data processing.

A software program is a set of rules or step-by-step instructions executed by a computer to perform a task. These programming tasks produce an output given some input.[14] Computer science is the study of how to program software.[15] This includes the development of concepts and techniques programming (such as algorithms) and how to structure data within the context of computer

programs.[16] Programming rules include simple arithmetic, defining variables, and rules about the order in which computational tasks should take place.[17]

Programming languages are more powerful to the extent that, in computer science terminology, they are more "expressive." Being more expressive means that a programming language has greater capability to capture and communicate a broader range of ideas than other languages.[18] Being expressive includes being able to create programs that can be defined in terms of other programs which has the benefit of increasing a program's level of abstraction, modularity, and ability to communicate concepts and rules.[19] Visual programming languages and the ability to interconnect software applications to each other vastly enhances the expressiveness of software and corresponding ability of scholars to express their own ideas.[20]

A common activity of programming is instructing software to undertake operations with respect to lists of data. These include sorting,[21] searching,[22] combining, and displaying data contained in lists. Performing operations with lists is common because data is often organized into long lists. Indeed, a key benefit of computing is to be able to perform tasks with respect to long lists of data that would otherwise be practically impossible to be performed manually.

Certain data types and programs constitute fundamental building blocks of software applications, including scholarly software. These data types include integers, decimals, strings, booleans (that evaluate to a true or false), and date/time.[23] In addition, a core aspect of programming is defining the strict, logical patterns that a program will follow. This includes defining conditional expressions that perform certain tasks only when certain things are true.[24] Simple conditional logic may be implemented with if-then logic or binary (boolean) logic such as mathematical operators (>, <, =,). Complex conditional logic may be implemented using programs that continue to run until a pre-specified value is found.[25] Logical operators such as and, or, and not are also ubiquitously used to create conditions. Likewise, programs often make it so that certain outcomes or properties of the program are true so long as certain conditions are true by implementing while logical operators.[26] Programming also often entails using repetitive or continuous loops that execute the same rule over and over again until a desired result is achieved. Programs that engage in repetitive tasks according to pre-planned rules and patterns form the basis for algorithms that are able to automate decision making and other tasks with high accuracy and in extremely high volumes.[27]

Scholarly software will likely seek to communicate a complex set of concepts and arguments. To deal with a corresponding increase in software complexity, programming techniques are used to make programs internally organized and modular so that "they can be divided 'naturally' into coherent parts that can be separately developed and maintained."[28] One method of building modularity is to group different parts of a program into objects with their own values assigned to them that depend on how

the objects interact with each other over time.[29] These objects can be abstract concepts (including scholarly concepts) or correspond to a natural category such as a city or an employee.[30] Organizing the components of a software program into objects that track the subject matter of a work of scholarship supports the ability of a software application to communicate scholarly ideas. This is especially true when the objects are visual programming objects and represent various concepts or propositions.

Computer program rules are not generic but rather have a form (syntax) and meaning (semantics) that are specific to a programming language.[31] Unsurprisingly, different languages are better suited for certain tasks, such as Javascript for web applications and Python for machine learning.[32] Programming languages also differ by their level of abstraction away from binary 0s and 1s into code that comes closer to resembling natural language. Visual programming languages operate at the highest level of abstraction and do not require creators to write any code.[33] Visual programming accordingly facilitates using software to communicate scholarly ideas because its programming components represent natural language concepts and propositions.[34]

As there is typically more than one way to create a program to obtain a desired output, a key consideration in computer science is determining the best way to obtain the desired output from a speed, efficiency, reusability, simplicity, or other perspective.[35] In developing scholarly software, a key consideration is accordingly how to properly plan and design the application. For example, in creating an application that displays connections between the works of several novelists, a key consideration may be the extent to which the works are classified according to static or dynamic categories. This is because a static classification system may be programmed more rapidly and be less prone to error, whereas a dynamic classification system may be more adaptable to change over time. Choosing among the many alternative ways to program software often involves principles of software engineering.

B. Principles of Software Engineering

This section identifies key principles of software engineering for scholars to understand when developing software. Software is a broader set of phenomena than a computer program. As noted by Ian Sommerville: "software is not just the programs themselves but also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful."[36]

Software engineering is distinct from computer science in that its focus is more systems-oriented and normative. According to Sommerville's leading text Software Engineering, whereas "[c]omputer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software."[37] Importantly,

> [c]omputer science theory…is often most applicable to relatively small programs. Elegant theories of computer science are rarely relevant to large, complex problems that require a

> software solution.[38]

The normative criteria of software engineering are to develop software that is functional and performant, maintainable, reliable, secure, and user friendly.[39] Given the diversity in computing systems and programming languages, "there are no universal notations, methods, or techniques for software engineering because different types of software require different approaches."[40]

The process of software development generally consists of specification, program development, user validation, and continuous evolution.[41] The waterfall approach to development requires each of these phases to be completed before proceeding to the other. By contrast, the incremental (or "agile") approach develops an initial version and then, through an iterative process, incorporates user feedback while continuing to engage in specification, development, and validation.[42] Developing scholarly software that is used to communicate should use agile methods. Agile approaches enable developers to incorporate feedback from other scholars as part of a scholarly ideation and comment process.

The development of scholarly software can also benefit by incorporating user requirements. To develop software specifications, developers must document user and broader systemwide needs and goals.[43] Typically, the development of requirements is an iterative process that is responsive to user feedback.[44] User stories (or scenarios) are a standard way to develop requirements tied to the needs of a specific user type or persona.[45] Users of scholarly software are likely to include traditional consumers of scholarship as well as broader set of users that may be drawn to use the software due to its graphical and interactive elements. Accordingly, scholarly software applications should be developed with the intellectual needs of its users as the application's requirements. These scholarly user requirements may include particular gaps in knowledge that users desire to be filled or particular methods of communication (e.g., a specific data visualization).

A key component of the planning and management of software is architecting. Software architectural design uses models to "identif[y] the main structural components in a system and the relationships between them"[46] The benefit of architecting software is to improve "performance, robustness, distributability, and maintainability of a system."[47] Architecture diagrams may incorporate object-oriented programming by showing how each of the important types of objects in a system are organized and what data and programs are associated with the objects.[48] This type of modeling and organization of a scholarly software application assures that it can be efficiently maintained and changed. Software applications are not restructured as easily as a written document.

C. Database Systems

Creating scholarly software involves organizing information relating to the subject of the work in a digital format—as data. This is because software operations that involve the use of information necessarily involve the storage, processing, and display of data. Not surprisingly, "computer scientists have developed a large body of concepts and techniques for managing data."[49] Accordingly, a foundational aspect of software development is establishing how data is structured, stored, processed, and displayed by a software application. Modern databases facilitate everyday activities such as online transaction processing and data analytics.[50] A primary purpose of a database system is to provide users with an abstract view of data and hide details about how data is stored and maintained.[51] Database systems are used to manage collections of data that are relatively large, valuable, and able to be accessed by multiple users or applications.[52]

In building software applications, scholars will most likely use relational databases when storing data within an application and non-relational (or semi-structured) databases when accessing data available on external websites or platforms. A relational data model organizes data into tables and is characterized by spreadsheets.[53] In a relational model, the same type of data must have the same attribute. For example, all rows have the same columns. By contrast, in a semi-structured data model, "individual data items of the same type may have different sets of attributes."[54] Semi-structured data is more flexible than relational data, enables the organization of a wide variety of data types, and facilitates online data transmission. The data object shown in Figure 4 shows an example of semi-structured data in the popular online data format known as JavaScript Object Notation:[55]

> ***Figure 4: Example of Semi-Structured Data***

```
{
    "id": "1",
    "firstName": "John",
    "lastName": "Doe"
},
{
    "id": "2",
    "firstName": "Mary",
    "lastName": "Peterson"
},
{
    "id": "3",
    "firstName": "George",
    "lastName": "Hansen"
}
```

The most common database operations enable users to:

- retrieve information stored in a database.
- insert new information into a database.
- delete information from a database.
- modify information stored in a database.[56]

An important consideration is what level of access different types of users have to a database, or what functionality they are permitted to perform.[57] A scholar may not want to permit all types of users to have the same level of access to data or functionality in order to communicate in a variety of ways with different users.

D. Web Applications and Design

The Internet (or "the web") is an interconnected network of online software applications.[58] The Internet has become so pervasive that most modern software applications are web applications.[59] Scholarly software could take the form of web applications given the omnipresence of the Internet, the availability of visual programming platforms only on the Internet, and the ability of web-based applications to be easily shared, receive feedback, and comport with other scholarly norms.

The three basic components of web technology are a markup language to create linkable hypertext documents and pages in a web browser, a standard notation for identifying the location of resources available over the network (e.g., websites), and a protocol for transporting messages over the network.[60] As a global network, the web operates according to universally adopted and compatible communications protocols, most notably the Transmission Control Protocol and Internet Protocol (TCP/IP) suite.[61] The web adopts the client-server paradigm, in which a web browser or mobile app (the client) sends and requests data from remote servers.[62]

The application layer protocol of the TCP/IP is the name of the first four letters of any website, the HTTP protocol. HTTP operates according to the request-response (and client-server) paradigm.[63] When it comes to data operations over the internet, the HTTP protocol uses the standard operations: GET obtains data, POST creates a new file or other resource, PUT updates an existing database or other resource, and DELETE to delete a resource.[64] These four core HTTP operations parallel that of database operations noted above.

HTTP operations are the core of how web applications and web services communicate, using the standardized architecture known as representational state transfer (REST) application programming interfaces (APIs).[65] Web service APIs have revolutionized how software operates by enabling web applications to easily connect and use each other's data or software functionality within their own systems. Software scholarship is likely to often use APIs and the four operations described above. The purpose would be to integrate services from a variety of web applications and data sources to develop and communicate scholarly ideas.[66]

Scholarly software in the form of web applications must properly implement web design to the extent the application is being used as a communicative tool. Web design principles relate to the display of an application's data and how users interface and input information via keyboard, mouse, and other devices. Web design emphasizes consistency, intuitive navigation, aesthetics, and overall unity and coherence within an application. According to Jason Beaird and James George, "[t]he most important point to keep in mind is that [web] design is about communication . . . design should not be a hindrance; it should act as a conduit between the user and the information."[67] Design techniques that have a significant impact on the effectiveness of web application communication include layout symmetry, proximity and placement of elements,[68] using a consistent color scheme,[69] and using a font appropriate for the message or user base.[70]

As discussed in the next Section, software applications are better for communicating scholarly knowledge than traditional articles and books precisely because they can communicate with web design—by supplementing prose content with textual components and interactive graphical elements.

3. Software Applications and Scholarship

The primary purpose of scholarly software is to further scholarly goals such as truth identification, knowledge production, theory development, and policy advocacy.[71] This Section identifies the three circumstances under which software qualifies as scholarship. It then explains four methods that can be used by scholarly software applications. These are summarized in Figure 5.

Figure 5: Categories and Methods of Scholarly Software

Categories of Scholarly Software	Methods of Scholarly Software
• Software that implements scholarship • Software tools for scholarship • Software that communicates scholarship	• App-ified argumentation • Interactive toolkits • Data visualizations • Policy tech

A. Software is Scholarship

Not all software is scholarship. Software is scholarship only to the extent that it implements research or scholarly ideas in its design or functionality, is used as a tool to develop scholarship, or communicates scholarly ideas.

i. Using Scholarship to Implement Software

Research in computer science, mathematics, and related disciplines is widely used as the basis for developing software that performs statistical and algorithmic computation. Often, this type of software is the basis for developing applications like artificial intelligence.[72] In addition, scholarship in the humanities may be used as a basis for decisions about the design, interactive features, and other properties of software applications. This is because humanities scholarship can guide how an application should be designed to promote goals such as user retention and usability.[73]

In particular, theories and findings in psychology and related fields are often used to make software applications more appealing and intuitive to use. For example, in Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines, Jeff Johnson uses perceptual and cognitive psychology to develop user interface (UI) design guidelines.[74] In Hooked: How to Build Habit-Forming Products, Ni Eyal discusses how to make software applications more engaging based on psychology.[75] Behavioral economics, the application of psychology to human decision making, is another field of academic research that has direct applicability to the design and functionality of software. Behavioral economics includes identifying patterns in human behavior such as decision-making heuristics,[76] and accordingly can inform how cognitive behaviors apply to how users interact

with software applications. For example, applying behavioral economics to the user interface of software applications can help determine what default graphics should be implemented. Behavioral economics can also apply decision making heuristics such as anchoring to inform how to create a point of reference for users as they navigate within an application.[77]

Linguistics is also a field that informs the design of user interface elements of software.[78] And as noted by Inger Lytje, "[w]hen using and designing software we engage in language activity, and the language we use is symbolic, but also visual, spatial, and iconic."[79] Accordingly, theories and empirical findings in linguistics can inform how text and graphical elements are presented to a user to achieve certain communication goals, such as overall clarity, distinction between ideas, and emphasis on certain concepts.

ii. Using Software to Develop Scholarship

As a tool for gathering, analyzing, and displaying large amounts of data, software can play an invaluable role in the production of research and scholarship that would otherwise not be possible. Software provides scholars with new and more powerful ways to engage in standard activities such as developing theories, testing hypotheses, discovering relationships, and engaging in foundational information gathering and research. As explained by the University of Illinois digital humanities department,

> an emerging area of research that refers to the investigation of humanities, arts, and social science research questions through advanced computing technologies. A social scientist who analyses a large dataset of census information using a supercomputer is engaging in [computational humanities, arts, and social sciences], as is a historian who investigates historical texts using visual analytic software.[80]

The Software Sustainability Institute identifies a category of Research Software Engineers "who develop software that is used by [more traditional] researchers."[81] This "scientific software" is bona fide scholarship because it furthers scholarly research agendas by developing software that takes into account the needs, methods, and subject matter of scholarship.[82]

A wide variety of software is tailored specifically for academic researchers in the humanities and social sciences. This software performs tasks, such as analyzing and identifying relationships in text and other forms of network analysis, data-driven analysis and prediction of economic and social phenomena, and more generalized statistical analysis of scholarly subject matter.[83] Computational analysis of text, for example, can enable a scholar to discover new patterns, relationships, and meanings within the corpus of a single writer's works or across those of many.[84] In addition, a wide variety of scholarship-relevant data is increasingly being made available online to be analyzed by

software applications. For example, the Folger Shakespeare API enables scholars to use or create software to search and analyze all of Shakespeare's works.[85] Doing so enables a scholar to develop theories about the meaning and interrelationships of Shakespeare's works, as well as relate them to other literary and scholarly texts.[86] Other scholarship-relevant data that is widely available includes the full text of academic journals that can be used to programmatically analyze the works of other researchers in ways that are not possible through search engines like Google Scholar or Semantic Scholar.

iii. Using Software to Communicate Scholarship

Software and scholarship are also fundamentally related because software can be used to communicate scholarly ideas. Scholarship traditionally takes the form of a written publication, such as a book, article, or essay that is organized into sections. Although originally in print, written scholarship is now available electronically in a variety of formats, including PDF and natively digital documents only available on the web.[87] These web-based works often have hyperlinked text to connect the reader to cited materials.

An important parallel between written scholarship and software is that each requires a type of engineering. Written works face constraints on length, formatting, and reader attention that impose limits and tradeoffs for scholars. Scholarship also typically relies on complex reasoning, creating interdependencies between an author's observations, arguments, conclusions, and findings. Accordingly, like the design of software, the separate components of scholarly work must be consistent with and support each other to create a work free from ambiguity and internal contradiction.

Like written publications, software applications are also mediums of communication and can thus be used to communicate scholarly ideas. As discussed below, software applications can communicate scholarly ideas by using various forms of text, graphics, and interactive elements within the context of a running software application. With these broader set of tools, software applications are a superior method of communicating scholarly ideas relative to static documents. Accordingly, software applications can enhance research agendas in the humanities and social sciences by making traditional, prose scholarship more thorough, persuasive, and analytically precise.

a. Textual Components

Software applications contain text both within the running application and also in documentation that accompanies the software. Typically, the text within the user interface of a running application is used to communicate the software's written output to the user and also about the graphical elements and functionality. For example, text can be used within a software application to show lists of information, as button labels (e.g., "save," "next"), and to organize different functional areas and menus. Likewise,

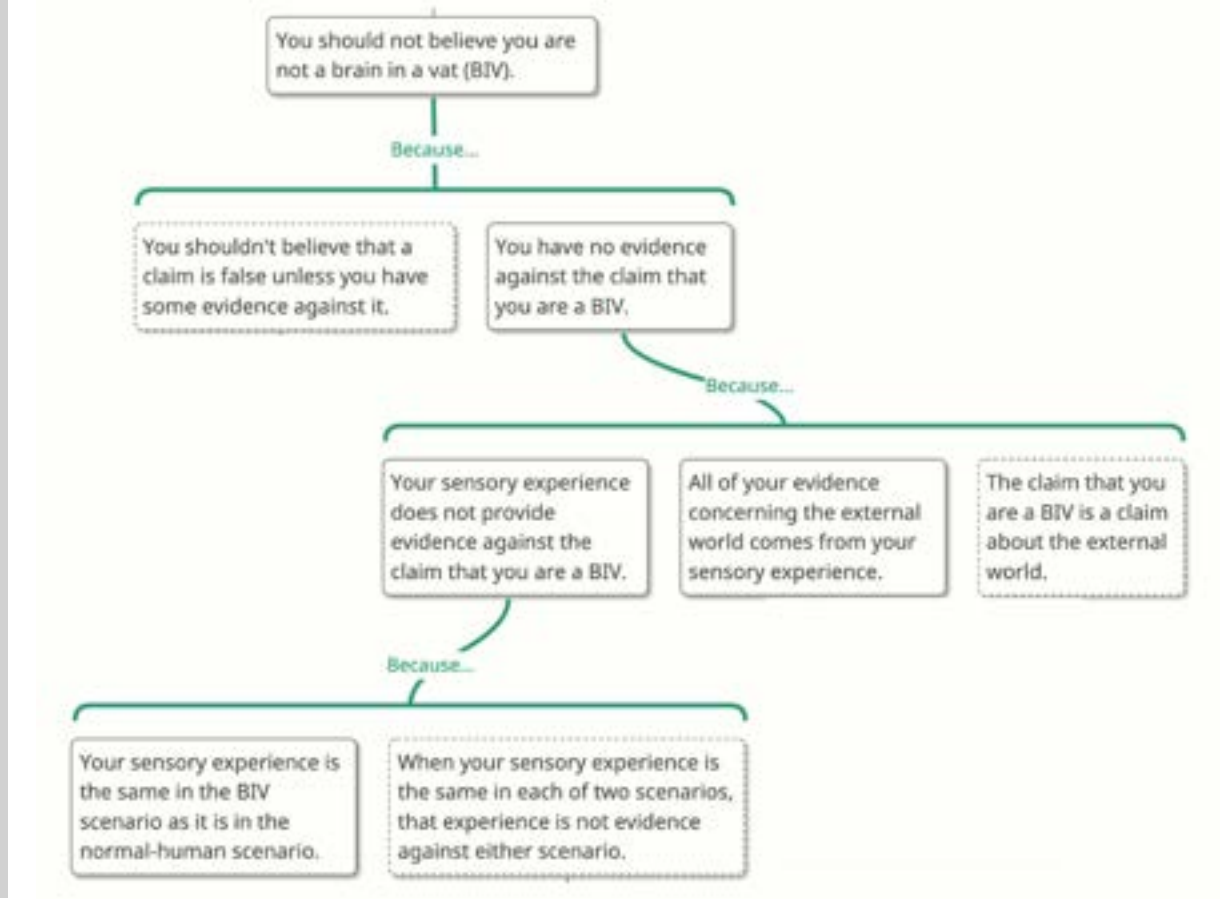
software documentation is typically used to communicate how software is engineered, how it can be customized, and how users should use an application. Software documentation, although accessible from the running application, is typically kept distinct from the application itself. I argue that both of these forms of software text can also be used to convey scholarly ideas, arguments, and other communications.

Importantly, text can be presented in a wide variety of customized formats and designs inside a software application. These include various sizes, colors, and styles, in an organized table, split into columns, and alongside various graphical and interactive elements. The increased flexibility of text presentation and design within a software application enables more effective communication than the standard presentation of prose in an academic article. By incorporating text into the UI of a running software application, a scholar can use the application to organize, format, and highlight prose in ways that are more effective than those available with standard academic publication formatting. Different pages of an application and its menu items can also be used to separate out, organize text, and create internal links. These methods are more effective than a standard table of contents for communicating complex ideas and reasoning.

b. Graphical Elements

Graphic communications studies the ways in which visual elements such as symbols, icons, and images communicate meaning. Key tools in graphic communication include flow charts, argument maps, graphs, charts, infographics, data visualizations, decision trees, timelines, and concept maps.[88] They also include hypertext, which is digital text that is linked to other text, data, or websites. A fundamental proposition of graphic communication is that the use of graphics can make a communication more explanatory, precise, and persuasive than prose communications.[89] For example, Figure 6 is an argument map of the "brain in vat" argument for Cartesian skepticism.[90]

> ***Figure 6: Argument Map Example***

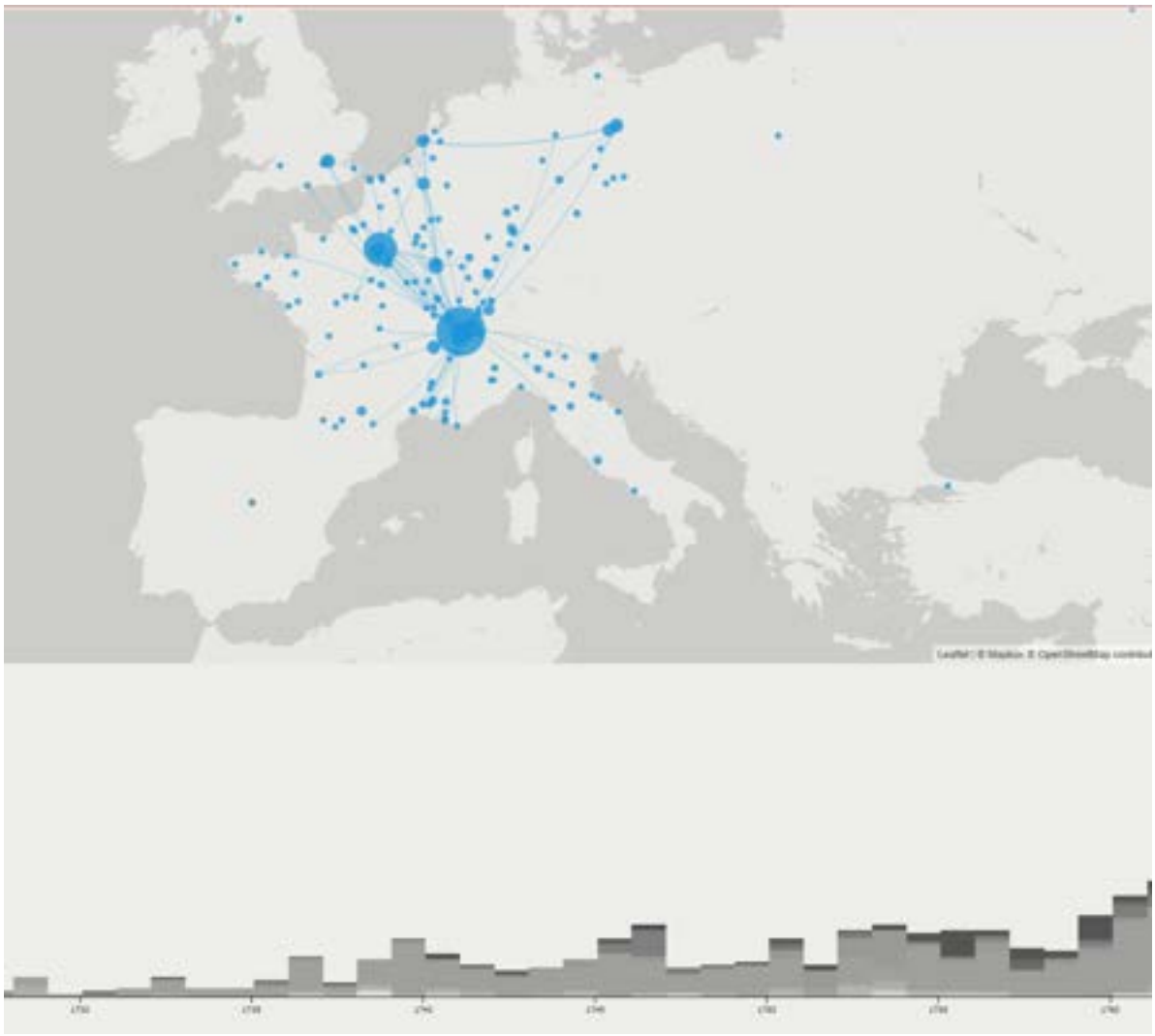


Scholarly software can incorporate an argument map along with prose argumentation and interactive elements such as clicking a component of an argument map. This creates a richer form of communication than merely incorporating an argument map in a traditional, static academic article.

Indeed, software applications are a perfect medium for graphical communication. The user interface of modern, web-based software applications is able to display any type of graphical element in innumerable ways. As noted by Richard Sherwin, Neal Feigenson, and Christina Spiesel, "[d]igital technologies allow the pictures and words from which meanings are composed to be seamlessly modified and recombined in any fashion whatsoever."[91] Any approaches, theories, and innovations in graphic communication can be implemented with software. Unlike using graphics within the text of traditional written scholarship limited to the confines of an essay, book, or similar medium, graphics within the context of a software application have no technical limits.

Fundamental principles of effective software design include properly organizing graphics, economizing on the use of graphics, creating intuitive navigation, and linking related elements.[92] Visual scholarship uses graphical elements to illustrate, analyze, and argue.[93] A photo essay being a classic example of using graphics to advance an argument or theory.[94] Examples of specific projects

include Stanford University's Mapping the Republic of Letters that uses computational analysis of digitized correspondence between 17th and 18th century intellectuals[95] and a New York City Museum of Modern Art project illustrating correspondence between artists in a network graph.[96] Figure 7 shows graphics from each of these projects respectively.

> ***Figure 7: Visual Scholarship Examples***
>
> Voltaire's Correspondence Network
>
> 
>
> Inventing Abstraction 1910-1925



Software applications that contain prose to articulate scholarly arguments can accordingly be enhanced by integrating graphical elements to reinforce their scholarly aims.[97]

c. Programmable Interactivity

In addition to communicating with prose and graphical elements, software applications can also communicate through interactive elements. Software application interactivity arises from the fact that programming software enables it to perform tasks that change the state of the text or graphics that are displayed in the UI and initiate actions such as processing data. Software can enable any of the graphical elements in an application to be interactive. Examples include enabling users to click on icons, scroll through and zoom in on a timeline, manipulate elements in a concept map, and initiate an animation.

Interactivity enables communication that is highly responsive and tailored to its recipients.[98] Software interactivity furthers communication by making the information that is being communicated by an application dependent on the prior actions of the user. These prior actions include what elements they have clicked on or what information they have entered. In explaining the benefit of creating an interactive graphic that displays interrelationships between ideas in philosophy, Deniz Cem Önduygu notes that:

> I understand an idea/argument/concept better when I see it along with others that are similar to it and in contrast with it . . . . what is also valuable here is the ability to see, by clicking on a sentence and displaying its connections . . . . In this respect, [interactivity] can function as a thinking tool that helps people think through these ideas by showing different formulations and

> negations of them. It can also serve as a preliminary academic tool that provides entry points to the literature by listing some of the relevant names and texts.[99]

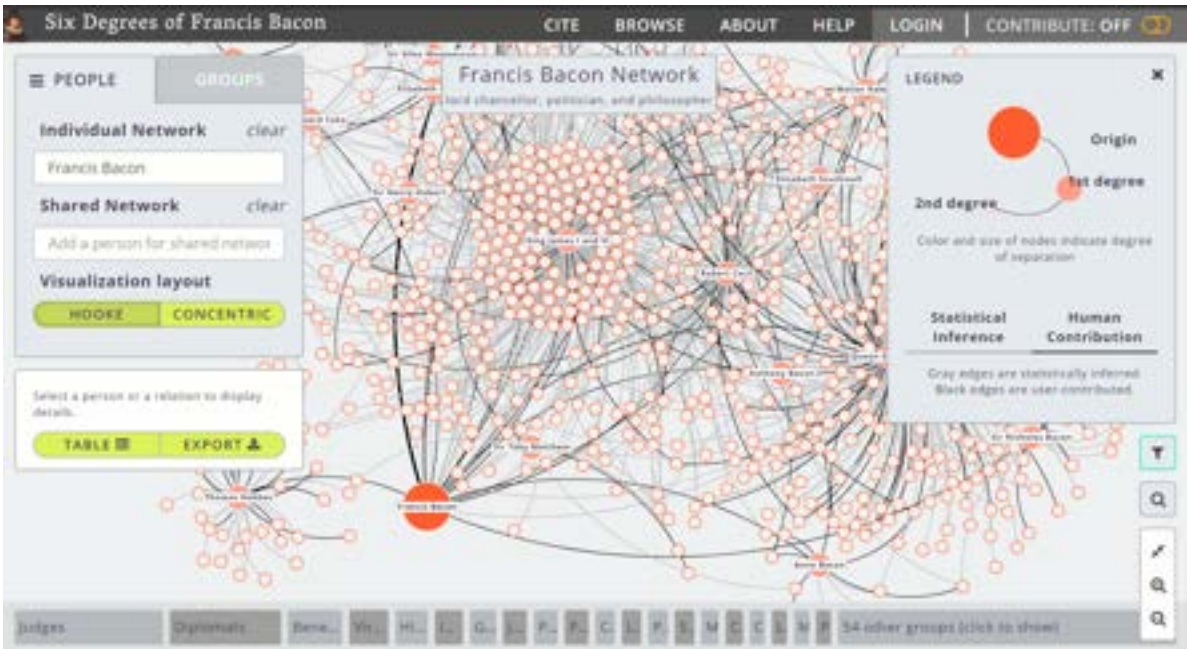
Because interactive software applications can continually respond to user inputs, they are also dynamic, unlike prose or graphics contained in a static article. As noted by the proprietors of the Six Degrees of Sir Francis Bacon interactive network graphic, interactivity is a property that also promotes scholarship:

> Unlike published prose, Six Degrees is extensible, collaborative, and interoperable: extensible in that people and associations can always be added, modified, developed, or, removed; collaborative in that it synthesizes the work of many scholars; interoperable in that new work on the network is put into immediate relation to previously studied relationships.[100]

Software interactivity can communicate scholarly ideas by displaying text that is responsive to a user's question or area of interest. For example, a user may select a potential counterargument from a drop-down menu and in response, the text in the UI can adjust to communicate how the author deals with that argument. Software interactivity can also communicate scholarly ideas by creating interactive graphical elements that display concepts, arguments, and relationships between subjects of scholarly investigation. These interactive graphical elements and methods include graphical representations of documents, groupings, classifications, pattern identification, and spatial elements.[101]

For example, clicking on components of a concept map may reveal definitions or related ideas or scholarship that clarify the concept. Likewise, clicking on components of an argument map, or inputting different text related to an argument map, may change how the argument is being mapped to clarify its reasoning structure.[102] Data that is visualized can also be made interactive so as to communicate its quantitative findings to the user based on user inputs and interaction.[103] Figure 8 shows the main interactive visualization from the Six Degrees of Francis Bacon project that displays and measures Francis Bacon's social interactions, demonstrating how intertwined he was with other scholars and leaders of his time.[104]

> ***Figure 8: Six Degrees of Francis Bacon***



B. Methods of Scholarly Software

I identify four methodological categories of software that can be used to greatly enhance scholarly communication: app-ified argumentation, interactive toolkits, data visualizations, and policy tech applications. Each of these categories of scholarly software can be developed using the types of visual software development tools discussed above.

1. App-ified Argumentation

Scholarship in the humanities often consists of the development of theories via scholarly argumentation - propositions that attempt to assert a truth using deductive or inferential reasoning. These theories or arguments include those about other theories and arguments, the interpretation of scholarly texts and other materials, the normative analysis, and the explanation of some aspect of human society (past or present). Software applications have the ability to clarify and make more persuasive the theory and supporting arguments a work is advancing through the use of textual components, graphics, and interactivity. These elements can show the connections between concepts and arguments in a way that traditional text does not.

In particular, software applications can help scholars formulate and argue for theories in several ways, including:

- Using the application's UI design to highlight, structure, and otherwise organize the text associated with (different aspects of) an argument;
- Using graphical elements to support the interrelation between aspects of an argument, such as by using concept maps and argument maps;

- Using interactive elements to display, combine, toggle, or otherwise manipulate or highlight text that contains arguments or propositions associated with (certain aspects of) an argument;
- Using interactive elements to understand how a theory explains, incorporates, or otherwise deals with a wide range of arguments of phenomenon;
- Automating the application or elaboration of theory to investigate its truth, moral value, or consequences; and
- Any of the foregoing with respect to counterarguments and alternative or competing theories.

Given the pervasive disagreements over the meaning of text, theories, and arguments in the humanities, the ability of software to clarify the precise meaning and formulation of arguments and theories is of significant value.

2. Interactive Toolkits

Qualitative research in the humanities and social sciences includes case studies, ethnography studies, phenomenological studies, and content analysis.[105] Qualitative research uses inductive reasoning and focuses on the subjective elements of human experience. It also relies heavily upon primary research materials such as interviews, public records, archival documents, audio and visual media, and participant studies.[106]

Software applications provide qualitative research with a much more comprehensive and engaging medium to express findings, research methodologies, analysis, and make available the primary materials upon which the research is based. This can be accomplished by creating a software application in the form of an interactive toolkit that has features and functionalities such as:

- Text that consists of standard research prose containing the core substantive content of the research;
- Several pages of menu-based navigation within the application that take the user to different aspects of the prose and underlying research;
- Usage of graphical elements including charts, animations, and video;
- Interactive elements that ask for user inputs (e.g., search boxes, drop down menus) and in return show or navigate the user to some aspect of the research, including responsive displays of the information or data underlying the research;
- Graphical elements that visualize the reliance of theory or conclusions upon data;
- Guided question and answer forms, responses, and contextualizing the input against the findings and framework of the research; and
- Connecting to or incorporating real-time data from external sources to stay continually updated.

Interactive elements for qualitative research include the user selecting which component or perspective on a case study to view the study from. This is because a different user perspective may change the study's findings or the subjective aspects of a phenomenological study. Examples of interactive toolkits include:

- PsyToolkit, which can be used "for demonstrating, programming, and running cognitive-psychological experiments and surveys, including personality tests."[107]
- The History of Philosophy Visualized and Summarized: "the history of Western philosophy showing the positive/negative connections between some of the key ideas/arguments/statements of the philosophers."[108] See Figure 9.

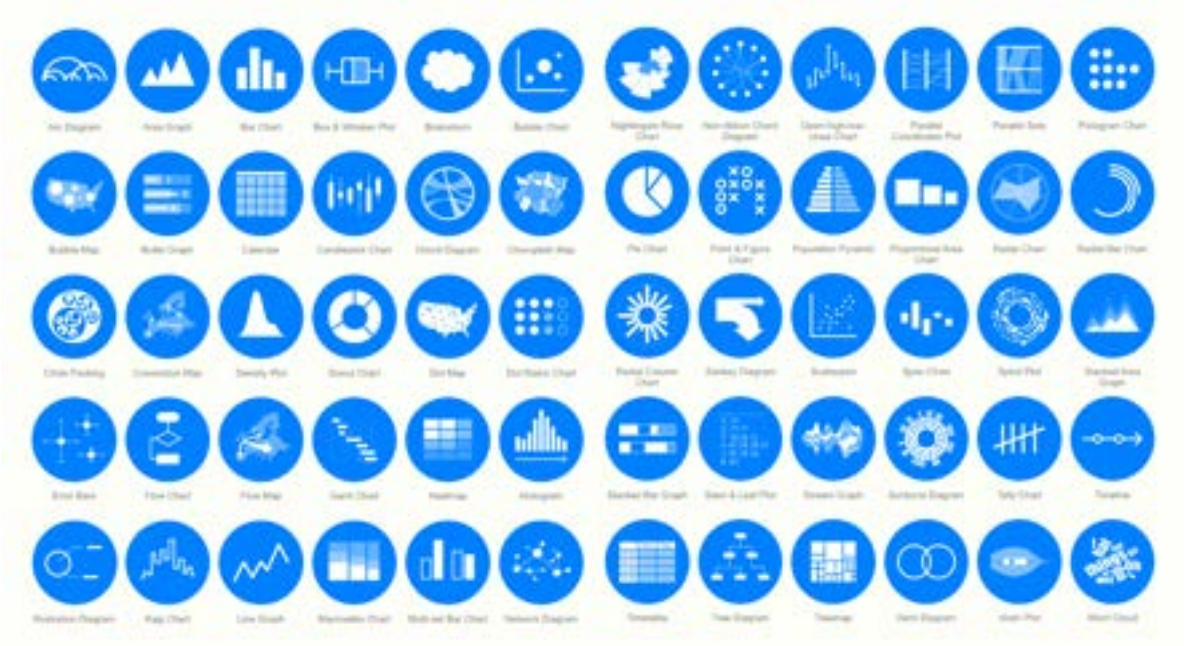
> ***Figure 9: History of Philosophy Data Visualization***
>
> 
>

Scholarly software in the form of interactive toolkits can provide additional and clearer insights from the observations, analysis, and other types of knowledge produced by a qualitative study compared to those in a traditional article or book.

3. Data Visualization

Quantitative research has long been applied to answer questions in the social sciences and humanities such as about the properties, relationship between, and future states of phenomenon. The foundational question in empirical research is whether there is a significant correlation between two variables (based on the statistical analysis of large data sets) that permit a researcher to draw an inference or conclusion about an underlying relationship.[109]

Data visualization tools can increase the explanatory power of quantitative research findings by providing a reader with coherent and meaningful presentations of large volumes of data and complex findings to show trends, relationships, and highlight significance. As shown in Figure 10, these data visualization tools include a wide variety of visual formats for data, ranging from standard pie charts and scatter plots diagrams to bubble charts and network diagrams.[110]

> ***Figure 10: Types of Data Visualization***
>
> 

Scholarly software adds to the explanatory power of quantitative research by enabling data visualizations to be directly embedded into the substantive text. Scholarly software also enables the visualizations to use color and advanced design methods, be interactive and animated, and be updated in real time by connecting software to live data sources.

Data visualizations within the context of software applications can also enhance the scholarly goals of quantitative research in the following ways:

- underlying datasets can be made readily accessible and searchable from the application
- users can interact with the theory and data of the study to better understand and validate them
- findings and data can be visualized to more accurately and meaningfully communicate research methodology, theory development, statistical findings, and analyze counterarguments and false or null hypotheses[111]
- updated and real-time data can be incorporated into the study to continually test hypotheses.

4. Policy Tech

Another type of scholarship has the goal of impacting the decisions of governmental policy makers. This type of scholarship often takes the form of identifying a social or economic problem, proposing a solution, discussing alternative solutions and why the proposal is better suited given a certain normative approach, and recommending general or specific courses of action.[112] Policy-oriented research may employ various methods including philosophical argumentation and social science findings such as those in sociology or economics.

An example of a policy area in which software applications can be used to further its specific goals is civic tech, which is "technology used to directly improve or influence governance, politics, or socio-political issues."[113] A primary purpose of civic tech is advocacy: "technology used for social and political advocacy purposes by nonprofits, political campaigns and ordinary citizens."[114] Examples of software being used in civic tech advocacy applications are the Countable platform that uses interactive graphics to explain policy developments and the Crowdpac software application that crowdfunds for political campaigns. Another policy area that scholarly software applications can assist with is human rights research and advocacy. Software applications can assist users in navigating legal and administrative requirements and collecting information to provide global awareness about rights violations.[115] Policy-oriented scholarship and research that takes the form of a software application can accordingly further normative goals by using the underlying research and argumentation to engage policymakers and other audiences to cause change.

Software applications are a potentially superior medium to promote and enact the normative goals of policy research. Applications can use the graphical elements and interactive aspects of software to better convey the quantitative and qualitative reasoning behind a proposed policy change, including the more emotional and fairness-oriented aspects that policymakers may find particularly persuasive. Immersive media in particular may forcefully convey well-researched and documented harms to particular groups that a policy change may be able to alleviate.

4. Developing Scholarly Software

This Section discusses important recent developments in technology that dramatically increase the ability of humanities scholars and social science researchers to directly engage in software development. It then shows how scholarly software can be developed using an exemplary visual development platform.

A. Technology Enablers

Recent innovations in the world of technology have made developing software for scholarly purposes much more accessible and meaningful for scholarly purposes.

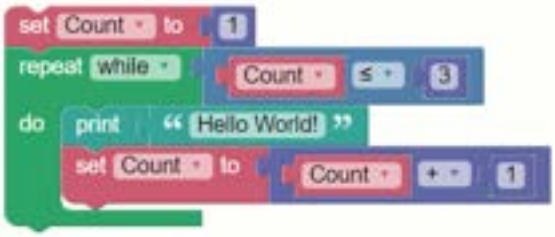
1. Visual Development of Software

Programming software has historically required programmers to use code to develop powerful and customized software beyond special purpose tools such as Excel. However, in the mid-2010s visual software development platforms finally passed a tipping point of sophistication so as to enable scholars and others not trained in coding to create extremely powerful and customizable software for scholarly purposes.

A very simple example of visual software development compared to traditional code-based development is shown in Figure 11 using the Google Blockly platform. The visual programming component for displaying "Hello World!" until the "OK" button is pressed three times is displayed next to its equivalent in the Javascript programming.[116]

> ***Figure 11: Visual Programing Versus Code Programming***

Visual Programming	**JavaScript Code**

	var Count;
	
	Count = 1;
	
	while (Count <= 3) {
	
	window.alert('Hello World!');
	
	Count = Count + 1;
	
	}

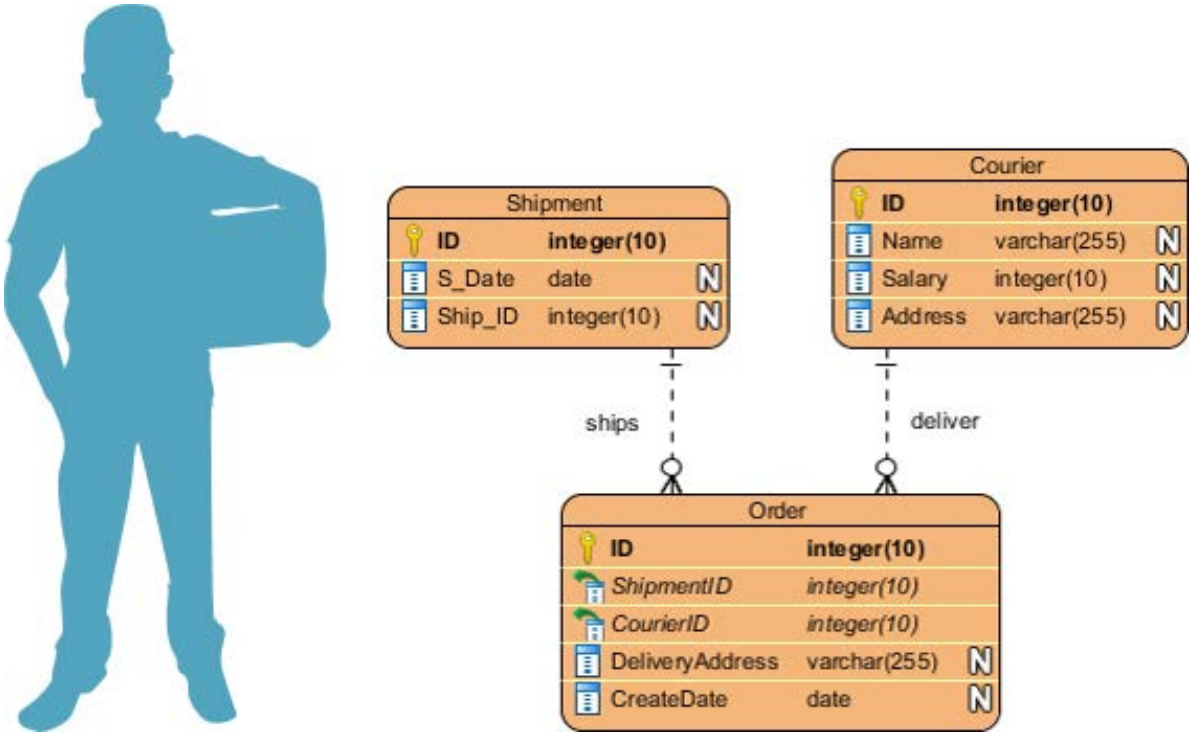
By abstracting away from underlying software code syntax in the software development process, visual software development enables scholars and researchers to focus on how software applications can further scholarly goals. With visual development, scholars do not need to be concerned with the syntax of any particular programming languages. Learning code syntax not only dramatically slows development, but also is far removed from the substantive content of scholarship. The high level of abstraction of visual development creates a tight coupling between programming components and the meaning that the components are intending to communicate. As indicated by the visual programming component in Figure 11, visual software development components nearly have the communicative power of writing.

Visual software components are both symbolic and functional, and therefore promote linguistic meaning with action-functionality. As noted by the no code platform Unqork, "[b]y fully abstracting code into a completely visual interface, users can rely on [graphical] components to build applications .

. . . No-code allows us to work in natural human languages."[117] With visual development, "ideas are immediately turned into [software applications] that work the way they should intuitively, allowing users to focus on ideas over the syntax."[118] For example, Blawx uses Blockly's visual development approach to program legal rules in a way that much more closely resembles traditional legal text than computer code.[119]

In the context of databases, visual software development enables the use of Entity Relationship Diagrams and other visual database tools that permit data processing "in ways that are intuitive to the human brain . . . ."[120] An ERD defines entities, their attributes, and shows the relationships between them to illustrate the logical structure of databases.[121] An example ERD for a database that organizes data about shipments is shown in Figure 12.[122]

Figure 12: Entity Relationship Diagram Example



Visual database tools can communicate a scholar's view of the important properties, relationships, and hierarchy among concepts, real-world entities, and other aspects of their work. Indeed, a visually displayed database is effectively a concept map that can be used for scholarly analysis and argumentation.

2. Open Data and Application Programming Interfaces

The power of no code technology is magnified because it has arisen in the midst of the two other important developments taking in technology: the open data movement and the application programming interface (API) revolution. Open data and API-enabled workflows significantly enhance the ability of scholars to express ideas by enabling them to connect and integrate an extremely wide array of external platforms, software-driven functionality and services, and real-time data sources.

The open data movement refers to the increasingly widespread availability of continually updated data about numerous aspects of human society, especially in the governmental and academic context. The accessibility of public data was greatly enhanced in January 2020 when Google's Public Dataset search became publicly available, making millions of data sets searchable.[123] In parallel to the increasing availability of public data is the increasing accessibility of tools that enable users to employ the methods of data science, data visualization, and machine learning without code or formal training in statistics.[124] These tools can be used alongside or directly integrated with the user interface of visual application development platforms.[125] Machine learning and data science tools enable analysis of large data sets to find patterns and estimate future outcomes based on data that reveals prior associations. For example, according to the no code data science tool ObviouslyAI,

> a large insurer developed a report for predicting the likelihood that a workers' compensation claim would lead to litigation. Claims with probability of litigation were referred to senior staff for early and attractive settlement offers. This saved the company 8% of the litigations they would face otherwise and $3 million annually.[126]

The API revolution consists of the increasingly widespread practice of software and data being made available to be easily integrated with other applications without having to rebuild any of the functionality of the external software. A prominent example is Google Maps, which is available via API so that websites and other apps can use Google Maps in a way that is relevant to their business without having to build their own global, GPS-based map application. The Programmable Web directory lists of over 22,000 public APIs.[127]

Supporting the connection of software applications via API are visual connector platforms such as Zapier and Integromat. Connector apps offer pre-made interconnections (often called "recipes") between different software systems. These premade connectors remove the need to read an application's API documentation to integrate its functionality. No code platforms also offer plug-ins that often make adding in the functionality of other applications even easier than using a connector platform. Connector apps, plug ins, and other API-enabled workflows significantly enhance the ability of scholarly software applications to communicate ideas. APIs can connect scholarly software to an extremely wide array of platforms and data sources that are relevant to one's research.

Taken together, the massively increased accessibility of software development, data sources, and ability to connect to other applications has pushed technology past a tipping point so that a single scholar or researcher working alone can build functional software faster and cheaper than an entire team of experienced coders prior to these developments.

B. Developing Scholarly Software with Visual Programming

As discussed above, the textual, graphical, and interactive elements of software applications enable applications to communicate scholarly ideas in ways that are superior to standard, static prose.[128] Visual development platforms enable scholars to incorporate all of these elements to communicate. Below, I focus on the approach of the visual software development platform Bubble as an example of how to develop scholarly software.[129]

1. User Interface: Text and Graphical Elements

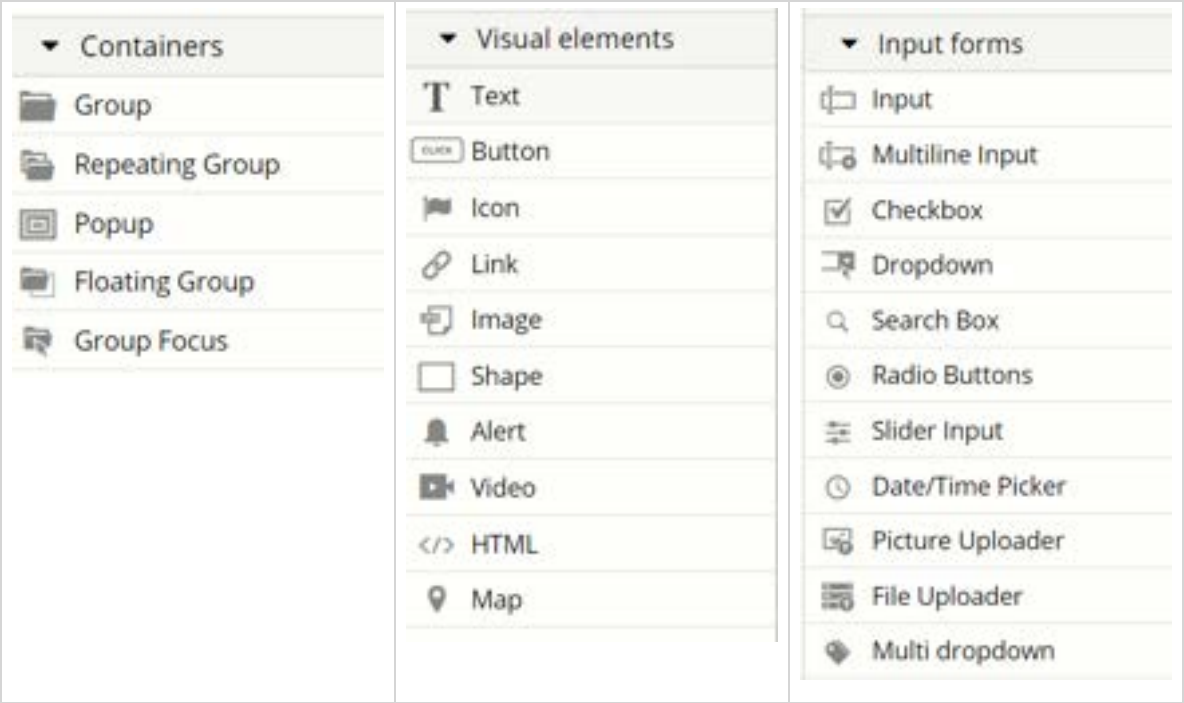
A software application's user interface is the visual means by which users view and interact with an application's data and programmable logic. UI elements include all of the textual and graphical elements that are able to communicate scholarship:

- Text fields,
- Buttons that initiate actions, such as navigating the user to another page or saving input data in a database,
- Input fields,
- Icon, images, video, and other graphical and design elements,
- Elements that group or otherwise organize other elements,
- Navigation elements such as sliders,
- Standardized elements for uploading files and inputting date/time, and
- Graphs, charts, timelines, and other data-oriented elements.

UI elements have properties. These properties include characteristics such as size, page position, color, and how they may behave when certain conditions are true or in response to user actions. User interaction with UI elements can also initiate programming tasks. These actions are governed by the logic programmed into the application, and may or may not interact with the application's database. UI elements may also have data attached to them (known as a "state") that does not reside in any database.[130]

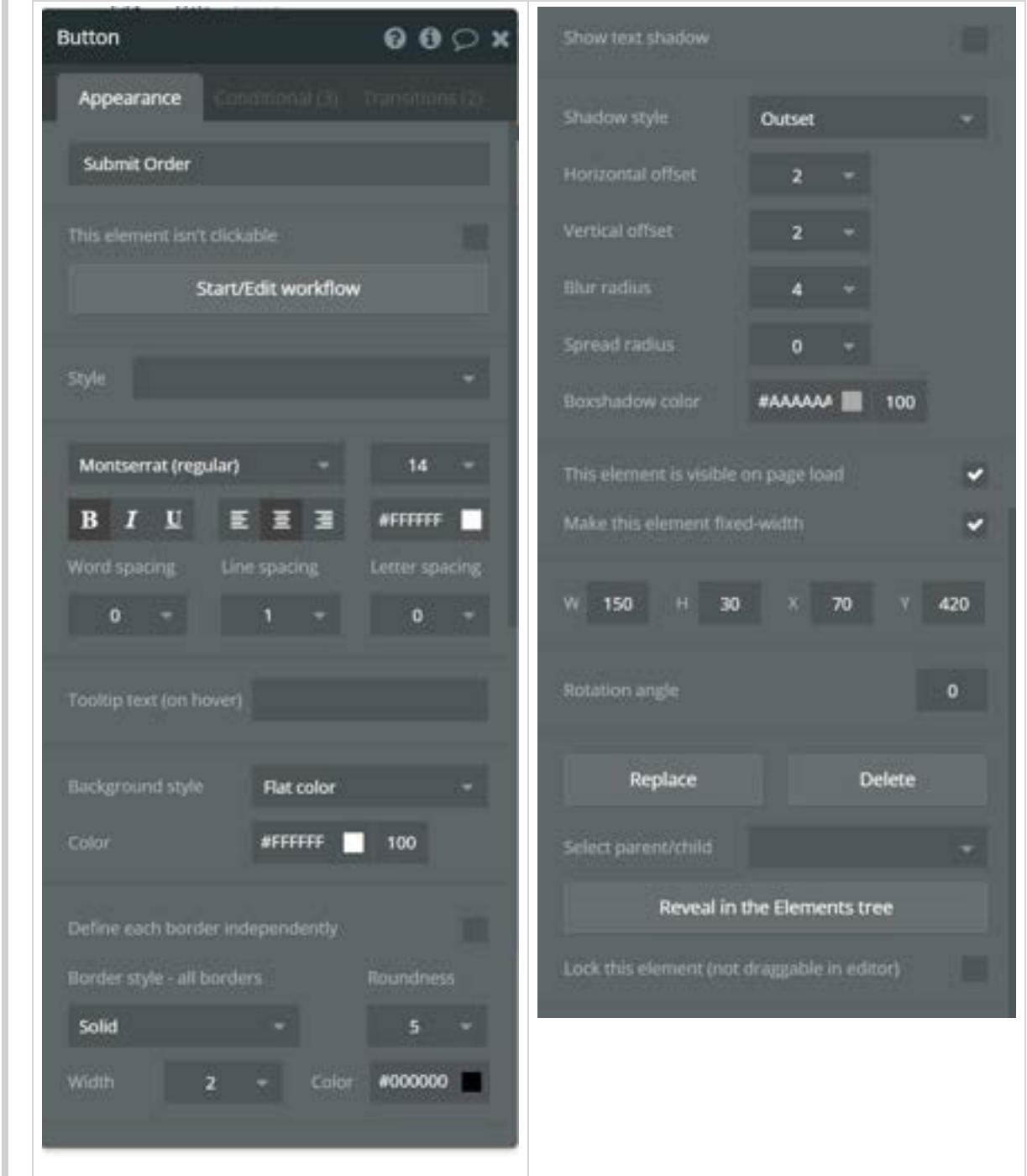
As shown in Figure 13, Bubble has a wide variety of built-in UI elements that can be added to a software application using the main application editor.

> ***Figure 13: Bubble User Interface Editor***



These elements can be added anywhere on a page by selecting, dragging, and dropping them. Once a UI element is added, its properties can then be viewed and modified by viewing and editing its corresponding Property editor. An example of this is shown in Figure 14.

Figure 14: Bubble UI Element Property Editor



Bubble also enables scholars to display quantitative data from the application's database or external sources in graphical form. This can be accomplished using a built-in graph UI element, a third-party plugin, or by embedding charts and other data visualization tools within an application using widely-used HTML and iFrame formats.[131] Bubble's built-in chart element, for example, enables the display of line, bar, radar, pie, and doughnut charts.[132] Social science researchers can also use Bubble's API plugin to interact with external data science and data visualization platforms.[133] This allows

researchers to use Bubble to perform statistical analyses and visually present data to further scholarship as discussed below.[134]

2. Programming Logic: Interactivity Through Workflows

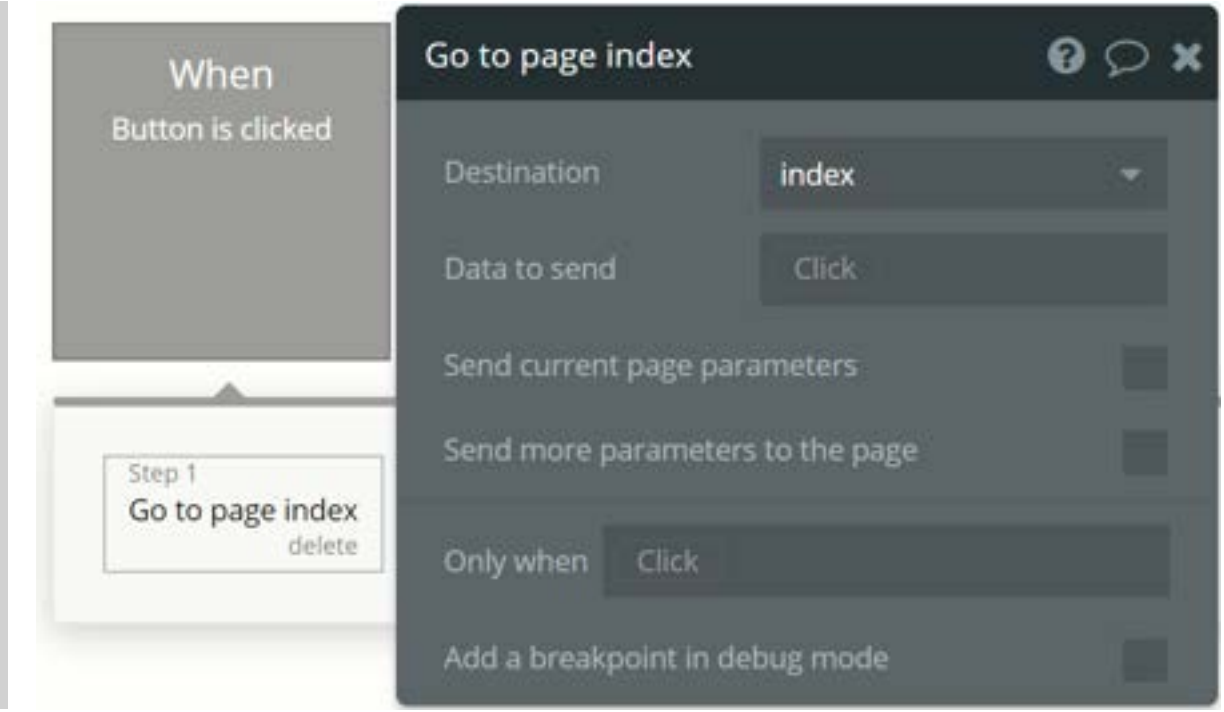
The heart of any software application is its programming logic that enables the application to undertake tasks. Programmability creates interactivity between a user and a software application because the application is instructed to undertake a task in response to user inputs or changes in external data. This type of software-generated interactivity can be used to communicate scholarship.[135]

Programmable workflows implement the logical operators and other fundamental computer science concepts discussed above.[136] Triggering events include user interface triggers (e.g., pressing a button, entering text in a text box element, a change in another element) and changes to data in an internal or external database (e.g., two days have elapsed since an email was sent, a new document was uploaded to a government website). Actions that take place in response to triggers include changes being made to a UI element (e.g., new text based is displayed, color changes to a graphic, the appearance of an image), changes being made to internal or external data, and the performance of an operation such as performing an operation on data. A second type of workflow structure is maintaining a property or performing an action while or so long as a condition remains true. For example, users may not be permitted to access certain data so long as they have not passed a security test.

Rules that govern workflows can also be made subject to exceptions so that the workflow only operates (or does not operate) when certain triggers take place or conditions are true. Workflows also enable significant automation when multiple actions take place in response to relatively few triggers. Ultimately, the connection between trigger and resulting action is governed by an application's underlying programming logic.

Bubble enables visual programming of workflows using its Workflow builder. Figure 15 shows a very simple workflow example that programs software to navigate the application's user to the Index page of the application if a certain Button is clicked.

> ***Figure 15: Simple Workflow in Bubble***



Workflows can also implement programming logic on external software systems in order to integrate the data or functionality of those systems within an application built with Bubble. The standard approach for web applications to interact is using the RESTful API protocol.[137] The Bubble API plugin enables applications to connect to other applications. In a common REST-based action, an application in Bubble will send data to an external application service that will return data based upon the data that it receives from Bubble. For example, the Google Programmable Search Engine API enables an application built with Bubble to run a customized search on any specific website from within the Bubble application.[138] This is accomplished by sending search terms to the Google Search Engine API corresponding to the customized search.[139] In response, the Google API will send search results back to be displayed within the application created with Bubble.

3. Data Usage

Data is a foundational aspect of functioning software. By interacting with UI elements, users can view, add, edit, delete, and have calculations performed on an application's data. In addition, workflows can process, change, or otherwise involve data within an application's database. And by using an API to connect to data that resides outside of an application, users can likewise create workflows that interact with data that is external to the application's database. Connecting to external APIs also enables functionality from third party applications to be integrated into an application's workflows, such as Google Maps, payments, and search functionality.

Creating a database is fundamentally an act of conceptual abstraction and categorization. To create a database, one must first create a conceptual schema of how real-world objects or information should be categorized, relate to one another, and be described with data-oriented properties. Data categorization involves determining the appropriate level of abstraction to create a database. Data relation involves determining how to connect two or more databases together due to sharing common data points, if at all. Data description involves how to identify the properties of data objects with particular data points. This often entails deciding what column headings should be used in a standard database.

In creating an application about Shakespeare's works, for example, the design of the database will revolve around decisions about how to categorize his works (e.g., by type or by theme), how his works may relate to each other so that common properties can be identified (e.g., an archetypal character that appears in many of his works), and how to describe any particular work with properties such as length, character motivations, and date published.

Each decision made in designing a database can incorporate scholarly determinations about the importance and interconnection between ideas, as well as be used to communicate and justify those decisions to scholarly software users as part of the scholar's theory or other scholarly goals. Databases that are reflected in UI elements and presented graphically can also be used to communicate ideas in addition to organizing and storing data.[140]

Bubble enables data-oriented functionality for software application creators with a built-in database. In terms of database conceptual hierarchy, databases, columns in a database, and rows in a database are arranged as types, fields, and things respectively.[141] Developers can use workflows to connect UI elements to database-oriented workflows, and can also schedule database actions to take place when certain conditions are met. Bubble enables developers to program database actions using a visual expression builder that searches, displays, and edits database entries in numerous ways.[142] Bubble enables different databases to refer to each other by enabling the field in one database to be single values or list of values from the field of another database.[143] Complex expression for extracting and replacing text in a database can also be used in Bubble with the formalized regular expression (or "RegEx") approach.[144]

5. LegalTech and Legal Scholarship

This Section focuses on legal scholarship as a type of scholarship that can be enhanced by the use of software applications. Legal scholarship is an interdisciplinary field that relates to and draws upon a wide variety of other fields and methods in the humanities and social sciences, ranging from legal history and legal philosophy to quantitative and empirical approaches. Legal reasoning is an inherently

computational form of reasoning which makes legal scholarship particularly well-suited to benefit from the logic-driven functionality and corresponding UI aspects of software applications.[145]

A. Law and Computation

Legal reasoning consists of the application of legal rules to facts to arrive at legal conclusions. Legal rules stem from the written opinions of judges deciding cases (jurisprudence), statutes and regulations, and legally binding contract language. The application of legal rules typically uses formalistic (deductive) reasoning such as syllogisms. Legal reasoning also employs analogical arguments and policy-based reasoning. Legal rules may be structured with requirements that are conjunctive, disjunctive, multi-factor, or consist of balancing tests, often accounting for exceptions.[146]

Because the core of legal reasoning is formalistic and deductive, legal reasoning is an inherently computational discipline.[147] Deductive legal reasoning takes legal facts as inputs, applies rules, and outputs a legal conclusion (i.e. an activity being deemed as illegal or a breach of contract). In computer science terminology, legal rules are expressions that evaluate inputs (facts) to generate outputs (legal conclusions).[148] It follows that legal reasoning is similar to computation. Its core activities involve making determinations about the scope and applicability of definitions, conceptual abstraction and hierarchy, and the proper application of potentially conflicting rules.

The inherently computational nature of law has been recognized since the 1940s.[149] More recent observations on the computational nature of law include that of Mark Flood and Oliver Goodenough, who state that "legal logic . . . aspires to the requisite characteristics of rigorous definition and internal consistency that should make it accessible to formal computational tools."[150] With respect to contract logic in particular, transactional attorney Carolyn E. C. Paris observed that "[a] contract is not like a newspaper article, a memo, or a pleading. A contract is more like a computer program, a set of rules and procedures, or a machine."[151] Likewise, Professor Harry Surden argues that "parties can make certain contract terms 'understandable' to a computer by translating the meaning of the term into a set of consonant computer instructions."[152]

Given the significant overlap between the formal structure and methods of legal reasoning and computation, the field of algorithmic law and practice emerged, using software to engage in legal reasoning and decision making. A foundational aspect of this work from a scholarly point of view is converting laws and legal documents into a format that can be computed (i.e., machine readable). For example, the CEN MetaLex project has developed "a standardized view on legal documents for the purposes of information exchange and interoperability in the context of software development."[153] Likewise, the Legal XML organization has developed standards for legal documents and related applications including electronic court filing, court documents, legal citations, transcripts, criminal

justice intelligence systems, and others.[154] The Legal Specification Protocol working group seeks to unify existing efforts with the goal of

> *. . . develop[ing] an expressive Legal Specification Protocol (LSP) [with] the capacity (i) to capture the event space salient to legal formulations, (ii) to represent the computational and logical structure of legal specification and (iii) to allow the execution of the process and workflow imbedded in that structure to provide useful applications to a broad swath of legal tasks, including contracting, compliance, and legal judgments.*[155]

Underlying computational law is legal informatics, the study of different types of legal information and activities involving systematic approaches to legal data.[156] Functionally, legal informatics provides the data-orientation that is necessary for software to apply computational methods to law. Legal informatics converts legal information into data compatible for use with a software application's database. Conceptually, legal informatics provides the basis for computational legal scholarship.

B. LegalTech and Legal Scholarship

1. Types of LegalTech

Legal technology ("LegalTech") is software that contributes to the practice of law and furthering client objectives. The goals of LegalTech include enhancing the delivery of legal services and furthering public interest goals and access to the legal system (justice).[157] Legaltech has many subcategories that mirror those of legal practice areas and the application of particular technologies. Given the commercial potential of algorithmic law, LegalTech often seeks to automate legal reasoning to make deterministic conclusions about the application of law based upon user information and other facts.

Key categories of LegalTech include:

- legal research
- electronic discovery
- automated pleading, motion, and discovery document-writing and analysis
- lawyer-client matching platforms
- law firm matter management
- litigation analytics
- contract management and drafting automation
- digitally-assisted and automated contract negotiations
- digitizing contracts and other documents into digital-native, structured-data documents
- contract analysis and analytics
- corporate legal operations analytics and workflow automation
- legal intake and analysis via expert systems

- online dispute resolution.

Legaltech applications utilize a wide variety of technologies that are common to modern applications generally, such as artificial intelligence[158] and automation.[159] For example, LegalTech developers create software to analyze case briefs to make suggestions for improvement, compare hundreds of thousands of contracts to produce quantitative risk assessments, and predict the rulings of judges. Digitized documents, data about documents, and legal services data can also be incorporated into an enormous range of semi- and fully-automated digital workflows that include client-related services and decision making. In terms of user interface, LegalTech apps may use traditional point and click and touch screens, voice inputs, and interactive chat bots. No code, visual development platforms tailored specifically for legal applications are becoming increasingly available and include Bryter, Documate, and Neota Logic's Canvas.

2. Types of Legal Scholarship

In a 2013 article in the Journal of Legal Education, Harvard Law School professor Martha Minnow identified several distinct types of legal scholarship.[160] They are summarized as follows:

> 1. restatement of legal doctrine contained in court opinions
> 2. reconceptualization of legal doctrine
> 3. policy analysis and recommendations
> 4. qualitative and quantitative testing or assessment of legally relevant proposition
> 5. assessment and critical perspectives of legal institutions, systems, or institutional actors, typically using an interdisciplinary approach
> 6. comparative legal history
> 7. philosophy of law.[161]

Legal scholarship is in part unique because its subject matter is the law and its intellectual methodology must at least take notice of legal reasoning. Nonetheless, legal scholarship has significant overlap with other disciplines in the humanities both in focus and methodology. Accordingly, legal scholarship possesses the same relationship with software that scholarship does more generally and can, therefore, be similarly enhanced through the application of software applications.

3. LegalTech as Legal Scholarship

Like software applications more generally, LegalTech applications qualify as scholarship to the extent they implement scholarship in their design or functionality, are used as a tool to develop legal scholarship, or used as a medium to communicate legal scholarship. As a medium to communicate, LegalTech applications can employ the methodological categories of app-ified argumentation, toolkits, interactive data, and policy tech.

App-ified argumentation is the primary method for software applications to add value to legal scholarship. It restates, reconceptualizes, or otherwise engages in theoretical argumentation with respect to legal doctrine, including legal philosophy. The computational nature of legal reasoning means that discussions and arguments about legal doctrine are conducive to being programmed into the software's functionality and presented with graphics, diagrams, and other visual media. For example, the computational nature of legal doctrine can be captured and expressed in the interactive functionality of software, whereby clicking a button or other UI trigger causes an on-screen action that expresses some aspect of the doctrine or argument. A scholarly LegalTech software application can enable the reader to click on elements within the application to highlight or reveal aspects of propositions about legal doctrine and their supporting concepts. Inputting data may also cause legal outcomes or conclusions driven by a scholar's underlying theory to be displayed on a screen that respond to the particular input.

Interactive toolkits can also be applied to enhance the goals of most types of legal scholarship. Most types of legal scholarship rely on a wide range of primary and secondary materials that support the qualitative arguments and findings of scholars. A scholarly software toolkit can make these materials readily accessible to the reader and otherwise avail itself of the features of toolkits described above.[162] A toolkit may also help to automate the production of legal knowledge and conclusions. This can be accomplished by obtaining information from a user and, based on the legally relevant aspects, output a legal recommendation, such as the legality of certain activity or what next steps should be taken. This functionality is scholarly to the extent the relationship between the input and output is governed by a scholarly understanding of the law. Indeed, scholarly LegalTech software applications may distinguish themselves by demonstrating how their particular doctrinal understanding of the law is more efficacious in making sense of, achieving, or predicting legal outcomes.

Data-oriented legal scholarship can also be significantly enhanced by using the wider range of graphical elements and functionality that software makes available. Data-oriented legal scholarship uses quantitative methods to predict the outcomes of judicial opinions, the enforceability of contract terms, the relationship between legal rules, and other research activities.[163] Quantitative methods are often dependent upon underlying scholarly theories about legal phenomena. For example, a software application that processes large volumes of contract language data to quantify legal risk may be relying on a scholarly understanding about contract law doctrine or regulatory requirements. When quantitative relationships are visualized in an academic law journal article, data visualizations are displayed as static images.[164] However, LegalTech software applications can make these visualizations dynamic and interactive and thereby improve their expressive power.[165] For example, a software application enables users to interact with and manipulate data visualizations, as well as view how the visualization changes in response to different data. In addition, a software application, unlike a traditional article, is not limited by the number and types of visualization that it can present to the

user. Legal scholars, therefore, do not need to be artificially constrained by law journal page limits and formatting in deciding which visualizations to incorporate into their work. In principle, any number and type of visualization can be included when software applications are used as a scholarly medium.

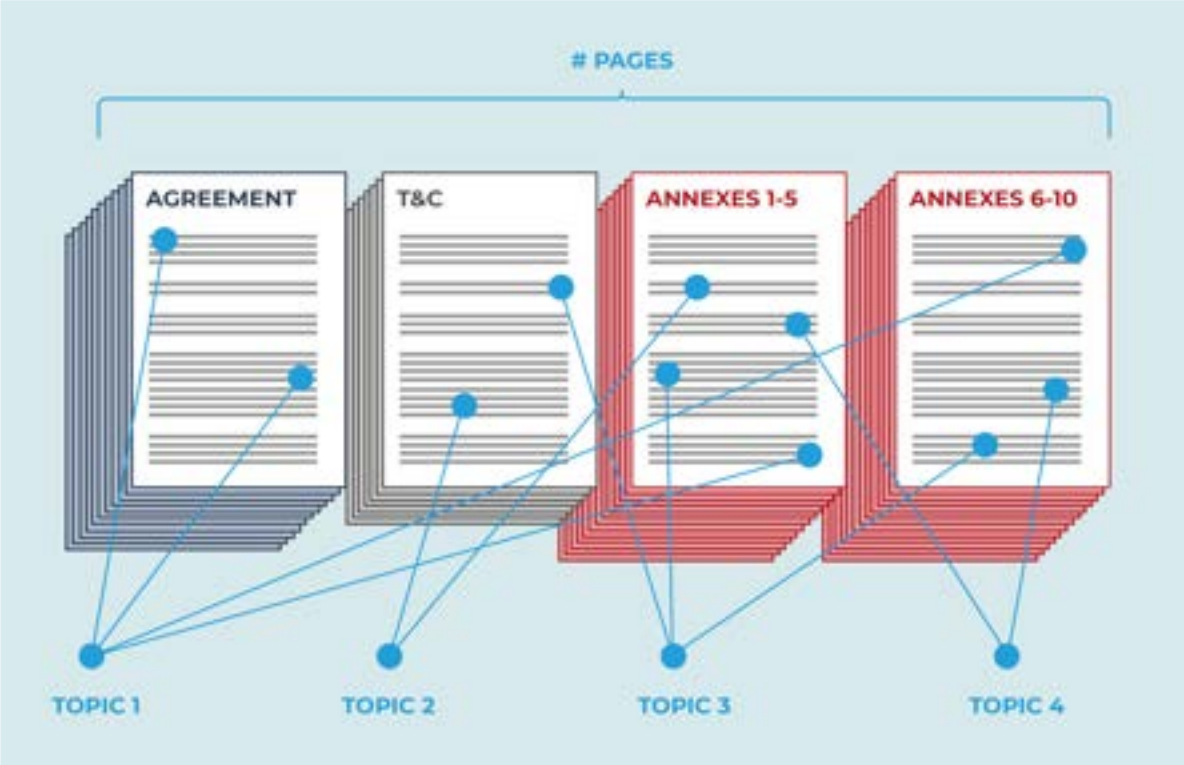
Scholarly LegalTech software that seeks policy change as its primary goal falls under the policy tech category of scholarly software. Policy change be accomplished by the software application creating a platform for affected parties to become educated about a topic, communicate with each other, and mobilize to engage in legal reform. Scholarly LegalTech applications may also help to gather empirical findings relevant to reform and create interactive visualizations that display the findings. For example, interviews of parties that are harmed by a law may be readily accessible using the menu interface of a software application toolkit. Economic and other quantitative measures of the impact of legal change can likewise be presented.

The communicative value of scholarly LegalTech software can be enhanced by applying design thinking principles to its user interface. Legal design is a rapidly growing, interdisciplinary subfield of LegalTech that seeks to make legal knowledge more usable and engaging.[166] Legal design principles can be followed when designing the UI and graphical elements of scholarly LegalTech, and may include principles such as the following developed by Gerlinde Berger-Walliser, Thomas D. Barton, and Helena Haapio:

> - identify user needs through observation and empathy
> - define project goals through communication, visualization, and prototyping
> - communicate effectively through simplified language
> - adapt to audiences with multiple needs through visual discourse
> - support legal functions through an optimal mix of language and graphics.[167]

World Commerce and Contracting created a Contract Design Pattern Library that explores various approaches to making long, complex commercial agreements more intelligible for legal and business users.[168] Figure 16 is an image from the Pattern Library that uses legal design to map out the interrelation between documents. This type of diagram can be incorporated into scholarly LegalTech software to demonstrate both qualitative and quantitative theories and findings in transactional law.

> ***Figure 16: Contract Document Map***



C. Case Studies

This section reviews three potential tools of scholarly LegalTech software. Although they are not fully developed works of scholarship, they are indicative of valuable components that can be incorporated into scholarly LegalTech software.

1. Interactive Legaltech Toolkits

Legal scholarship can be significantly enhanced by creating software applications in the form of interactive toolkits. A toolkit that could serve as the basis for scholarly software is the Farmers Market Legal Toolkit (FMLT) maintained by the Center for Agriculture and Food Systems of Vermont Law School. The FMLT describes itself as a toolkit that "includes legal resources, best practice recommendations, and case studies for market leaders on selecting and enhancing business structures, accepting [Supplemental Nutrition Assistance Program (SNAP)] benefits, and managing common risks."[169]

The FMLT includes many elements of scholarly software toolkits, such as:

- an interactive menu system to navigate the substantive content, including case studies
- significant use of graphical elements to help the user understand substantive content, including expandable menus within a page and navigation elements to learn more about a specific topic
- popup text that defines technical terms when the user hovers over them with their mouse

- guides and checklists that reflect a doctrinal understanding of applicable law.

The FMLT is intended for market participants and practitioners and therefore does not include substantive scholarly content. Nonetheless, it is instructive of the medium for scholarly LegalTech. To make such a toolkit scholarly, an author would need to include scholarly content such as:

- arguments and theories such as about the policy behind the law that applies to farmers markets
- well-reasoned frameworks for farmers taking particular actions
- quantitative studies that may be relevant to the FMLT's risk management tools
- engaging in legal reasoning about applicability of the SNAP based user inputs
- policy proposals to expand the SNAP.

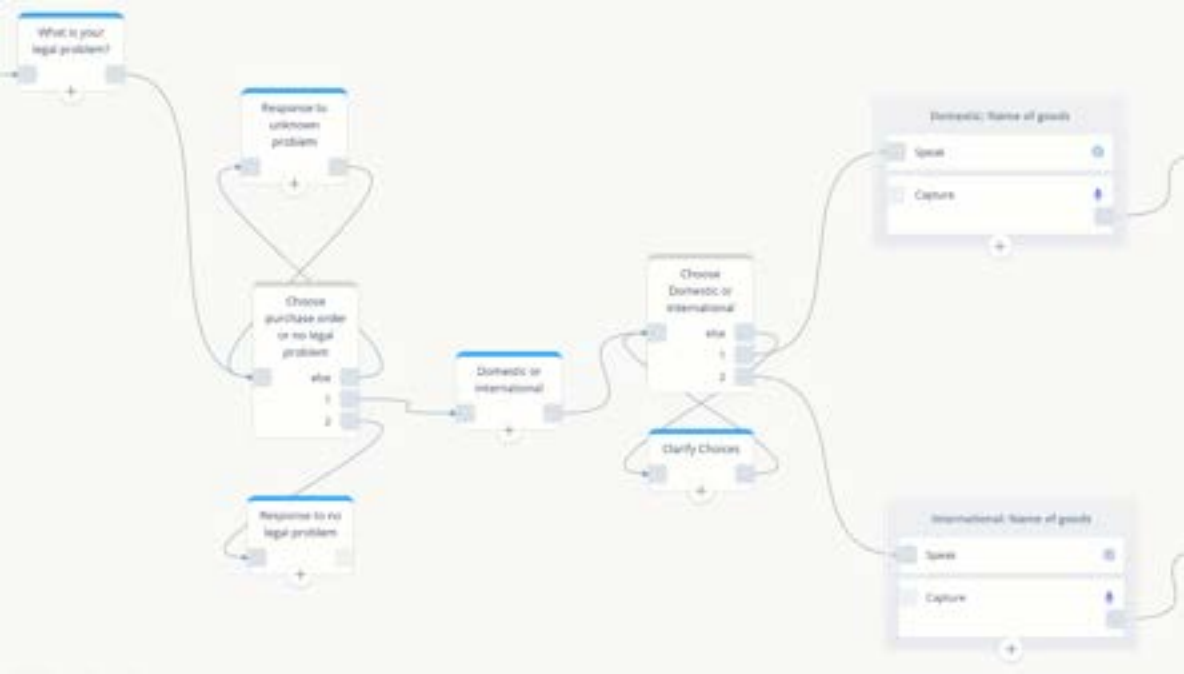
2. Voice Recognition and Contract Drafting

Voice recognition is a type of user interface in which a user speaks to input information instead of typing, using a mouse, or some other interface. Voice recognition can be used by scholarly LegalTech applications to create interactive communications and workflows guided by a user's speech. For example, voice recognition can be used to program an application to ask questions from the user and draft a contract or related documents based on the user's responses and according to the underlying logic of the application.

The present author created an application that generates legally binding purchase orders drafted with specific contract language based upon the user's verbal choices about the jurisdictional aspects of the agreement and business terms. The software application was created using the voice visual programming platform Voiceflow.[170] The application's workflow logic creates a document with the spoken business terms and according to the preprogrammed logic for the applicable legal terms.

The workflow requires the user to choose a jurisdiction, which is a simple form of computational law that embeds legal logic into the application's programming. If the user says "New York," then the document is drafted with New York law language by sending the data to the New York template. If the user says "international" for the jurisdiction, the data is sent to a template that is drafted with language that specifies the United Nations Convention on the International Sale of Goods as the governing law. The underlying legal and business logic is displayed within Voice Flow as shown in the Figure 17:

> ***Figure 17: Voice Recognition Logic***



Once the voice input flow is complete, the application will send data to the relevant purchase order template in Google Docs that will populate the template with the appropriate variables. In this case, it would be Number, Buyer, Goods, and Quantity (a term generally required by the Uniform Commercial Code for legally binding agreements). Figure 18 shows the abbreviated template that the application uses to draft the purchase order.

Figure 18: Purchase Order Template

The application is connected to Google Docs using the Zapier API connector platform.

PURCHASE ORDER

Purchase Order Number: {{Number}}

Buyer: {{Buyer}}

Seller's Supply Company
12345 Road Lane
New York, NY 10001

{{Goods}}	{{Quantity}}
Total:	**{{Quantity}}**

Voiceflow sends the data obtained from a user via voice to a Google Sheet which in turn triggers a workflow using Zapier to populate a Google Docs purchase order template.

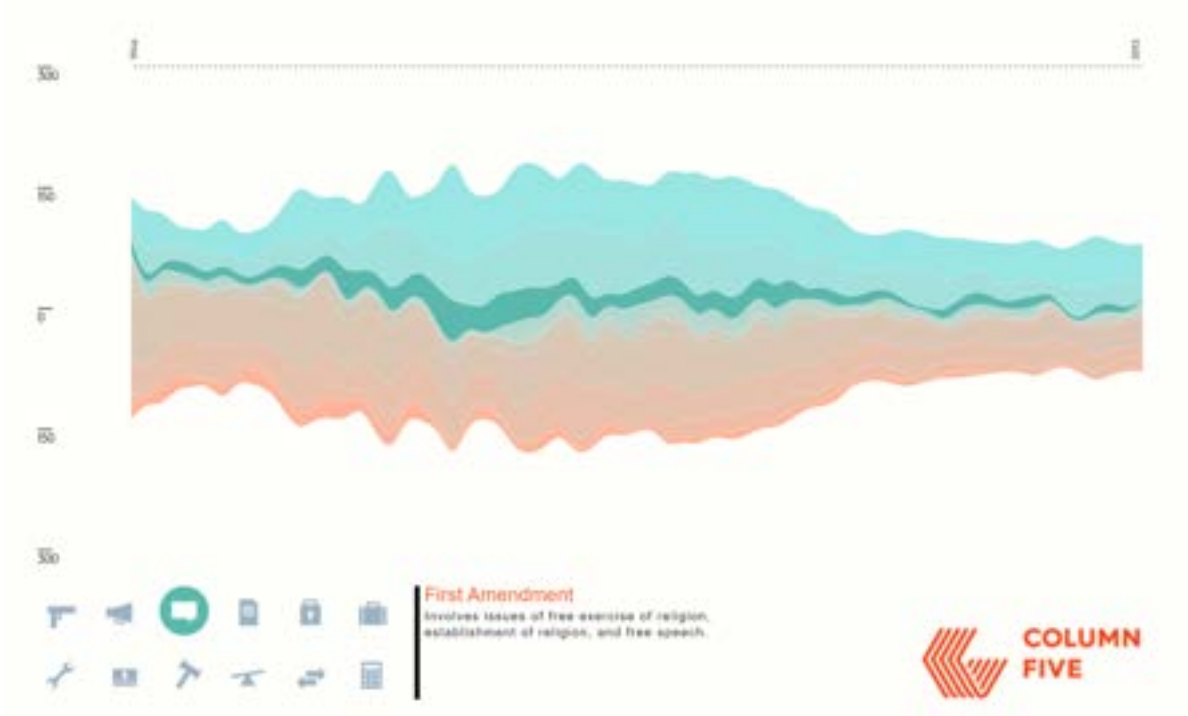
A LegalTech application can include this type of automated drafting functionality to further scholarly goals. For example, transactional legal scholarship may argue for a particular method of drafting contracts that is either more efficient or less likely to subject be to certain types of legal risks based upon an underlying scholarly theory. Consequently, automated drafting can be a way to demonstrate the theory and apply it in various scenarios to further the scholarship. Automated drafting functionality can also incorporate and be responsive to scholarly findings about contract negotiations. This can be accomplished by programming the application to draft contract clauses that are representative of the accompanying scholarly claims or data.

3. Court Data Visualizations

The most traditional form of legal scholarship is doctrinal legal scholarship that analyzes court opinions. The purpose of such scholarship is to provide an explanation and assessment of a set of interrelated cases on a topic or by a particular court system or judge, and often provides guidance or recommendations to courts going forward. Court opinions can be visualized in many ways, and often visualizations are included in scholarship to provide additional explanatory power. These visualizations can include charts that summarize opinions and graphics that provide some type of quantitative analysis of prior case holders.

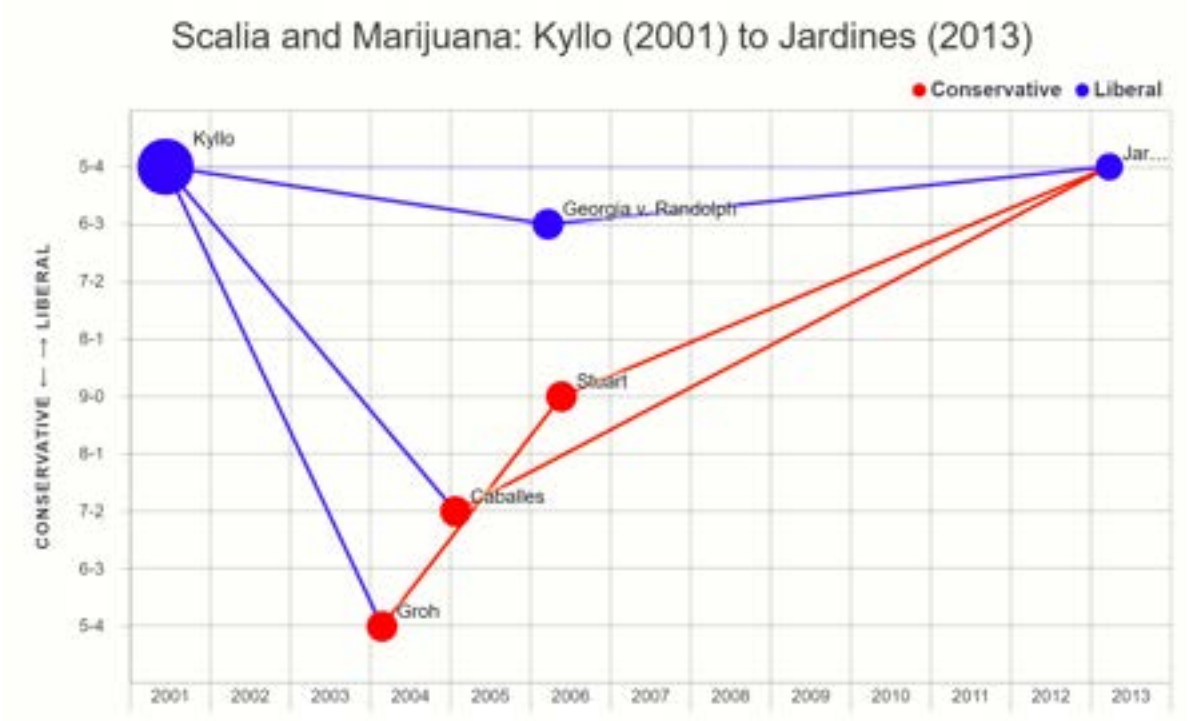
Interactive data visualizations of court opinions can be used within software applications to display various properties of opinions in a visual format that users can interact with to reveal more information. For example, the Supreme Court court data visualization tool created by Column Five allows the user to visualize what portion of the Supreme Court's decisions have been devoted to a particular topic over time. Figure 19 shows a visualization of First Amendment cases when the user clicks on the relevant icon.

> ***Figure 19: Supreme Court First Amendment Cases Visualization***



Another court visualization tool is the Supreme Court Citation Network developed by the Free Law Project and Professor Colin Starger.[171] Figure 20 shows a visualization of Supreme Court marijuana cases in which conservative Justice Antonin Scalia sided with the liberal justices.

Figure 20: Supreme Court Scalia Opinions Visualization



The scholarly value of this visualization is to "illustrate[] how Scalia was actually a liberal on the Fourth Amendment. The map is anchored by two Scalia opinions in which…the supposedly conservative justice sided with an accused marijuana criminal – *Kyllo* and *Jardines*."[172] In other words, this visualization provides scholarly value by clarifying and communicating knowledge about a Supreme Court justice's voting record.

6. Conclusion

Due to the increasing power and accessibility of software, humanities scholarship and social science research has the potential to undergo a fundamental transformation in how it is developed and communicates knowledge. By turning written scholarship into functional software applications, scholars and researchers can vastly improve the ability of their works to communicate their analysis and findings. Scholarly software can also improve the quality of scholarship by adding data-oriented programmable functionality and interactive graphics to traditional scholarly publications. This is because the act of programming and creating interactive graphics can make scholarly analysis more thorough, precise, and receptive to feedback.

Humanities scholars and social science researchers should accordingly create scholarly software applications to increase the quality of their work and the ability to communicate their analysis and findings. Building software has become so accessible that minimal training is required to program software applications that make use of databases, computational logic, and interactive graphics. Likewise, publicly available data and specialized software tools that perform machine learning, data visualization, and other data-oriented tasks are becoming increasingly available and accessible for scholars to produce and communicate their scholarship. As software continues to proliferate and has the potential to vastly improve most areas of life, software applications should be developed by scholars to advance the goals of the humanities.

Houman is Professor of Law and Director, Innovation Center for Law and Technology, New York Law School. B.A. 1998, University of California at Berkeley; J.D. 2002, University of Southern California. *I would like to thank New York Law School faculty workshop participants, Madiha Zahrah Choksi, James Grimmelmann, Colin Starger, and Paul Ohm for their comments. All errors and opinions are my own.*

Footnotes

1. Already, the distinction between "document" and "application" is rapidly collapsing with the advent of software that enables users to include software components inside and alongside written text. See Tom Warren, Microsoft Previews the Future of Office Documents with Fluid Framework for the Web, Nov. 4. 2019, https://www.theverge.com/2019/11/4/20942031/microsoft-fluid-framework-office-web-preview. ↩

2. Fred Gibbs, Digital Humanities Definitions by Type 290, in Defining Digital Humanities: A Reader (2013). ↩
3. See Section III.A.2. ↩
4. For some notable exceptions see Paul Ohm, Computer Programming and the Law: A New Research Agenda, 54 Vill. L. Rev. 117 (2009) (noting "[c]omputer programming can be used for novel and more effective methods of *publishing and communicating*" legal scholarship in the form of blogs and web applications) (emphasis added); Geoffrey Rockwell, Is Humanities Computing an Academic Discipline?, in Defining Digital Humanities: A Reader 22 (Terras et al. eds. 2013) (arguing that scholarship in the form of interactive multimedia "is not just a tool for study of other objects, it is the *delivery vehicle for content* . . . designed to convince, delight, or instruct") (emphasis added); James Coltrain & Stephen Ramsay, Can Video Games Be Humanities Scholarship?, Debates in the Digital Humanities 2019 (noting that software in the form of video games can be a powerful vehicle for "capturing the scholarly experience, particularly as it relates to articulations of new *arguments and interpretations*") (emphasis added). ↩
5. See Columbia University, Digital Humanities, https://digitalhumanities.columbia.edu/projects/. ↩
6. Donald Knuth, Literate Programming 99 (1992); Literate Programming and Eve, http://witheve.com/deepdives/literate.html. Indeed, academic articles themselves interspersed with code can constitute literate programming. See Paul Ohm, *Computer Programming and the Law: A New Research Agenda*, 54 Vill. L. Rev. 117 (2009) ("This is the world's first law review article that is also a working computer program. It owes its chimeric nature to a technique known as 'literate programming.'"). ↩
7. This is consistent with the widely used three-tier software architecture. See Ariel Ortiz Ramirez, Three-Tier Architecture Software, Linux Journal, July 1, 2000, https://www.linuxjournal.com/article/3508. ↩
8. The Artifice of Dialogue: Thinking Through Scepticism in Hume's Dialogues, in Stéfan Sinclair & Geoffrey Rockwell book Hermeneutica (2016). http://hermeneuti.ca/artifice-dialogue/. ↩
9. Voyant, Tools, https://voyant-tools.org/?corpus=hume.dialogues.by.book&query=natural. ↩
10. See Section III.B.1. ↩
11. How To Build A Quora Clone Without Code, https://bubble.io/blog/build-quora-clone-no-code/#display-dynamic-content-in-a-feed. ↩

12. See Section III.B.2. ↩
13. Webflow, The Node Code Revolution 28 (2020), https://ebooks.webflow.com/chapter/why-no-code. ↩
14. Ashok Arora, Computer Fundamentals and Applications 74 (2015). This does not mean that computational tasks are inherently deterministic such that the same input(s) will always produce the same output in a given program. Programming languages can be ambiguous. See James Grimmlemann, All Smart Contracts Are Ambiguous 2 J. L. & Innov. 1, 11-14 (2019). A set of programming instructions that perform a specific task is a procedure. ↩
15. See also David Evans, Introduction to Computing Explorations, in Language, Logic, and Machines (2011) (defining computer science as "the study of information processes."). ↩
16. Oxford Dictionary of Computer Science 111 (2016). ↩
17. Harold Abelson, Gerald Jay Sussman & Julie Sussman, Structure and Interpretation of Computer Programs 6-15 (1996) (hereinafter "SICP"). ↩
18. Expressive Power, Wikipedia, https://en.wikipedia.org/wiki/Expressive_power_(computer_science) ↩
19. See SICP, at 75 ("One of the things we should demand from a powerful programming language is the ability to build abstractions by assigning names to common patterns and then to work in terms of the abstractions directly . . . . To express such patterns as concepts, we will need to construct procedures that can accept procedures as arguments or return procedures as values."); Evans at 55. Indeed, much of programming consists of manipulating and reasoning about such higher-order procedures. See SICP, at 108 (noting that "higher-order procedures enhance the power of our language by enabling us to manipulate, and thereby to reason in terms of, general methods of computation. is much of the essence of programming."). Likewise, although programs regularly use simple data such as numbers, composite or compound data such as lists of data also achieves the goals of higher abstraction. See SICP, at 108 (noting that "to elevate the conceptual level at which we can design our programs, to increase the modularity of our designs, and to enhance the expressive power of our language."); Composite Data Type, https://knowthecode.io/labs/types-of-data/episode-3; Evans at 82 (defining "a List as a data structure that can contain any number of elements."). Lists of data known as an array typically have a key associated with each value that gives you the actual value in the array. ↩

20. See Section III.B.2. ↩
21. See generally Evans at 153-166. ↩
22. See generally Evans at 167-178. ↩
23. See Unicode: Overview, https://home.unicode.org/basic-info/overview/. Programs typically limit how expressions operate so as to prevent errors such as attempting to add "2" plus "2" by accidentally adding "two" plus "two. See John M. Zell, Python Programming: An Introduction to Computer Science 158 (2004). ↩
24. See SICP, at 22. ↩
25. See Zell at 176. ↩
26. Zell at 195-197. ↩
27. These algorithms provide much of the benefits of modern computing, including artificial intelligence. A fundamental distinction in computer science between different types of repetitive tasks is whether they are performed over and over again successively without change to the task (iteration) or whether they proceed by continuously executing the rules on subdivisions of data until a solution is reached (recursion). ↩
28. SICP at 108. ↩
29. SICP at 108. ↩
30. Objects possess similar computational properties such as identity data and financial data, and are typically organized into classes and subclasses where the subclasses have certain general properties of the parent class (known as inheritance). See Evans at 200-202. ↩
31. Zell at 5. ↩
32. Different languages may fall within broader programming paradigms such as object oriented programming or functional programming. Ultimately, all human-readable programming languages must be converted ("compiled") into binary notation so that its rules and data can be processed by the computer's circuits (which can only process 0s and 1s). Zell at 6. ↩
33. See Section III.B.1. ↩
34. See Section III.B.1. ↩
35. See generally SICP section 1.2. ↩

36. Ian Sommerville, Software Engineering 18 (10th ed. 2016). ↩
37. Sommerville at 20, ↩
38. Sommerville at 23. ↩
39. Sommerville at 20, 22. ↩
40. Sommerville at 18. ↩
41. Sommerville at 23. ↩
42. Sommerville at 49-50. Problems with agile approaches are quickly outdated documentation and requiring constant refactoring to maintain system wide consistency and integrity. Id. ↩
43. Sommerville at 102. Functional requirements are statements about the services that should be provided and how the software should perform. Non-functional requirements include timing, security, and compliance and typically pertain more to system requirements. Id. at 105. ↩
44. Sommerville at 111. ↩
45. Sommerville at 117-119. Formalized models using standardized symbols such as the Uniform Modeling Language can provide a highly detailed and clear illustration of requirements and specifications. See generally Sommerville at Chapter 5. ↩
46. Sommerville at 168. ↩
47. Sommerville at 169. ↩
48. Sommerville at 203. ↩
49. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 1 (7th ed. 2020). ↩
50. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 4 (7th ed. 2020) (“the processing of data to draw conclusions, and infer rules or decision procedures, which are then used to drive business decisions”). ↩
51. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 29 (7th ed. 2020) (“the processing of data to draw conclusions, and infer rules or decision procedures, which are then used to drive business decisions”). ↩

52. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 2 (7th ed. 2020). ↩
53. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 8 (7th ed. 2020). ↩
54. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 8 (7th ed. 2020). See also Chris Hastie, Zero to Snowflake: An Introduction to Semi-Structured JSON Data Formats, Jan. 21, 2020, https://interworks.com/blog/chastie/2020/01/21/zero-to-snowflake-an-introduction-to-semi-structured-json-data-formats/. ↩
55. How to Fetch and Display JSON Data in HTML Using JavaScript, https://howtocreateapps.com/fetch-and-display-json-html-javascript/ ↩
56. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 8 (7th ed. 2020). These four operations parallel what is commonly known as CRUD database operations: "create, read, update and delete." Id. at 419. ↩
57. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Database Systems Concepts 24-25 (7th ed. 2020). ↩
58. Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 3 (2nd ed. 2009) 4 (noting that "[i]nitially, what people shared over the Internet consisted mostly of static information found in files . . . No longer was it sufficient to build a web site (as opposed to a motley collection of web pages). It became necessary to design a web application."). ↩
59. See https://www.statista.com/statistics/510350/worldwide-public-cloud-computing/ ↩
60. Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 3 (2nd ed. 2009) ↩
61. The TCP/IP suite is made up of the Network Interface layer, the Internet layer, the Transport layer, and the Application layer Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 2 (2nd ed. 2009) ↩
62. Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 2 (2nd ed. 2009) ↩

63. Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 3.3.1 (2nd ed. 2009) ↩
64. A web resource is a web-available file, database, or "thing, entity, or action that can be identified, named, addressed, handled, or performed, in any way whatsoever, on the Web.". See https://en.wikipedia.org/wiki/Representational_state_transfer. ↩
65. Leon Shklar & Rich Rosen, Web Application Architecture: Principles, Protocols and Practices Ch. 5.3 (2nd ed. 2009). ↩
66. See Section III.B.2. ↩
67. Jason Beaird & James George, The Principles of Beautiful Web Design 5 (3rd Ed. 2014) (emphasis added). ↩
68. Jason Beaird & James George, The Principles of Beautiful Web Design 15-26 (3rd Ed. 2014) ↩
69. Jason Beaird & James George, The Principles of Beautiful Web Design 61-76 (3rd Ed. 2014) ↩
70. Jason Beaird & James George, The Principles of Beautiful Web Design 148-150 (3rd Ed. 2014) ↩
71. The primary purpose of scholarly software is not to create better software from a technical or engineering point of view; nor is it to develop commercial applications. Scholarly software is likely not well-suited as a commercial product, given the heavy use of documentation, citation, research, and other scholarly methods that would otherwise proceed too slowly to meet customer needs and market demands. However, scholarly software may lay the basis for the development of commercial products by providing a foundational form of research and development into the value of a technology-driven approach to answering certain foundational questions that are of broad commercial interest such as the nature of a market or the preferences of users. Scholarship in areas such as business, management, and operations would be particularly well-suited for commercial application or development. Scholarly software will tend to have greater commercial applicability to the extent that its subject matter is closer to commercial fields of inquiry. For such fields, scholarly software may be especially valuable in directly or indirectly providing deep and detailed research into potential customers. This is true because a leading cause of new business ideas failing is due to being insufficiently tailored to what actual customers are willing to pay for. ↩
72. *See generally* Stuart Russell & Peter Norvig, *Artificial Intelligence: A Modern Approach* (4th ed. 2020). ↩
73. For an overview of usability see Travis Lowdermilk, User-Centered Design: A Developer's Guide To Building User-Friendly Applications (2013); David Platt, The Joy of UX: User Experience and

Interactive Design for Developers (Usability) (2013); Carrie Hane & Mike Atherton, Designing Connected Content: Plan and Model Digital Products for Today and Tomorrow (2017). ↩

74. Jeff Johnson, Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines (2010). See also generally Jon Yablonski, Laws of UX: Using Psychology to Design Better Products & Services; Liam J.Bannon, From Human Factors to Human Actors: The Role of Psychology and Human-Computer Interaction Studies in System Design (1995). ↩

75. Ni Eyal, Hooked: How to Build Habit-Forming Products (2014). ↩

76. Alain Samson, An Introduction to Behavioral Economics, behavioraleconomics.com, https://www.behavioraleconomics.com/resources/introduction-behavioral-economics/. ↩

77. J.P. Kesan & R.C. Shah, Setting Software Defaults: Perspectives from Law, Computer Science and Behavioral Economics, 82 Notre Dame L. Rev. 583 (2006-2007); Behavioral Software Engineering: A definition and systematic literature review, P Lenberg, R Feldt, LG Wallgren - Journal of Systems and software, 2015. ↩

78. Linguistics includes the study of written and visual communication tools. See What is Linguistics, UCLA Department of Linguistics, https://linguistics.ucla.edu/undergraduate/what-is-linguistics/ ↩

79. Inger Lytje, A Cognitive Linguistic Perspective on the User Interface, in Usability Evaluation and Interface Design: Cognitive Engineering, Intelligent Agents, and Virtual Reality, 229 (Michael J. Smith et al eds. 2001). ↩

80. What are the Computational Humanities, Arts, and Social Sciences?, Aug 29, 2013, http://ichass.illinois.edu/index.php/what-are-the-computational-humanities-arts-and-social-sciences/. See also https://dhdebates.gc.cuny.edu/page/cfp-computational-humanities (defining digital humanities). ↩

81. Software Sustainability Institute, Manifesto, https://www.software.ac.uk/about/manifesto. ↩

82. James Hetherington, The Craftsperson and the Scholar, Nov. 9, 2012, https://www.software.ac.uk/blog/2016-10-07-craftsperson-and-scholar. ↩

83. See generally Stefan Jänicke et al., On Close and Distant Reading in Digital Humanities: A Survey and Future Challenges, EuroVis (2015); Ted Underwood, Seven Ways Humanists Are Using Computers to Understand Text, The Stone and the Shell, June 4, 2015; Adam Mann, Computational Social Science, 113 Core Concepts 468 (2016), https://www.pnas.org/content/pnas/113/3/468.full.pdf; Mario Molina & Filiz Garip, Machine Learning for Sociology, 45 Annual Review of Sociology 27 (2019); Sander Münster & Melissa Terras, The Visual Side of Digital Humanities: A Survey on Topics,

Researchers, and Epistemic Cultures, 35 Digital Scholarship in the Humanities, 366 (2020), https://doi.org/10.1093/llc/fqz022. Initiatives to standardize how text is digitized have also been developed. See TEI: Text Encoding Initiative, https://tei-c.org/ ("a consortium which collectively develops and maintains a standard for the representation of texts in digital form. Its chief deliverable is a set of Guidelines which specify encoding methods for machine-readable texts, chiefly in the humanities, social sciences and linguistics"). ↩

84. Matthew G. Kirschenbaum, The Remaking of Reading: Data Mining and the Digital Humanities, The National Science Foundation symposium on next generation of data mining and cyber-enabled discovery for innovation, 2007, https://www.csee.umbc.edu/~hillol/NGDM07/abstracts/talks/MKirschenbaum.pdf. ↩

85. https://www.folgerdigitaltexts.org/api. ↩

86. https://www.folgerdigitaltexts.org/api/Bennett_FDT_4Points.pdf. ↩

87. See, e.g., the online-only Harvard Law Review Forum, https://harvardlawreview.org/topics/forum/. ↩

88. See Edward R. Tufte, The Visual Display of Quantitative Information (2001). ↩

89. Tim Brown, Design Thinking, 86 Harv. Bus. Rev. 84, 86 (2008). ↩

90. See Are You a Brain in a Vat?, Philmaps.cm; The Brain in a Vat Argument, Internet Encyclopedia of Philosophy, https://www.iep.utm.edu/brainvat// ↩

91. Richard K. Sherwin et al., Law in the Digital Age: How Visual Communication Technologies Are Transforming the Practice, Theory, and Teaching of Law, 12 B.U. J. Sci. & TECH. L. 227, 230 (2006).

See also Darren Newbury, Making Arguments with Images: Visual Scholarship and Academic Publishing, in The SAGE Handbook of Visual Research Methods 654-55 (Pauwels & Mannay eds. 2020). ↩

92. See Aaron Marcus, Principles of Effective Visual Communication for Graphical User Interface Design. See also Meredith Davis & Jamer Hun Visual Communication Design: An Introduction to Design Concepts in Everyday Experience (2017). See also generally Handbook of Visual Communication: Theory, Methods, and Media, Sheree Josephson, James Kelly, Ken Smith (2020); Matthew O. Ward, Georges Grinstein, Daniel Keim, Interactive Data Visualization Foundations, Techniques, and Application (2010). ↩

93. See generally Stéfan Sinclair, Stan Ruecker, and Milena Radzikowska, Information Visualization for Humanities Scholars, in Literary Studies in the Digital Age: An Evolving Anthology (2013) ↩
94. See Darren Newbury, Making Arguments with Images: Visual Scholarship and Academic Publishing, in The SAGE Handbook of Visual Research Methods 654-55 (Pauwels & Mannay eds. 2020). ↩
95. See generally, Mapping the Republic of Letters, http://republicofletters.stanford.edu/; Nicole Coleman, Mapping the Republic of Letters, Open Knowledge Foundation, March 22, 2012, https://blog.okfn.org/2012/03/22/mapping-the-republic-of-letters/ ↩
96. Museum of Modern Art, Inventing Abstraction 1910-1925, http://www.moma.org/interactives/exhibitions/2012/inventingabstraction/. ↩
97. See also Report from Dagstuhl Seminar 18482, Network Visualization in the Humanities, https://drops.dagstuhl.de/opus/volltexte/2019/10359/pdf/dagrep_v008_i011_p139_18482.pdf. ↩
98. See generally Design of Visualizations for Human-Information Interaction: A Pattern-Based Framework (Ebert & Elmqvist eds. 2016); R. Bühling, M. Wißner M. & E. André, Visual Communication in Interactive Multimedia, in The 11th International Symposium on Smart Graphics (2011). ↩
99. History of Philosophy Summarized, https://www.denizcemonduygu.com/philo/. ↩
100. About, Six Degrees of Francis Bacon, http://www.sixdegreesoffrancisbacon.com/about. ↩
101. See generally Stéfan Sinclair, Stan Ruecker & Milena Radzikowska, Information Visualization for Humanities Scholars, in Literary Studies in the Digital Age: An Evolving Anthology (2013). ↩
102. For a list of argument mapping tools see http://www.phil.cmu.edu/projects/argument_mapping/. ↩
103. See generally Matthew O. Ward, Georges Grinstein & Daniel Keim, Interactive Data Visualization Foundations, Techniques, and Applications (2010). ↩
104. Six Degrees of Francis Bacon, http://www.sixdegreesoffrancisbacon.com/. ↩
105. Carrie Williams, Research Methods, 5 Journal of Business & Economic Research 3, 65 (March 2007). ↩
106. See Carrie Williams, Research Methods, 5 Journal of Business & Economic Research 3, 68-69 (March 2007). ↩

107. Psytoolkit, https://www.psytoolkit.org/. ↩
108. History of Philosophy Visualized and Summarized, https://www.denizcemonduygu.com/philo/. ↩
109. Carrie Williams, Research Methods, 5 Journal of Business & Economic Research 3, 67 (March 2007). ↩
110. See Severino Ribecca, The Data Visualisation Catalogue, https://datavizcatalogue.com/. See also generally Digital Humanities: Visualizations, New York University, https://guides.nyu.edu/dighum/viz. ↩
111. See generally Claus O. Wilke, Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures (2019); Cole Nussbaumer Knaflic, Storytelling with Data: A Data Visualization Guide for Business Professionals (2015). ↩
112. Martha Minnow, Archetypal Legal Scholarship: A Field Guide 63 J. Legal Educ. 65, 66 (2013); https://www.mercatus.org/system/files/Ellig-SenateHSGACommittee-Testimony.pdf (identifying the methodology of regulatory impact analysis). ↩
113. Derek Poppert, Navigating the Field of Civic Tech, Aug. 18, 2018, https://medium.com/tradecraft-traction/navigating-the-field-of-civic-tech-c1f9670c8f69. ↩
114. Id. ↩
115. See generally Technology Tools in Human Rights (explaining "how human rights defenders (HRDs) are using technology tools to communicate, analyse and manage information, identify new incidents and archive data"), https://library.theengineroom.org/humanrights-tech/. ↩
116. Try Blockly, https://developers.google.com/blockly. ↩
117. Welcome to the Era of No-Code in the Enterprise, https://www.unqork.com/resources/articles/welcome-to-the-era-of-no-code-in-the-enterprise. See also ("No-code's visual workflows create a common language that both business users and engineers can speak."), https://www.unqork.com/resources/blog-articles/explaining-no-code-to-engineers ↩
118. Humanizing Development with No-Code, Unqork, https://www.unqork.com/resources/blog-articles/humanizing-development-with-no-code. ↩
119. Jason Morris, Blawx: Rules as Code Demonstration, MIT Computational Law Report, Aug. 14, 2020, https://law.mit.edu/pub/blawxrulesascodedemonstration. ↩

120. Humanizing Development with No-Code, Unqork, https://www.unqork.com/resources/blog-articles/humanizing-development-with-no-code. ↩
121. Entity Relationship Diagram, SmartDraw, https://www.smartdraw.com/entity-relationship-diagram/. ↩
122. Mike Chapple, Entity Relationship Diagram, March 8, 2019, https://www.lifewire.com/entity-relationship-diagram-1019253. ↩
123. Discovering Millions of Datasets on the Web, Google, Jan. 23, 2020 https://www.blog.google/products/search/discovering-millions-datasets-web/. ↩
124. See, e.g., How No-Code Machine Learning Algorithms Work, Obviously.ai, May 14, 2020, https://www.obviously.ai/post/how-no-code-machine-learning-algorithms-work; Why Datawrapper, https://www.datawrapper.de/why-datawrapper/; Powerful topic modeling in clicks… without coding!, Gyana, https://www.gyana.co.uk/post/topic-modeling-in-clicks-without-coding/; Ludwig, https://ludwig-ai.github.io/ludwig-docs/ ("Ludwig is a toolbox that allows to train and test deep learning models without the need to write code."); About, Lobe, https://lobe.ai/about; Northstar ("an interactive data science platform that . . . empowers users without programming experience, background in statistics or machine learning expertise") http://northstar.mit.edu/; About Us, Metaranx ("Metaranx is a drag-and-drop software platform that allows you to build your very own AI tools, applications, and creations without having to code", https://www.metaranx.com/about-us; The Wolfram Approach to Machine Learning ("making state-of-the-art machine learning in a full range of applications accessible even to non-experts"), https://www.wolfram.com/featureset/machine-learning/; Introduction to RAWGraphs, RAWGraphs ("an open source data visualization framework built with the goal of making the visual representation of complex data easy for everyone"), https://rawgraphs.io/learning/introduction-to-rawgraphs/; Cloud AutoML https://cloud.google.com/automl/. See also generally Rohit Chatterjee, Top 10 Tools For No-Code AI & ML, Analytics India, Feb. 11. 2020, https://analyticsindiamag.com/top-10-tools-for-no-code-ai-ml/. ↩
125. See, e.g., Allen Yang, No-Code Deep Learning with Bubble & Peltarion, Bubble, July 16, 2020, https://bubble.io/blog/bubble-peltarion-machine-learning-ai/. ↩
126. Data predictions for the Insurance industry, Obviously.ai, https://www.obviously.ai/case-studies/insurance. ↩
127. Programmable Web, Categories, https://www.programmableweb.com/category/all/apis. ↩
128. See Section II.A.3. ↩

129. What is Bubble, Bubble,https://bubble.io/blog/what-is-bubble/. For a comprehensive list of other platforms and tools for building software see No Code List, https://nocodelist.co/?search=data. ↩

130. Bubble Manual, Element Custom States, https://manual.bubble.io/working-with-data/element-custom-states. ↩

131. Bubble Reference, HTML Element, https://bubble.io/reference#Elements.HTML ↩

132. Bubble Reference, Chart.JS, https://bubble.io/reference#Plugins.chartjs. ↩

133. Bubble Manual, Adding API Connections, https://manual.bubble.io/building-plugins/adding-api-connections ↩

134. See Section IV.B.3. ↩

135. See Section III.A.3.C. ↩

136. See Section II.A. ↩

137. For an overview of RESTful APIs, see Kenneth Lange, The Little Book on Rest Services (2016), https://www.kennethlange.com/books/The-Little-Book-on-REST-Services.pdf. ↩

138. About, Programmable Search Engine, https://programmablesearchengine.google.com/about/. ↩

139. Google Custom Search, Using REST, https://developers.google.com/custom-search/v1/using_rest ("Search query - Use the **q** query parameter to specify your search expression."). ↩

140. See Section III.B.1. ↩

141. Bubble Manual, Working With Data Key Concepts, https://manual.bubble.io/working-with-data/key-concepts; Bubble Reference, Data Sources and Operations, https://bubble.io/reference#Data. ↩

142. Bubble Reference, Data Messages, https://bubble.io/reference#Data.Messages. ↩

143. Bubble Manual, Connecting Types With Each Other, https://manual.bubble.io/working-with-data/connecting-types-with-each-other. ↩

144. Bubble Reference, https://bubble.io/reference#Data.Messages.text.extract_regex; Regular Expression, Wikipedia, https://en.wikipedia.org/wiki/Regular_expression ↩

145. *See generally* Michael Genesereth "Computational Law: The Cop in the Backseat" White Paper, CodeX—The Stanford Center for Legal Informatics (2015). Available at: https://law.stanford.edu/publications/computational-law-the-cop-in-the-backseat/. ↩

146. See generally Phoebe C. Ellsworth, Legal Reasoning, in The Cambridge Handbook of Thinking and Reasoning 685-704 (eds. K. J. Holyoak & R. G. Morrison Jr. 2005). .https://scholarlycommons.law.hofstra.edu/cgi/viewcontent.cgi?article=2283&context=hlr; Huhn, Wilson Ray, The Stages of Legal Reasoning: Formalism, Analogy, and Realism (2003). Villanova Law Review, Vol. 48, p. 305, 2003; University of Akron Legal Studies Research Paper, https://ssrn.com/abstract=1704042. ↩

147. Relatedly, an entire discipline of legal ontology seeks to apply formal methods to legal knowledge and reasoning, and often applies it to software applications. See generally Núria Casellas Legal Ontology Engineering: Methodologies, Modelling Trends, and the Ontology of Professional Judicial Knowledge (2011). ↩

148. See also Michael Genesereth, Computational Law: The Cop in the Backseat (noting that "[p]hilosophically, Computational Law is closely aligned with the Legal Formalism school of Jurisprudence."), http://logic.stanford.edu/publications/genesereth/complaw.pdf. ↩

149. Kelso, Louis O., "Does the Law Need a Technological Revolution" 18 Rocky Mntn. L. Rev. 378 (1945) ↩

150. Flood, Mark D. and Goodenough, Oliver R., Contract as Automaton: The Computational Representation of Financial Agreements (March 26, 2015), Office of Financial Research Working Paper No. 15-04., https://ssrn.com/abstract=2648460. ↩

151. Carolyn E.C. Paris, Drafting for Corporate Finance: Concepts, Deals, and Documents 163 (2007). ↩

152. Harry Surden, Computable Contracts 46 UC Davis L. Rev. 629, 637 (2012) (arguing that "parties can make certain contract terms 'understandable' to a computer by translating the meaning of the term into a set of consonant computer instructions"). ↩

153. MetaLex, http://www.metalex.eu/. ↩

154. LegalXML, http://www.legalxml.org/. ↩

155. LSP Working Group, Developing a Legal Specification Protocol: Technological Considerations and Requirements, Feb. 14, 2019, https://www-cdn.law.stanford.edu/wp-content/uploads/2019/03/LSPWhitePaperJan1119v021419.pdf. ↩

156. S. Erdelez & O'Hare, Legal informatics: Application of Information Technology in Law, 32 Ann. Rev. Infor. Science &Tech. 367-402 (1997). ↩

157. Marc Lauritsen & Quinten Steenhuis, Substantive Legal Software Quality: A Gathering Storm?, (2019) ("Automated legal services may be the best hope for access to justice and legal wellness for billions of our fellow humans. AI & Law activists are encouraged to find ways to bring their utensils to the feasts of knowledge automation that lie ahead."), https://dl.acm.org/doi/pdf/10.1145/3322640.3326706. ↩

158. Daniel Faggella, AI in Law and Legal Practice – A Comprehensive View of 35 Current Applications, March 14, 2020, https://emerj.com/ai-sector-overviews/ai-in-law-legal-practice-current-applications/; Lauri Donahue, A Primer on Using Artificial Intelligence in the Legal Profession, January 03, 2018, https://emerj.com/ai-sector-overviews/ai-in-law-legal-practice-current-applications/; Bernhard Waltl & Roland Vogl, Explainable Artificial Intelligence – the New Frontier in Legal Informatics, Jusletter IT 21, 22 (2018) (identifying numerous different types of AI reasoning approaches in law legal scholarship). ↩

159. IACCM-Capgemini Automation Report (2018) (providing an overview of contract automation tools), https://s3.eu-central-1.amazonaws.com/iaccmportal/resources/files/10162_iaccmcapgeminiautomationreport.pdf ↩

160. Martha Minnow, Archetypal Legal Scholarship: A Field Guide, 63 J. Legal Educ. 65 (2013), https://jle.aals.org/home/vol63/iss1/4/. ↩

161. See generally id. ↩

162. See Section II.B.2. ↩

163. See, e.g., Paul Ohm, *Computer Programming and the Law: A New Research Agenda*, 54 Vill. L. Rev. 117 (2009) (arguing that "[l]egal scholars should create computer programs to develop better tools, data, and insights"); Daniel Martin Katz, Michael J. Bommarito & Josh Blackman, *A General Approach for Predicting the Behavior of the Supreme Court of the United States*, PLoS ONE (2017); Frank Fagan, *Big Data Legal Scholarship: Toward a Research Program and Practitioner's Guide*, 20 Virginia J. L. & Tech. 2 (2016). Given the textual nature of cases, statutes, contracts, and other legal materials, quantitative studies in law often employ technologies such as machine learning and natural language processing. See generally Harry Surden, Essay, Machine Learning and Law, 89 Wash. L. Rev. 87 (2014), https://digitalcommons.law.uw.edu/wlr/vol89/iss1/5; John Nay, Natural Language Processing and Machine Learning for Law and Policy Texts (April 7, 2018), https://ssrn.com/abstract=3438276. ↩

164. See J.B. Ruhl and Daniel M. Katz, Measuring, Monitoring, and Managing Legal Complexity, 101 Iowa L. Rev. 191, 214-221 (2015) (showing the measurement of legal complexity with trees, network diagrams, and other visualizations). ↩

165. See Section II.3.C. ↩

166. See Margaret Hagan, Law by Design, https://www.lawbydesign.co/legal-design/. ↩

167. See Gerlinde Berger-Walliser, Thomas D. Barton & Helena Haapio, From Visualization to Legal Design: A Collaborative and Creative Process, 54 Am. Bus. L.J. 347, 365 (2017). ↩

168. Contract Pattern Design Library, https://contract-design.worldcc.com/. ↩

169. Farmers Market Legal Toolkit, https://farmersmarketlegaltoolkit.org/. ↩

170. Braden Ream, Intro to Voiceflow Series, Voiceflow, May 24, 2020, https://learn.voiceflow.com/en/articles/2563825-intro-to-voiceflow-series. ↩

171. Our Newest Launch: A SCOTUS Data Viz Tool, Free Law Project, Feb. 22, 2016, https://free.law/2016/02/22/our-newest-launch-a-scotus-data-viz-tool/. ↩

172. Scalia and Marijuana: Kyllo (2001) to Jardines (2013), https://www.courtlistener.com/visualizations/scotus-mapper/484/kyllo-2001-to-jardines-2013/. ↩