

# Me.TV: A Visual Programming Language and Interface for Dynamic Media Programming

Vivian Chan Diep

B.S. in Management, Boston College (2012)

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of Master of Science at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
September 2015

©Massachusetts Institute of Technology 2015. All rights reserved.

## Signature redacted

Author: .....  
Program in Media Arts and Sciences  
August 7, 2015

## Signature redacted

Certified by: .....  
Andrew Lippman  
Senior Research Scientist

## Signature redacted

Accepted by: .....  
Prof. Pattie Maes  
Academic Head, Program in Media Arts and Sciences

# **Me.TV: A Visual Programming Language and Interface for Dynamic Media Programming**

by  
Vivian Chan Diep

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,  
on August 7, 2015, in partial fulfillment of the requirements for the degree of Master of Science

## **Abstract**

Sit back. Relax. And don't touch that dial.

The culture of televised media experiences has changed very little since the time it began in the 1930s, but new internet technologies, like Netflix, Hulu, and Youtube, are now quickly forcing major change. Although these new internet technologies have given the viewer more control than the historical dial, they have also left behind some of the greatest contributions of traditional television. These contributions include not just the well-favored simplicity of use, but also the sense of social experience and connectedness, the ease and continuity of scheduled programming, and the understanding that television is now, current, and pulsing.

This thesis presents Me.TV, a web platform that combines the benefits of traditional television and on-demand viewing for a new experience that allows us to let go, watch the same channels as our friends, flip our preferences around, get constant, current content, and still have control over the type and timing of content. To make this experience possible, we present a visual programming language at the center of the Me.TV platform that enables users to create complex rules with simple interactions. The visual language constructs allow users to create static preferences, such as genre constraints, and plan for non-static ones, such as a current mood, in as many channels as they want. To support the Me.TV programming language, the platform comprises of an editor, translation engine, application programming interface, video player and navigation dashboard, which we prototype in this thesis as a javascript web application.

*Work reported herein was funded by the Media Lab Consortium and the Ultimate Media Program.*

Thesis Supervisor: Andrew Lippman  
Title: Senior Research Scientist

# Me.TV: A Visual Programming Language and Interface for Dynamic Media Programming

by  
Vivian Diep

Signature redacted

Thesis Advisor .....  
Andrew B. Lippman  
Senior Research Scientist and Associate Director, MIT Media Lab

Signature redacted

Thesis Reader .. .....  
Matt Carroll  
Research Scientist, MIT Media Lab

Signature redacted

Thesis Reader .....  
Cesar A. Hidalgo  
Associate Professor, MIT Media Lab

## Acknowledgements

For the many stories, crazy ideas, straight-talk, Blazing Saddles quotes, and outpouring of knowledge and wisdom, I thank Andy, my advisor. You showed me how to embrace the magical, wacky ideas and turn them into something productive and, most importantly, 'Viral'. I have learned how hard it is to make something a physically affective experience with digital materials but I have learned also, at least a little more, how to make it happen. For this thesis and much, much more, I am forever grateful.

I thank my thesis readers Matt Carroll and Cesar Hidalgo. Matt, thank you for your relentless support and your honest feedback during the many stages of the development of Me.TV. Cesar, thank you for your discerning comments and advice.

Thank you to all of Viral Communications for your friendship, advice, conversations, and solidarity. Savannah Niles, Travis Rich, and Thariq Shhipar: thank you for being willing soundboards and for sharing your experiences with me. To my office-mates and friends, Amir Lazarovich and Tomer Weller, who have sincerely helped me through tough times and wrestled the chaos of ideation with me more than they probably enjoyed, I am extremely grateful. Thank you, Margaret Yu, for your contributions as a UROP to the development of the Me.TV prototype and your joyful spirit of the office every Monday, Tuesday, and Wednesday. And of course, thank you, Deborah Widener, for your support and generosity.

Within the Media Lab, I thank my friends and classmates for all the friendship, laughter, and support. For helping me with this thesis more than your fair share, I am very grateful to Bianca Datta, Valerio Panzica La Manna, Juanita Devis, Ermal Dreshaj, and Sunny Jolly. And for helping me manage the panic of the thesis process, thank you, Linda Peterson and Keira Horowitz.

Thank you also to my friends outside of the Media Lab, and especially Narek Shougarian and Lukas Schrenk for also contributing to this thesis.

Lastly, I thank the people I hold dearest: my parents, my brother, and my partner, Sydney Do. For your constant love and support, I could never thank you enough.

# Contents

- 1 Introduction** **6**
- 2 Contribution** **8**
- 3 Background and Context** **8**
  - 3.1 Visual Programming Languages . . . . . 8
  - 3.2 Navigation Design . . . . . 21
- 4 Related Work** **23**
  - 4.1 Visual Programming Languages . . . . . 23
  - 4.2 Navigation Design . . . . . 25
- 5 Design and Implementation** **30**
  - 5.1 Summary . . . . . 30
  - 5.2 Language . . . . . 31
    - 5.2.1 Media Object Nodes . . . . . 32
    - 5.2.2 Events . . . . . 34
  - 5.3 System Framework . . . . . 38
    - 5.3.1 Server . . . . . 38
    - 5.3.2 Programming Environment . . . . . 42
    - 5.3.3 Video Player . . . . . 44
    - 5.3.4 Channel Navigator . . . . . 46
    - 5.3.5 Socket Messaging Structure . . . . . 47
    - 5.3.6 Users and Network . . . . . 48
- 6 Evaluation** **49**
- 7 Future Work** **51**
- 8 Survey Results** **52**
- 9 Bibliography** **58**

# 1 Introduction

From channel surfing to binge watching, the transformation of the visual media experience is unmistakable. We have gone from scheduled programming and rigid channels to a multitude of on-demand systems available on any device from nearly every provider, whether friends, corporations, or algorithms. The options have become seemingly infinite and the choice is all ours. At first glance, we are empowered and thrilled by the level of control. Then we are consumed by it. The choice becomes a part of the problem. This big bang moment of media, with all its new abilities and resources, has created a new set of challenges without fully resolving the question: "What do we want to watch?"

Whereas a human being scheduled our TV programming every day, we now have the option to choose our own at any instant. The caveat is that we either don't know what we want or we don't want to have to figure out what we want. We know we want entertaining, relevant content that suits our current mood and whims, but we also want to be surprised and discover new things like we did with television. The media experience as a whole is an organic process that pulsates with popular culture, our friends' interests, and our immediate yet transient state. This undefined, temporal personality is what cannot be captured with static preference settings, but perhaps can be captured with the programmability of media programming. This thesis presents Me.TV, a visual programming language and platform that retrieves media output based on user-defined inputs.

Me.TV bridges the gap between traditional television and on-demand viewing. Current media viewing takes place in one of three programming styles: traditional television, on-demand, and algorithmic media programming.

Traditional television is the concept of scheduled and pre-selected media content for broadcasted viewing. Television is curated by humans and is associated with simple and limited controls over content. These controls include channel selection by entering the channel number or browsing the channels through an up and down button. The constant browsing through channels, avoiding ads, and just searching for something new and interesting is a behavior we all know as channel surfing. On-demand viewing was the supposed remedy to channel surfing. It is the concept of nearly all content for instant viewing and includes a selection process by exact identification through text search and browsing by categorical menus such as genres, of which Netflix has reached in the thousands.

On-demand providers like Youtube, Netflix, and Hulu have proved to be incomplete solutions to traditional television's problems as we still cannot find something to watch. It is so hard, in fact, to decide on something to watch that comedians have made parodies of the endless browsing, thoughtless recommendations, and overly specific content genres [1]. The only problem in common between traditional television and on-demand viewing is the endless browsing, which was at least more efficient with television.

It seems that as we move from traditional television to on-demand viewing, we are sacrificing three of the greatest benefits of televised, scheduled programming. Firstly, broadcast television is inherently social. Viewers on the same channel see exactly the same programming, creating an experience that is shared remotely. We have some inkling of what our friends are watching and the time-sensitive nature provides fodder for day to day conversation. Secondly, broadcast television is minimal input. Controlling the output of the television monitor is a simple button click and allows us to leave it and forget it. Thirdly, television is now. Television feels live and content is tailored to be relevant to its audience.

Me.TV empowers one to program their own programming by taking advantage of external web services and browser-based viewing. In order to combine the elements of surprise, a sense of social input, and experiential control, Me.TV is a system design that includes a visual programming language, programming environment, custom video player, navigation system, and back-end architecture. Each of these components work together to allow a user to create, subscribe, and modify channels that grab content dynamically according to programmed rules. These rules are specific enough to cater to our on-demand preferences while retaining the television experience where we can set to a soft preference and see the world's media in addition to our own.

## 2 Contribution

*A visual programming language that allows media content to be retrieved, queued, and removed according to user-defined constraints and input variables*

In order to prototype this, we also provide a system that consists of a programming environment for this language, an engine to translate visual programming into executables, a video player for playback of created scripts in the form of channels, and a dashboard design for channel navigation.

## 3 Background and Context

### 3.1 Visual Programming Languages

Me.TV's visual language takes cues from many older visual programming languages. Here we offer some background on visual programming languages to illustrate some of the lessons learned applied to Me.TV as well as the classification of Me.TV within the field. Firstly, we answer what constitutes a visual programming language and the classification system applied to them. As we begin to understand the elements of a visual programming language, we form some of the requirements of Me.TV and design guides for accomplishing the goals of the Me.TV language. Secondly, we examine research that offers guidance on how to create the necessary language elements to achieve the type of language we want. Third and last, we summarize work in navigation design for media browsers as context for the Me.TV browsing platform.

The history of visual programming languages is relatively short, sitting at less than 50 years old and has not changed drastically. Some of the first visual programming languages are based on the flowchart, which, in computing was used mainly as a way to help assembly language programmers organize program structures and process flow, aiding in the ability to enforce structured programming and cleanly progress without getting lost in lower-level details. A sample flowchart is shown in Figure 1. Until the 1970s, however, the flowchart was not executable, leaving it digital but short of being called a visual programming language.

Robert Taylor's First Programming Language (FPL) software allowed users to generate flowcharts based on a given set of symbols and textual input, which the software translates into Pascual for execution. The elements of the flowchart as a visual programming language have persisted, joined by icons as technological strides were made in computer graphics, processing, and memory. As flowchart-like languages and



icon-based languages developed, a third type of visual programming language was formed - table and form-based systems.

These table and form-based systems are middle-of-the-spectrum languages that rely on graphic representations but do not consider these icons to be central to the language. They are often used to describe schemas and databases.

These three broad categories are the result of Nan C. Shu's concept of the profile of a visual language [2]. Shu's work helped organize visual programming languages and outlined much of the early history of the field. Still, debate goes on regarding

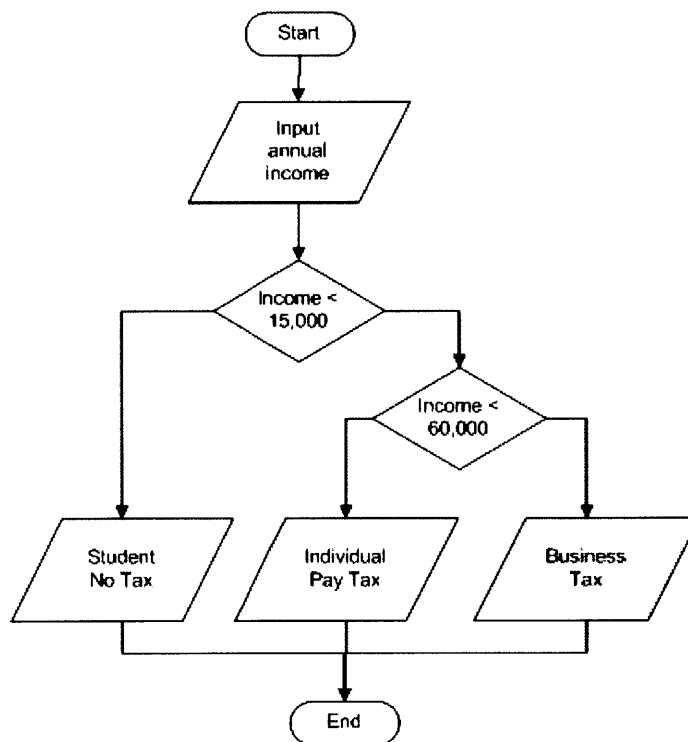


Figure 1: Example of flowchart, a diagram that represents process or flow through visual connections between structural elements. [3]

classification. Visual programming languages have harder and softer definitions depending on who you ask. Some languages are still considered visual languages although they are largely text based and rely only on a single spatial relationship. Regardless of the nuances to the definition, there are dimensions to a visual programming language that can be examined and explained. These are the three

dimensions of visual programming languages laid out by Shu in Figure 2:

1. Visual Extent
2. Language Level
3. Scope.

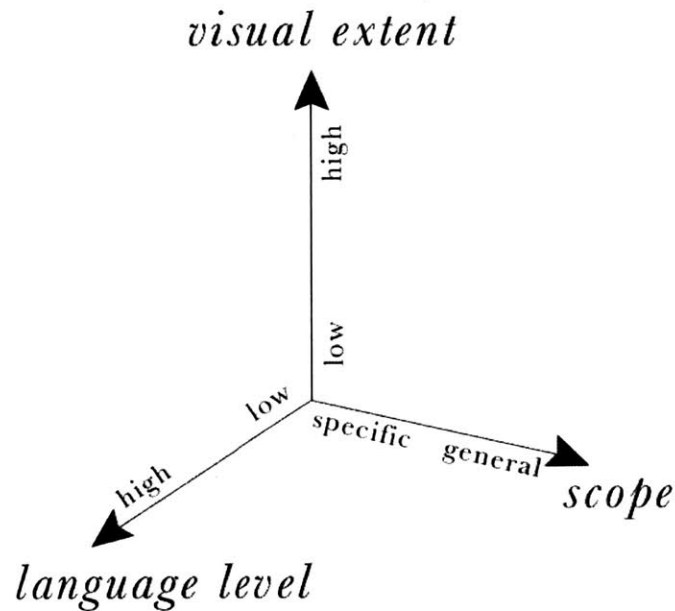


Figure 2: Shu's Three Dimensions of Visual Programming Languages

These dimensions help to classify the many visual programming languages out there. By looking at each dimension, we have the ability to discuss languages in more generic terms and evaluate them on the combination of dimensions a language possesses.

The first dimension is visual extent. For a language with a high visual extent, it will utilize visual expressions, which are 'meaning visual (non-textual) representations...used as language components to achieve the purpose of programming.' [2, pp. 139]

The second dimension is language level. Language level is determined by observing the complexity of code required to perform a task. When there are fewer elements involved or fewer lines of code required to complete a task, then the language is

considered to be a higher level language. Conversely, if a task requires many steps and lines of code, it is a low-level language.

Lastly, Shu defines the third dimension, 'scope', as a measure of language capability, rating languages that allow a user to create, control, or do more as a widely or generally scoped language. Languages that are specific to a domain or geared towards particular capabilities and ignore many others are considered narrowly or specifically scoped.

With these dimensions in mind, a review of some historically significant visual programming languages can be put into perspective with regards to MeTV.

### 1. Diagrammatic Languages

Diagrammatic languages include charts, graphs, and diagrams that are executable. Typically, symbols are used to represent processes and objects. Spatial and visual connections are made between symbols to denote a flow. The best known flow chart in visual language research is the Nassi-Shneiderman (N-S) diagram. N-S diagrams are easy to read and hard to introduce sneaky errors. By using space-efficient rectangular constructs with visually-obvious boundaries and logic the N-S diagram system became a favorite in the early attempts to make flowcharts executable. N-S diagrams are strict and clear with no possibility of arbitrary transfer of control. An example of a few N-S diagram symbols is illustrated in Figure 4.

One example of research expanding on the N-S diagram is the Programming-Support System. This 'pioneering work' by the IBM San Jose Laboratory is explained by Shu [2, pp. 157-166] where the N-S diagram system is used to create a system aimed at "

- (a) Establishing charting techniques to specify programs in a way that clearly shows their structure and logic;
- (b) Providing an interactive graphics system for the drawing and editing of these charts;
- (c) Providing a preprocessor/compiler mechanism to translate charts into executable code; and
- (d) Providing self-documentation as a by-product of the program development process." [2, pp. 61].

An example of the Programming-Support System is detailed in Figure 5. The system includes a graphical user interface and allows for both textual and pointer input (cursor). A user would use keyboard inputs to create a new NSD program (an extension of the N-S diagram system to include headers for programming) and symbolic constructs within the program such as 'IF', 'DO-LOOP', and 'CASE' blocks. Inserting text and other similar functions are accessible through the keyboard but a cursor is used in conjunction to allow for embedded code blocks within structures like loops. These rules of creation result in a diagram of textual elements that is written code but spatially arranged for improved flow understanding. [2, pp. 164]

Once the user has completed the script, a preprocessor is used to translate the created charts into PL/I (programming language developed by IBM) source programs for compiling by a regular PL/I compiler and then executed. An example program created with this system is detailed in Figure 6.

After the introduction of NSDs, the preprocessor was recognized as a drawback in one aspect of programming: debugging. When debugging, it is useful to have interactive execution. Programming with Interactive Graphical Support (PIGS) came out as an experimental system for the University of Hong Kong. PIGS implemented interactive support for testing and debugging at execution by making a number of changes, the most significant of which is the use of an interpreter rather than a compiler to allow the user to interact with the program during execution and make changes to the NSD program. Users can watch the execution of the program and follow the flow, allowing the user to visually understand what is going wrong when the code executes.

A popular, active, and highly influential development in the world of programming languages is the Smalltalk, created by Adele Goldberg, Dan Ingalls, and Alan Kay. Like PIGS, Smalltalk as a language and programming environment built in the responsiveness of the programming through the use of an interpreter which immediately showcased results of the code. This ability to inspect code line by line and its pure object-oriented nature made Smalltalk a powerful but also human-friendly language. Figure 3 shows Smalltalk example code, much of which is legible to non-programmers because of the use of objects which is closer to the real world and the English language. Programming environments for Smalltalk introduced concepts like inspection and the Model View Controller (MVC) user interface paradigm. Smalltalk's influence is still felt today and Me.TV can easily trace back some of its components to this pioneering work. [4]

```

Smalltalk-71 Programs

to T 'and' :y do 'y'
to F 'and' :y do F

to 'factorial' 0 is 1
to 'factorial' :n do 'n*factorial n-1'

to 'fact' :n do 'to 'fact' n do factorial n. ^ fact n'

to :e 'is-member-of' [] do F
to :e 'is-member-of' :group
do'if e = firstof group then T
    else e is-member-of rest of group'

to 'cons' :x :y is self
to 'hd' ('cons' :a :b) do 'a'
to 'hd' ('cons' :a :b) '<-' :c do 'a <- c'
to 'tl' ('cons' :a :b) do 'b'
to 'tl' ('cons' :a :b) '<-' :c do 'b <- c'

to :robot 'pickup' :block
do 'robot clear-top-of block.
    robot hand move-to block.
    robot hand lift block 50.
    to 'height-of' block do 50'

```

Figure 3: Smalltalk-71 Program

The ability to debug line by line through an interpreter became a necessary feature for Me.TV in order to support understanding of the content retrieval system. Instead of the typical interaction with the flow of the code, however, we focused on presenting the results of the code as the code was being built and allowing for the interaction to happen immediately, on the same screen. So while this provides the programmer feedback on the results of the programming, the readability of the visual language is still in question. To further improve the legibility of the programming language, we look to diagrams. Diagrams show the flow of data instead of focusing on the structure of the programming.

In order to describe the flow of data, Shu presents 'data flow programs' as a category of diagrammatic languages. Data flow programs are intuitive to read and combinable into larger programs. Each node, however, does not inherently exhibit self description. It is therefore necessary to augment them with other cues such as text or extended graphics. An example of a diagramming language that describes data flow rather than code structure is the state transition diagram.

NSDs, PIGS, and other similar languages are concerned first and foremost with the functions of the program being developed and the data upon which it operates. State transition diagrams, however, are a category of languages concerned with the human computer interaction and how the user understands

the programming. "One of the principle virtues of state transition diagram notations is that, by giving the transition rules for each state, they make explicit what the user can do at each point in a dialog, and what the effects will be." [5, pp. 51-59]

Me.TV implements code as icons, structures with visual elements, but its diagramming elements make it much more like a state transition diagram. Here, the necessity of that feature is because the user is programming for media data to be processed and queued in real time and is rather less concerned with the actual code structure. The actual code is largely abstracted away into form-based interactions.

Diagrammatic programming languages in use today with success include LabVIEW [6], Grasshopper 3D [7], and Node-Red [8]. Each of these examples are domain-specific, providing the visual elements most necessary and relevant to their respective domains. In particular, Grasshopper 3D is an example of a modeling language and environment within the category of computer-aided design (CAD) software solutions. It uses direct manipulation and icons to generate graphics.

The grand ancestor of Grasshopper 3D and other CAD-like software is, of course, the famous Sketchpad created by Ivan Edward Sutherland for his PhD thesis. Sketchpad was developed as a new mode of communication with the computer, utilizing a 'light pen' as the direct input and the ability to explicitly manipulate and store drawn symbols.

Me.TV uses the cursor as the means of allowing the user to drag and drop items wherever the user pleases. A program is drawn to how a user wants to see it, but employs the use of connectors like those in diagrams to denote the flow of code. In that way, the construction of the visuals is nearly entirely up to the user but it does not provide the same level of control that Sketchpad does, or its descendants aim to.

## **2. Iconic Languages**

Iconic languages are those that use icons and their spatial arrangement to instruct, rather than using these visual elements to do or show. To instruct is to use visual elements to create instructions to be executed. To do or show, also

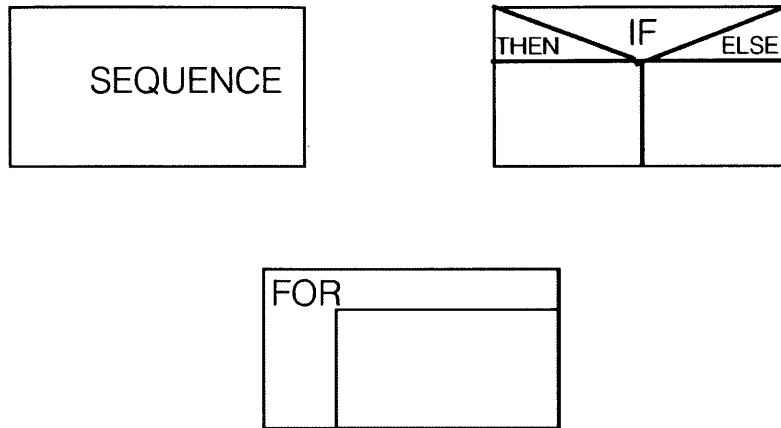


Figure 4: "NSD symbols for code structure"

known as 'Programming by Rehearsal' by Finzer and Gould, is to interact with visual elements to produce an outcome in rehearsal which can be later performed in the same manner. Shu considers only those iconic languages that instruct, or follow the 'do as a I say' constraint, to be true visual programming languages. Rehearsal languages are 'program development methodology' [2, pp. 195].

To illustrate some significant early work and the variety of visual elements and systems for icon-based languages, we examine VennLisp, Xerox Star, and PICT.

VennLisp had executable graphics based on the Lisp programming language. Users nest function calls and the parser creates a parsing tree based on the spatial enclosing relations among the visual objects. Figure 7 illustrates the VennLisp programming language. VennLisp was developed in order to further understand how humans use graphics to communicate, specifically how well spatial parsing facilitates understanding.

Xerox Star is an early example of user interface design utilizing metaphors to guide the design. Xerox's Star system applied the metaphor of the office environment to the digital environment, complete with files, drawers, printers, baskets, and slightly more abstract concepts of properties, options, windows and commands. Xerox's complete devotion to the design of a conceptual model of

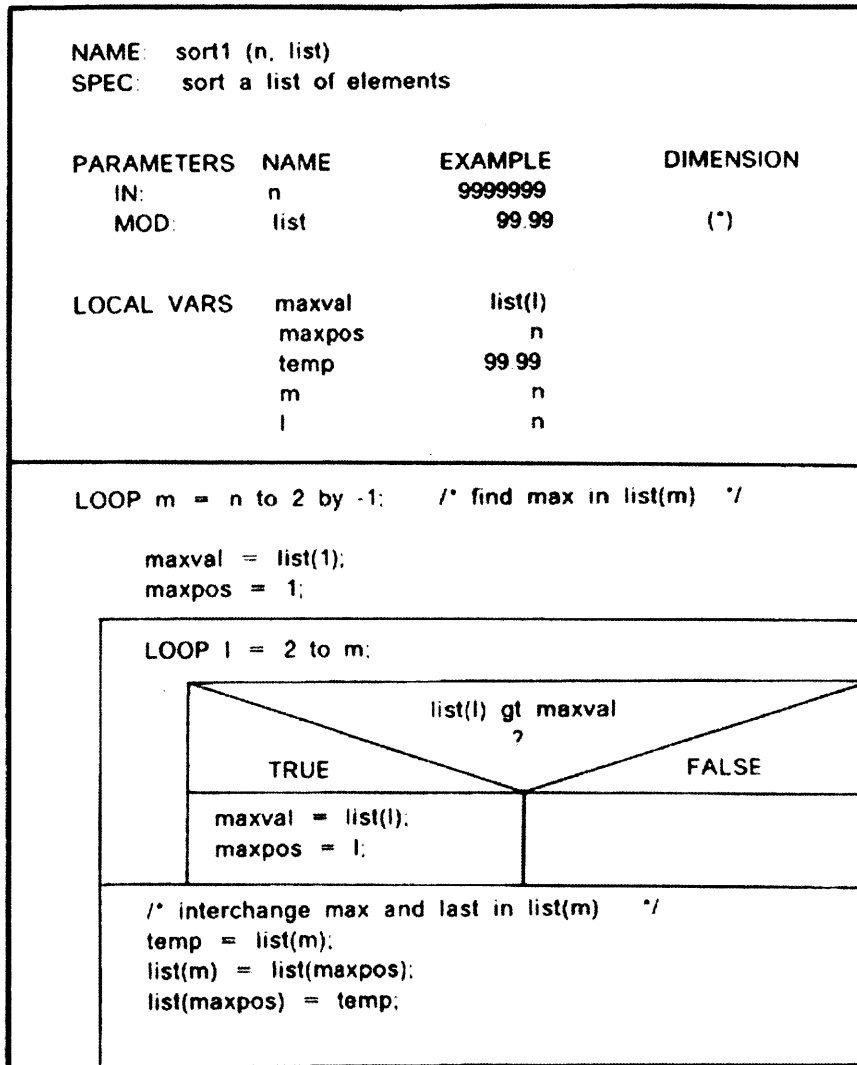
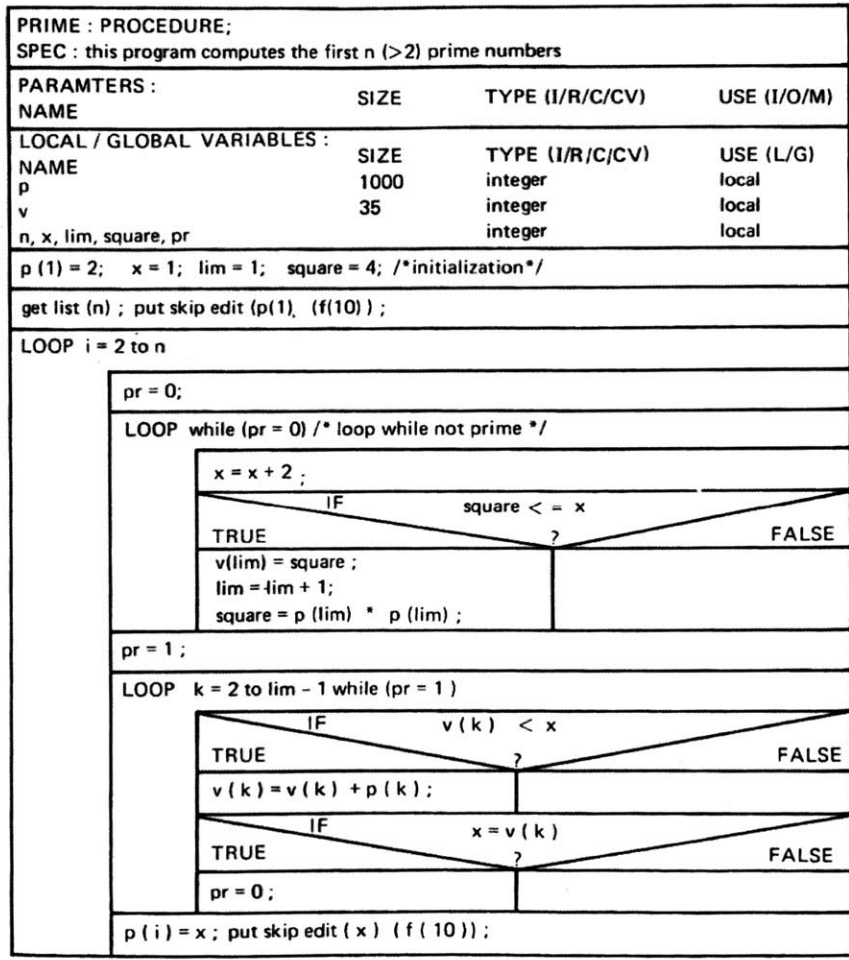


Figure 9-16. An example of an NSD program.

Figure 5: "An example of an NSD program [2]"

user interaction with the full system has been strongly influential in the computer industry with adoption of this metaphor easily recognized today. Figure 8 illustrates the Star system's desktop.





(a)

Figure 9-18. (a) An NSD program.  
 (b) PL/I procedure generated for the program in (a).  
 (Ng<sup>10</sup>. © 1979 IEEE. Original art courtesy of R. Williams.)

Figure 6: "An NSD program [2]"

Xerox Star's metaphor-based system was an intuitive design, where settings and applications were accessible through the mapping of the metaphor to the office. Me.TV is based on the idea of a television experience and so uses terminology and iconography from traditional television. The point of failure in using the metaphor is when we consider the abstract concept of a unit of media.

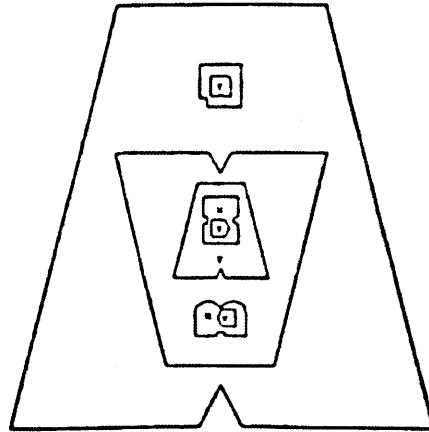


Figure 10-4. A VennLisp form.  
 (Lakin<sup>13</sup>. Reprinted by permission of Plenum Publishing Corp.)

Figure 7: "VennLisp recursive function [2, pp. 196]"

The terminology is slightly too specific with television and there does not exist a generic vocabulary for metadata excepting the obvious such as 'title' and 'publication date.'

The properties of media, to the layperson, are not commonly referred to but is increasingly so given the extensive use of digital media today. Events are even more abstract as a concept and disparate from the concept of television, but are similarly going to be more familiar to the average person, especially as the internet of things creates opportunities for event-based programming. For those items, we rely more on iconography and the use of an interpreter to demonstrate code at every new element or structure. PICT exemplifies a highly visual language wherein users draw their algorithms as a logically structured, multi-dimensional picture, connecting the available icons to create the desired flow of control. Because of the design of PICT's icons and the necessity of 'fit' between icons and their connectors in the editor, PICT has a narrow scope as the Xerox Star does. An example of the symbols within PICT is illustrated in the PICT library in Figure 9.

As Me.TV was of a slightly broader scope than PICT, the pure use of icons as code blocks without further interaction made the creation process too involved and unnecessarily complicated for Me.TV's purposes. Taking the result from one

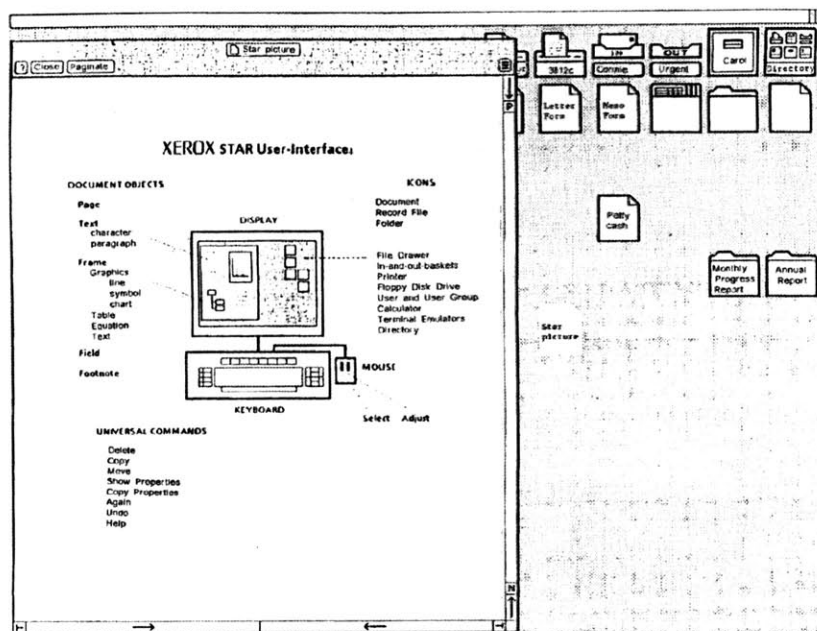


Figure 10 16. A desktop as it appears on the Star screen  
(Smith et al.<sup>4</sup> Reprinted with permission of Xerox Corporation.)

Figure 8: "A desktop as it appears on the Star screen [2, pp. 208]"

icon and passing it to the next connected one, however, is an important feature of Me.TV, used in the creation of event objects. Each of these and other iconic programming languages emphasize the need for clarity in the design of icons as well as a consistent and structured set of principles of operation and use. Implementation of an iconic language, if done well in designing the visual communication of icons, can significantly decrease the barrier to creating code for novices. The development of icons themselves as a field of study is also worth mentioning in the scope of understanding the background of visual programming languages. To be an effective communication device, the design of icons is of great importance. [2, pp. 206-212]

Icons can range from the representation to the arbitrary. When icons are representation, they are clearly understood when they represent objects we know and are metaphorical to the function of the icon. If the icon is abstract, we are symbolizing a function or concept that is familiar or comparable to what we know. When we do not have physical reality and common knowledge to rely upon, our

(Sub) program name	(Sub) system indicator	Help bulletin board/ data structure display
System menu / input keyboard		User program / Easel

Figure 10-22. Areas of display.

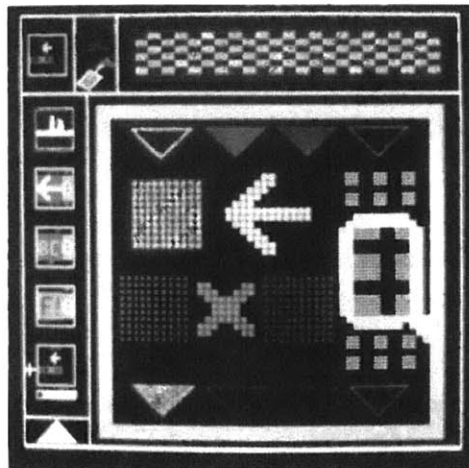


Figure 10-23. Program library.  
(Glinert and Tanimoto<sup>9</sup>. © 1984 IEEE, original art courtesy of E. Glinert.)

Figure 9: "PICT program library [2, pp. 216]"

icons become more arbitrary and function as signs. [9]

Once we've created an icon system, we then have to transform the images into language by further exploring two attributes of the icons. The first being the aspects of the icon which includes its bitmap, icon, and window names, its menu

tables, and its built-in functionality. The second is the relations of the icon which includes any points to parent icons, siblings, and child icons. Within an icon editor, the icons can become a high level language where the design of the editor can aid in the ability to craft coherent visual code. Me.TV has an icon set that is mostly abstract as the data and functions we symbolize are not perfectly represented by the physical world.

<u>Design</u>	<u>Function</u>
representational	picture
abstract	symbol
arbitrary	sign

Table 1: Lodding's classification of icons

Visual programming languages are heralded as technologies that will lower the barrier of entry to powerful computation and complex computer instruction but many have proved to be incoherent, illegible, or simply difficult to maintain as code can easily be updated but images may not.

### 3.2 Navigation Design

As we move towards a visual media experience governed less and less by a programmer, we reconsider the interfaces we are using and the level of control we actually have, either in the grand scheme or within our smaller circumstance. With our navigation still relatively limited to a simple remote control with regards to television, our navigation desires and media habits have outpaced our current interface. From this pain point middle-ground solutions have cropped up, such as Chromecast, Amazon Stick, and Apple TV to name a few.

Furthermore, as the interaction moves from changing channels to the asynchronous ability to watch anything on-demand, new approaches and evaluations of current media browsing systems and especially recommender systems have come to the forefront of discussion. Visual media, or television as we used to know it, has increased requirements of interaction. With greater personalization of our experiences through systems like TiVO and Netflix, we have allowed our preferences to be made explicit with only the expectation of our content in return, although recommendation systems can help but not especially hurt a system. [10]

Largely this has been a boon for marketers as greater troves of information on our preferences are handed over without bounds [11]. Mostly, however, these navigation interfaces are created for the lowest common denominator possible. They are ultra-simple and allow for basic personalization in the form of watchlists, playlists, and other crafted lists such as 'top trending' lists [11]. More complex media navigation systems have been explored in the form of 3D browsing [12], visual querying [13], and even adaptive user interfaces which employ machine learning to create automated changes in the user interface rather than increase the complexity of interaction [10].

## 4 Related Work

### 4.1 Visual Programming Languages

#### 1. Scratch

Scratch is a web-based graphical programming language, programming environment, and social network.

"Three core design principles for Scratch: Make it more tinkerable, more meaningful, and more social than other programming environments."  
[14, pp. 65]

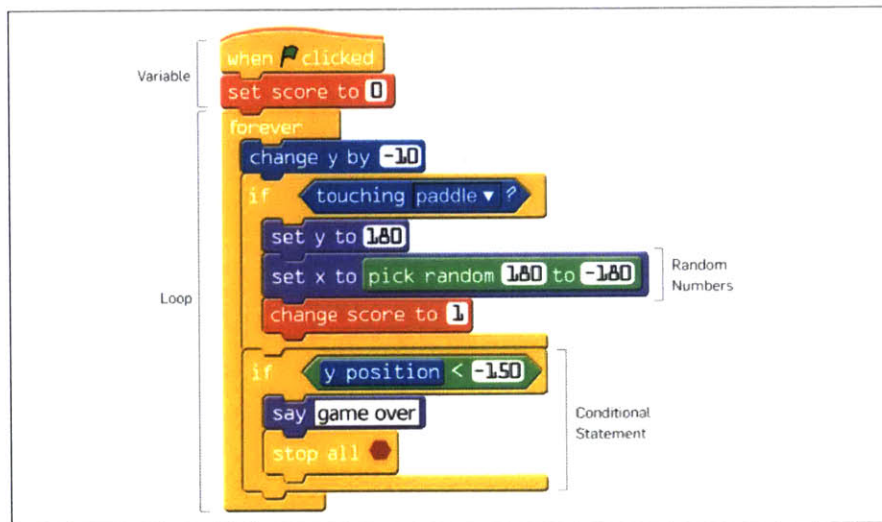


Figure 10: "Sample Scratch script (from Pong-like game) highlighting computational and mathematical concepts" [14, pp. 64]

The visual language of Scratch is based on simple blocks fitting together in only syntactically logical ways, like pieces of a puzzle, and the programming environment is similarly conducive to engaging novice programmers, allowing for 'tinkering' by considering spatial arrangements in a physically natural sense. For instance, code blocks of parallel positioning on the editor are run as parallel threads.

The blocks are created by Scratch's developers and designers to be as supportive of meaningful projects as possible. Scratch's designers "know that people learn best, and enjoy most, when working on personally meaningful

projects" [14, pp. 64]. In order to create a more meaningful programming environment, the Scratch designers prioritized embedding diversity and personalization into the programming. Because of that, Scratch supports many different types of programming projects including animations, stories, and games by providing a variety of tools. It even allows programmers to import personal media into their projects. Scratch's last design tenant is to be more social, engaging the community of users to collaborate and support each other. "For many Scratchers, the opportunity to put their projects in front of a large audience– and receive feedback and advice from other Scratchers– is strong motivation" [14, pp. 65].

## 2. IFTTT - If This, Then That

IFTTT, read like '-ift' in 'gift', is an automated task service that connects APIs (Application Programming Interfaces) between web services and products. For instance, the publication of a new NASA Astronomy Picture of the Day can trigger a Facebook [15] post linking to it without any input beyond establishing and enabling this task. IFTTT further simplifies this process by exhibiting previously established and popular recipes from other users and itself. To create a custom recipe requires only a few steps for the most part and greatly empowers the average user, especially in professional settings.

"The simple-but-useful concept has gained the company its largest-ever funding from Andreessen Horowitz and Norwest Venture Partners. With the additional cash, the company is looking to expand its service to the physical world by tapping into Internet-connected devices and the so-called Internet of Things. Future IFTTT software could be used to program lights or air conditioners to turn off in a home at a certain time of day, for example – indeed, some IFTTT users are already using Yo-based recipes to do just that." [16]

The simplicity of the interface, like Scratch, uses blocks to communicate 'buildability' and is a much more high level language, if that, than other visual programming languages. A sample of step one to programming an IFTTT recipe is shown in Figure 12. A completed recipe, ready to be turned on/off, is shown in Figure 11.





Figure 11: Sample recipe that sends an email whenever a NASA Astronomy Picture of the Day is posted

### 3. Recast

Recast is a recent thesis project from the Viral Communications group at the MIT Media Lab and inspiration for Me.TV. It is a platform for the average user to become their own broadcaster, cultivating snippets of the latest and greatest in trending news, by phrase, keyword, publication date, and source, to name a few filters. Figure 13 shows a keyword filter about to be applied to a content block, the representation of a command to fetch content.

Other specifications include the ability to set the length of the clip, see and select related items, as shown in Figure 15, and add voiceovers using the default microphone detected by the browser as shown in Figure 14. All of this happens within the same editor and can be tested immediately. Recast is a high level language of narrow scope but powerful enough to spin out a channel of the newest content in just a few minutes with more control and access than can be currently boasted with existing technologies for its purpose.

Within Figure 13, we also see that Recast has the ability to accept keywords, either by direct textual input or through a voice command using voice-to-text technology, simplifying the programming process further.

## 4.2 Navigation Design

### 1. Movie Map

The ability to discover content without venturing too far away from what is, in our minds, established preferences has caused a number of different systems to arise for cultivating, searching, featuring, and sampling unfamiliar media. A visually significant system, however, is more rare with both browser limitations and the unfamiliarity of spatially arranged titles as barriers. Movie-map [17] is an in-browser exploration tool where one defines a search title and a graph of related titles is generated and displayed. The graph arranges titles as clusters of

# ifthisthenthat

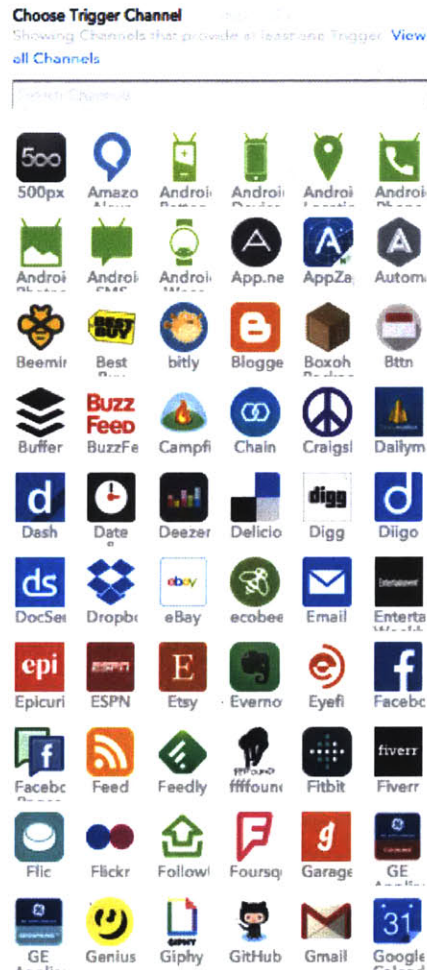


Figure 12: The *THIS* specification step of the IFTTT recipe building process

relatedness. A user can then click on a title and be taken to another map where the selected title becomes the center of the graph and other related titles are generated around it. In this way, a user is genuinely exploring by taking nearest paths and confining themselves less to establish genres. Figure 16 is a screen capture of the force-directed graph generated by the title *Pride and Prejudice*.

## 2. Netflix

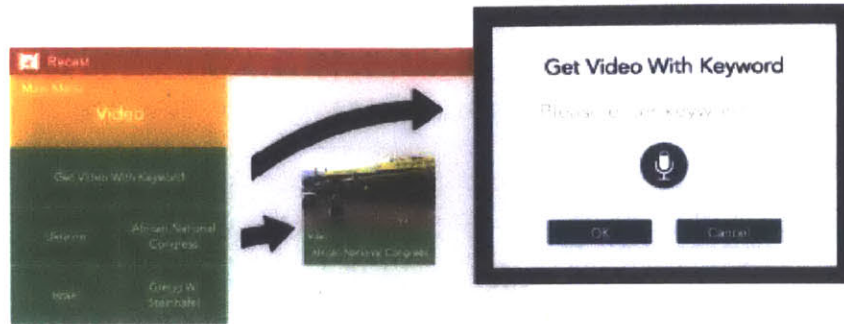


Figure 13: Recast video retrieval based on keywords

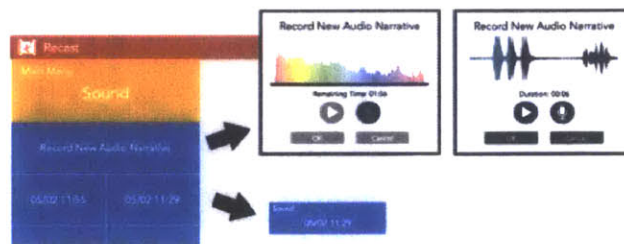


Figure 14: Recast audio recorder allows for voice-overs created while in the editor and tested immediately



Figure 15: Image previewer allows users to see image search results and select related images to include in their final product

Netflix has been said to be “crushing the competition”, forcing cable companies to offer more on-demand content, accelerating the demise of the movie rental business, and fundamentally changing the way media is consumed with new habits like binge-watching [18]. The ability of the customer to dictate to Netflix



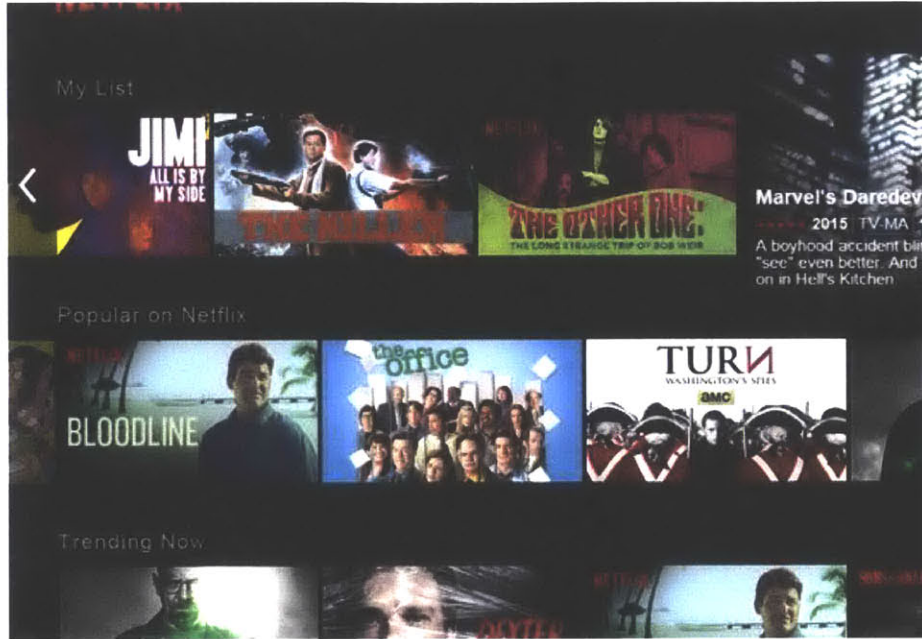


Figure 17: Netflix new UI image from USA Today [19]

aggregator and curator, Digg has the experience and power necessary to create playlists of the most popular or newest videos from all across the web. DiggTV precisely presents the best and newest and has simple controls to the content: back, forward, pause, explore (genre list), or exit. The video is always full screened until interactions cause the menu overlay and even transitions between videos with a television static video.

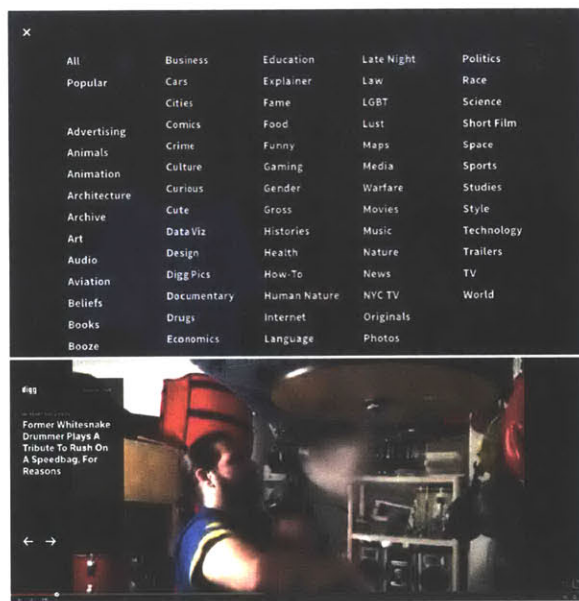


Figure 18: Screen capture of DiggTV genre menu and video player menu on hover

## 5 Design and Implementation

### 5.1 Summary

The overall design goals for the system are:

#### 1. To Empower The Average User to Program Their Own Media

The Me.TV system design is aimed at empowering the viewer to go beyond the boxes of genres and time schedules and create rules that can provide for how a person is at a given point in time. While we often prefer one genre over another, many of us feel more dictated by our mood than our previous viewing patterns. To not be at the mercy of an external programmer for a cultivated experience might mean a lot to someone who cannot stand horror or horror-themed films but wants to stay in on Halloween night.

Beyond such an occasional use case, we have probably all experienced a time when we simply needed utterly stupid and slapstick comedy to freshen us up before our nightly news watching. But after an hour of the doom-and-gloom of news, we want to turn up the dial back up on joy and light-heartedness. Me.TV's core design goal is to make these and other types of rules possible with minimal user input. We avoid the playlister scenario where users pick and choose each and every time while providing customized content lineups. A user should create a Me.TV script once and be able to run it forever.

#### 2. To Lean Closer to a Television-ish Experience

One of the major design concerns in feedback of older design iterations was over-involvement of the viewing experience. The experience of television where one turns on and flips forward or back has been under less review for change than the problem of content recommendation although it remains an almost immovable requirement. In fact, one of the few major experiments was the second screen. As a packaged product, it has been largely dismissed altogether by the industry as a failed experiment [23]. But it still exists in the wild, organically created when we experience lulls in the content we watch or when we are itching to share a thought on it.

The beauty of the organic second screen was that it was a welcome distraction from the first screen, fully controlled by the user and not simply an opportunity for more advertising. The manufactured second screen is the increase in involvement with the programming, or content, when one would rather let it go for the most part and just react to what we see or feel. Me.TV aims to provide the opportunity for reactions without requiring any actual interaction.

The systems working together should result in a visual media experience that is at once custom and continuous. The experience should not require constant intervention, nor should it rely on a third party to provide the content lineup. Instead, it should allow for both permanent and temporary inputs. These inputs could be from the viewer, the viewer's friends, or from interruptions like breaking news events.

In the design of Me.TV, the idea around inputs manifested itself in a custom video player, using dials whose connected actions and title are set by the channel creator. For instance, a channel can have an 'optimism' dial that when turned up will respond with more light-hearted news to play next. When the optimism dial is turned down, the channel might be programmed to respond with more news about war and death. The intensity and motion of the dial's needle is detected by the channel script and responds according to the user's programmed listeners. Figure 30 illustrates a dial within the video player.

### 3. **To Integrate A More Social Attitude Around Sharing Media Preferences**

Like the ability to recommend content to Facebook friends in Netflix, Me.TV aims to bridge the action of content viewing and sharing or recommending. Recommendations from friends, or word-of-mouth, is an established and researched marketing strategy. Research from 2010 by The Diffusion Group details how 90% of social network users consumed media because a friend shared it [24]. So it turns out that although we try to hyper-personalize media, our intake in fact does come significantly from friends and it seems to be something we want.

Instead of sharing individual content or viewing watched content lists of friends, Me.TV aims to share future experiences by making channel scripts entirely shareable, modifiable, and synchronous. In other words, channel scripts can be made public, taken from and by anyone, experienced simultaneously, and used either within another channel script or opened as copy of the original script to view and modify in the editor. In a sense, this interaction design is more aligned with the style of Youtube, where channels are playlists of videos that can be subscribed to for new content, and easily shared via social media.

## 5.2 Language

The goal with Me.TV was to create a language that was simple to adopt, legible, powerful, and scalable to support nearly any whim for a media experience. The language should be expandable so that although users now work purely with



event-driven visual code, in the future they should be able to create more complex code structure with controls like conditionals and loops. Working with an abstract concept like visual media metadata, however, proved to be less intuitive when creating cohesive icons and interactions. Therefore, Me.TV's visual programming language is a hybridized or middle-of-the-spectrum visual programming language.

The language uses diagrammatic grammar to describe flow and connectivity between code representations. Icons represent code structures and provide an access point to form-based interaction for greater specification of the structure, allowing for granular details to come into play in the programming process. This ability to scope up or down with details around code elements means greater scalability for future development of the language. While the fundamental rules remain consistent, the language maintains readability and can expand functionality without becoming an overwhelming set of iconography and diagrammatic grammar.

The built prototype of Me.TV contains two major structures that create complete rules. These are *media object nodes* and *event objects*. Each structure is represented by an icon and shape. Circles are used to denote items that serve as either data retrievers or data containers and so are the shape of media objects. Rectangular elements denote action and represent code structures that plan for events, making them the shape of our event objects.

### 5.2.1 Media Object Nodes

Media object nodes are central to the visual programming language and are step one of building a channel script. For any channel to contain content, the channel's script must provide the rule for fetching content to fill its queue. A media object node is that basic rule. It provides a series of parameters that make refinement of content fetching much like an advanced filter search. Instead of pure search using text and regular expressions, however, the scope of the visual language and iconographic style allows for the use of largely non-textual icons that can communicate soft parameters such as mood or emotion. This is illustrated in Figure 19, which shows a parameter selection interface that uses animated GIFs to communicate a concept while attaching a name to it.

The other parameters of the media object nodes are:

1. **Keywords**

Create a list of keywords, which can function as tags for inclusion or exclusion, as a part of the search for contents

2. **Genres & Mood**

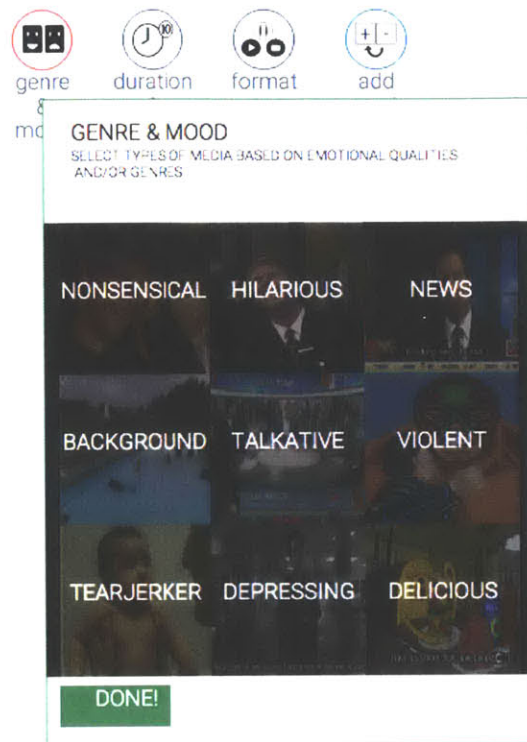


Figure 19: Using animated GIFs as icons in a selection interface for a media object parameter. This redundancy is aimed at reducing the possibility of miscommunication but also flattening the learning curve by reducing the need for strenuous interpretation of visual elements.

Add genres or moods to the search, which filters out content that does not fit the selected genres/moods

3. **Duration & Quantity**

Add restrictions to the length of the content retrieved, the number of items retrieved, or the amount of time to be filled by this media object's retrieved queue

4. **Single Source**

Add specific sources from which all content from this object node must be retrieved

5. **Format**

Add a specification on the type of content to be retrieved, such as 'movie', 'tv show', or 'music video'

## 6. Distribution

This is a toggle that tells our script reader to either 'add evenly', 'add front-loaded', or 'remove' the content node from the queue. Adding allows for specification of distribution within the queue during the viewing session. Media object nodes are defaulted to 'add evenly'.

Each of the media object parameters are represented as icons as illustrated in Figure 20. Each icon is set within a circle to remind the programmer of the nature of the icon. Below this enclosed icon is text. While the icon is inert, meaning it has not been clicked and therefore does not factor into the media object's retrieval of content, the text remains a label of the icon. If the icon has been acted upon, two signals help the programmer easily read what the media object has been specified to do. The first signal is a simple change of color of the enclosing circle's outline from black to red. The second signal is the changing of the text to reflect the parameter specifications. After



Figure 20: This is a media object freshly made without any specifications made yet.

each parameter is set, the visual code is interpreted and a request is made to the server to produce the resulting queue, creating a 'hot code'. The resulting queue is presented in the preview area as it would be presented if the channel script were executed as a whole in that current state. The necessity of this 'hot code' feature means that the visual code needs to be kept tracked of and updated after every interaction. To fully read the visual elements and create executable instructions to send to the back end would increase the time necessary to make updates to the previewer but is also more difficult to debug.

Therefore, we take advantage of the local storage available in the web browser to store data on the elements created and parameterized. Figure 21 shows the data structure in local storage. When a media object is updated, its slot in the local storage system is updated appropriately and a POST request is made to the server by passing the local storage slot as data. A POST request is a type of HTTP request that allows us to send data along with our HTTP request.

Similarly with events, we store their creation and specifications in local storage and make HTTP requests to the server after each interaction.

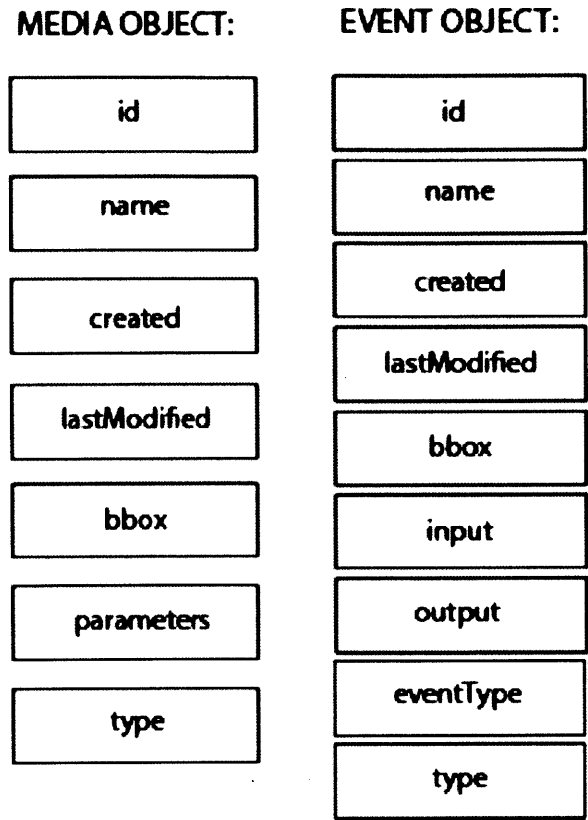


Figure 21: Local Storage Data Structure

### 5.2.2 Events

One of the most tightly packed functions within the language is the concept of events. Events are occurrences that trigger a message to be sent or broadcast. These messages often lead to further instructions, called event handlers. Like their name implies, event handlers are instructions for handling the occurrence of a specific event.

Events are well established systems and allow user interfaces to be interactive by employing event-driven programming, whereby a program's flow of execution is entirely determined by events. For example, a program establishes a mouse click as an event that then triggers a handler to create pink circles on the screen. In this program, a keyboard press might create blue squares next to every pink circle. It is not established that pink circles are drawn before blue squares but the events allow for this to be a possible outcome through a series of user interactions. Other outcomes may occur as well and be as valid.

Current examples of event-driven programming are everywhere online, most especially evident in javascript applications. Recently, however, greater conceptual examples of event-driven programming have made the ranks. The web service IFTTT and it's peers are event-driven systems where not just the interface is event-driven. The entire concept of IFTTT's service is creating event-driven programs that can be turned on and off, custom-made by users or pre-made.

For Me.TV, we take a page from IFTTT and pack functionality into extra simplistic interfaces so that any user can understand the event occurrence and set a handler. Each event is a categorical event, focusing more on the type of event than the specific API(s) it may employ. For example, a 'Breaking News' event might include listeners in the background that poll several news organizations' APIs for news alert but output a single event to the channel when any of the APIs returns a positive occurrence. The event object data structure is described in the following list of event components.

### 1. **Input**

Input is indicated by a 'TRIGGER' label and square handle acting like a child of the event object. The handle shape is meant to help the programmer determine what the event arm's purpose is and to what it can be connected. Clicking the square handle in the prototype's event set opens a dialogue box containing a form to further specify the event terms within the selected event.

### 2. **Output**

Output is indicated by a circle handle and 'OUTPUT' label, which communicates the need to connect a media object node to the handle. If the output to the queue should be the content from the event, the circular handle can be dragged into the event object's input handle and connected, signaled by a darkening of the connected media object's outline. In that case, the ID saved for output will be that of the event ID.

### 3. **Event type**

Event type is the event object itself. Like IFTTT, the programming is simplified by simply offering the selection in full as icons. Upon opening the event menu, event type is chosen based on the category of event being sought. For example, a schedule-based event can be based on the actual time or a calendar event. Because both of these events fall into a 'scheduling' category, they are only specified as one or the other with more detailed terms in the input dialogue box.

The event objects created in this prototype are shown Figure 22. At the top-left, we have created a user input event with an icon of a user with a lightning bolt. This event creates a dial with minimum and maximum values specified in the TRIGGER dialogue box. This dial is added to the user interface of the video player when the channel is

played. Upon moving the dial needle, the output of the event either retrieves more content based on the connected media object to the output handle or removes items from the queue that match the connected media object. A fully connected event object with a set TRIGGER is shown in Figure 23.

Secondly we have the event object with the newspaper icon in the top-right corner. This event object is triggered when a breaking news event occurs. Further specification of the news event allows certain categorical news to be the focus rather than any news alert. When the specified news event occurs, the connected output media object's retrieved results are added to the channel's queue.

Thirdly, we have the schedule object with the clock icon in the bottom-left corner. The scheduler event object allows a programmer to use a schedule event to decide when to add or remove content to the queue. A schedule event is determined in the TRIGGER dialogue box to either be an exact hour and minute of the day or when a type of event rolls around on a connected Google calendar.

Lastly, the bottom-right corner shows the Google Fit icon. This event is based on a connected Google Fit account and the step-count goals specified in the TRIGGER dialogue box. Like the other event objects, reaching the terms of the specifications in the TRIGGER, the output is the addition or removal of content from the connected media object.

## **5.3 System Framework**

### **5.3.1 Server**

The server is what processes all requests from the front-end, serves the content as a list of urls to be played in the video player, translates the visual programming into executable javascript, and manages the users and their script sharing. All communication between the front-end interface and the back-end code is done through the Application Program Interface (API). All data is stored in the MongoDB database, chosen for the ease of use and variability of data structures. A high level view of the architecture for the video player is shown in Figure 24. Figure 25 shows the relevant architecture when the user is engaged in the programming environment rather than the video player. Within the architecture, the major API endpoints provide the access to necessary back-end scripts and data. These API endpoints are described here:

#### **1. Back-end Scripts**

The back-end scripts encompass all the code that requests, processes, and returns data through the use of events and sockets. These scripts are the

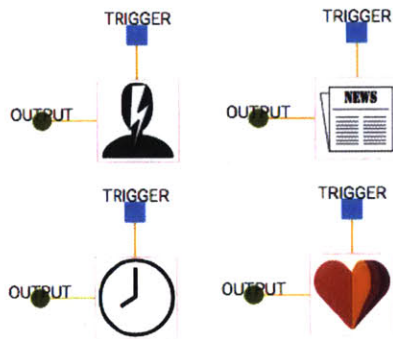


Figure 22: The square denotes that the object belongs to the 'event' class. Events are selected based on the API they connect to and the input arm interfaces with the user as a selection box upon clicking the handle. The selection box allows specification of the exact event through use of the API. The circular handle of the output arm indicates it should be connected to a content node.

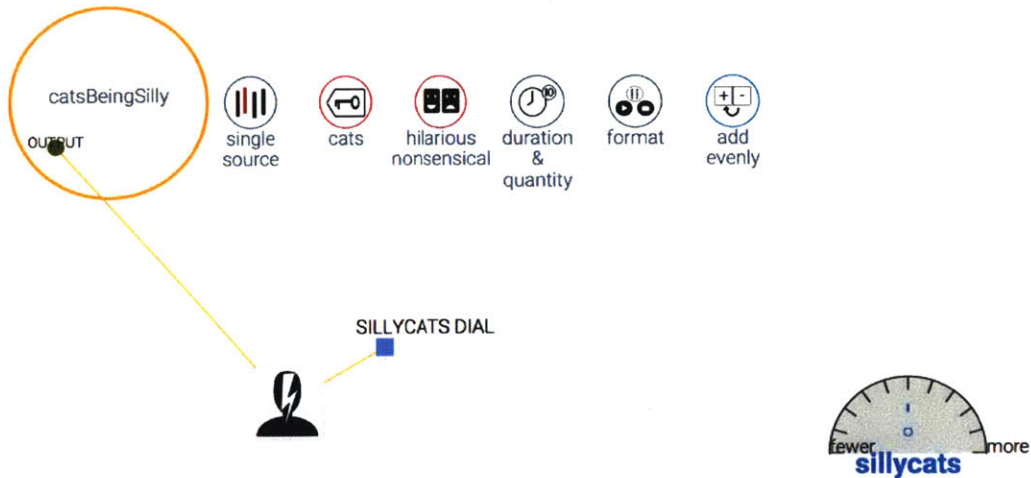


Figure 23: Fully connected event object with the resulting dial.

connection to the database and external APIs. The front-end makes all requests for data or database changes by emitting these requests either through socket

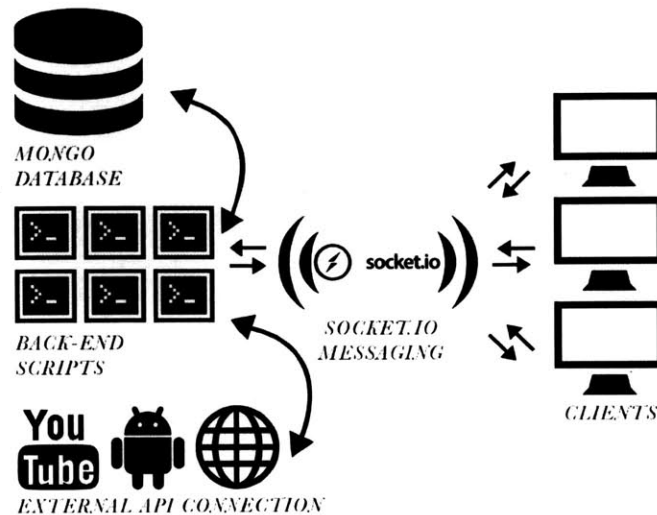


Figure 24: The back-end consists of the Mongo database and the scripts that process data from the database as well as external APIs. Icons courtesy of Icomoon [25] under GNU License.

messages using Socket.io [26] or HTTP requests where the server then distributes the necessary tasks to back-end scripts and returns the requested information. The different endpoints of the API are explained below.

(a) **New Channel Script**

Creating a new script in the front-end triggers a request to the back-end to create a slot in the database for the new script, saving certain details like the date of creation and the user's id. This API endpoint returns the id of the script so that further requests to update the database can be made efficiently.

(b) **Load Channel Script**

When a script is saved and is opened to be edited, the request to the back-end is to load a script based on the given script id and user id. The details of the script are passed to the front-end for processing and rendering. The processing and rendering essentially programmatically create the visual elements with all the relevant events attached so that the old elements are editable again. The previewer is populated with the appropriate



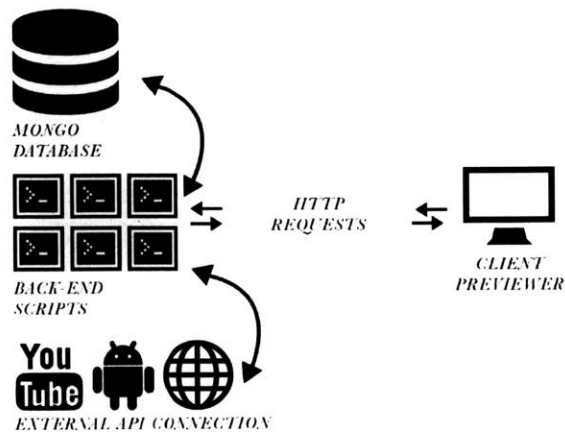


Figure 25: This overview shows the difference between the video player process and the programming environment process. The programming environment relies on HTTP requests between a single client's previewer in the scripting environment. Icons courtesy of Icomoon [25] under GNU License.

elements like the dials and the queue preview. The queue is created by sending a hot code request, as described next.

(c) **Hot Code Request**

When creating and editing a script, the programming environment's preview area is populated with a queue of content. Whenever a new code structure is completed, such as a parameter specification, a hot code request is triggered, sending a POST request to the back-end with the data structure of the media object. In return, a list of content metadata, that includes the ID of the data structure that caused it into being, is used to create the preview queue.

(d) **Run Channel Script**

While a hot code request gets real and real-time data, running a channel script requires more of the back-end and so this becomes a different process. Here, the script is essentially compiled by the back-end, sockets are initiated, and the data is sent continuously to create a seamless experience. The data structure sent to the video player is also slightly different in that it must include the event object's data when the event object includes user input. This defines a portion of the video player's user interface.

Furthermore, the queue of content itself can include more metadata than the hot code request based on events. Rather than keeping track of individual content's relationship to the media objects, keywords and other metadata are directly included and events that may remove items of certain matching keywords is run directly in the front-end.

The major components of the architecture as a whole are described here:

## 2. Database

Because metadata on media varies greatly between sources and our system works purely in javascript, MongoDB was the chosen database since MongoDB is scalable and stores documents in a JSON-like structure. Me.TV's database structure is outlined in Figure 2.

<b>Users</b>	<b>Channels</b>	<b>UserMediaHistory</b>
<i>holds data about users</i>	<i>holds data about channels and their scripts</i>	<i>holds data on media that has been viewed and by whom</i>
id	id	id
name	name	name
date_joined	creation_date	creation_date
last_login_date	last_access_date	last_modified_date
<b>extra_information</b>	source_author_id	<b>users_list</b>
<b>profile</b>	current_author	<b>media_metadata</b>
<b>friends_id_list</b>	<b>script</b>	

Table 2: The core three database collections are detailed here by column. Within each, the bolded field name indicates that the field contains a nested data structure

## 3. External APIs

In order for many of the events to work, such as the 'breaking news' event, external APIs have to be employed. In particular, this iteration of Me.TV uses the Youtube V3 API and New York Times API for fetching content. The reasoning behind the choice of these two APIs as demonstrations of media is:

### (a) Accessibility

The API is not overly restrictive with access and call limits

### (b) Documentation

The API is well documented and widely used by a community to ensure ease of debugging and availability of wrapper libraries

- (c) **Content** The API provides less restrictive access to actual content
- (d) **Metadata** The API provides some minimal amount of metadata with the content
- (e) **Variable Use** The API allows for demonstration of non-traditional causes of change in media programming

As current APIs for any broadcast television is not available at the time of this prototype, media selection based on actual broadcast television was eliminated from the prototype.

### 5.3.2 Programming Environment

The programming environment incorporates a toolbar, a preview section, and the editor where the programming occurs.

#### 1. The Toolbar

Within programming environments, a helpful feature, as demonstrated by successful visual programming systems like Scratch and Grasshopper3D [7], is a defined set of tools by which one can refer to frequently and immediately [27] but also always be able to see in totality as a helpful way to know the system. The toolbar consists of an icon for the event object menu but in future will include other major categories of programming structures.

The toolbar presents each option as a drop-down menu of the available constructs. Selection of the event construct closes the menu and adds the graphical construct to the editor for dragging, dropping, and interaction. This drop-down is shown in Figure 26.

#### 2. The Editor

The editor is the defined area in which the programmer creates code. Me.TV's editor is a blank space that when clicked produces a content object node as shown in Figure 20.

In the design of this interaction, a frequently voiced problem was the inability for novices in such an environment to know where to begin without a lot of guidance. Given a toolbar, many asked for a default action to ease them into the process. Therefore, the first interaction defaults to the creation of the most important aspect of the programming language, the one action that is required for



Figure 26: The event drop-down menu

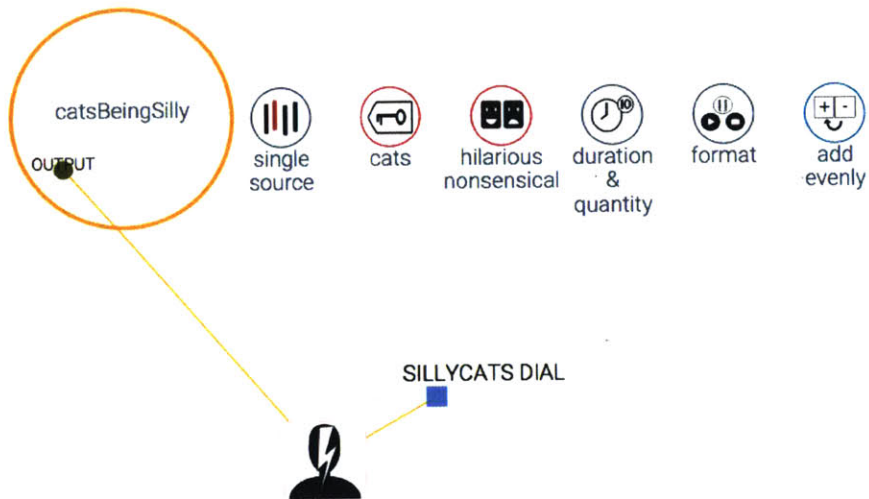


Figure 27: A new event is created by using a simple drop-down menu with connectors ready to be dragged to their destination



Figure 28: The Me.TV programming environment, the editor.

any channel to be created: a media object node. Interaction with the media object produces immediate results in the preview area which further aid in understanding the programming being created.

### 3. The Preview

The preview area provides immediate feedback to the programmer on the type of results that can be expected from their most recently code. When a media object node's parameter is specified, the preview area updates with the type of content being fetched by the content object node given the parameters specified. The preview is a peek into the channel experience and includes the ability to interact with the video player's user inputs as they are created and defined in addition to seeing the type of content queued by the script.

The biggest caveat with the preview feature, however, is that unlike other hot code features in programming environments where dynamic media is not involved, this one boasts less accuracy because the content is fetched in real time at any time the script is executed, fundamentally changing the type of results that are retrieved from the server.

### 5.3.3 Video Player

The video player incorporates three main elements to create the custom experience of Me.TV.

#### 1. Video Viewing Area

This is essentially a basic HTML5 video player, modified to work with Me.TV channel scripts, using default controls like play, pause, volume, skip, and back. When content is not of a video format that can be played back by the HTML5 video player, an iframe is created for the user to access the content. Video loading in a web video player introduces lag into the viewing experience and so is as seamless

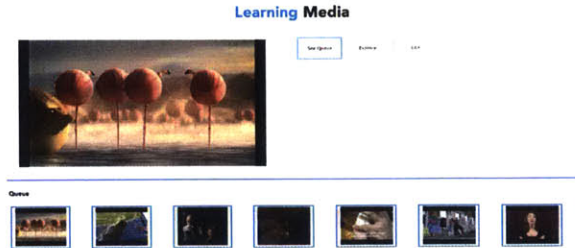


Figure 29: The Me.TV video player interface with queue showing and controls on the right.

as television, however, videos can be preloaded to a certain extent to minimize the lag time between fetching video and playing it.

## 2. Queue

The queue is an area that contains a list of the content generated by the channel script, display as images with text overlay. The queue appears whenever there is interaction with the entire video player environment. These interaction cues are events from the mouse and keyboard. The queue closes when the queue button is toggled.

When a channel is chosen for immediate playing by the user, the channel script is executed to obtain a JSON list of content metadata, called the media queue, from the server. The queue of content is a list of content metadata where the URL is rendered either within the appropriate video player or as an iframe, depending on the video source. A rough sample outline of the media queue structure is shown in table 3.

The channel scripts listen for events either from the server or the user interface (the video player). Depending on the programming of the channel, the channel script will change the queue in different ways. As certain events may be filtering the queue in real time, fields like 'keywords' and 'categories' may be included in the metadata of each item within the media queue. Therefore, not all items and not all queues have the exact information but a basic few fields of metadata are required.

These required fields are bolded in table 3. At present, the only media format used here is video, but can be extended in future to include other types of media for a fluid, boundary-free experience more in line with how we actually access information within our browsers.

Each media queue's item metadata is stored in the user's history upon viewing and the data field 'last\_access\_date' is updated to the current time of storage in history.

<b>Field</b>	<b>Description</b>
<b>id</b>	Unique identifier that is used in saving the content to history.
<b>name</b>	Title given by data source.
<b>description</b>	Description given by data source.
<b>duration</b>	Duration is the time length of content in seconds and may or may not be available, but is usually given by data source.
<b>source</b>	Source field is given by data source and is the URL at which the content can be fetched.
<b>format</b>	Format is, currently, either a web video format like MP4 or set to 'page' to indicate necessity of iframe.
<b>publish_date</b>	Publication date given by data source.
<b>last_access_date</b>	Last date content was viewed by user. Field is updated when content gets loaded into player.
<b>keywords</b>	Keywords field is available when script dictates an event based on the value of this field.
<b>interaction_data</b>	Interaction Data is an optional field that allows behavioral data on user to be associated with content.

Table 3: Rough outline of media queue, showing the essential fields in bold

### 3. User Input Area

The user input area is a small area that contains the dials specified by the channel script for user interaction. Any change in the dial's needle setting creates an update to the queue that is visible while the user input area is also visible. The dials' titles and end values are entirely defined by the user within the channel script. Dials are direct manipulation images. See Figure 30 for the prototyped player's user input area.

#### 5.3.4 Channel Navigator

The channel navigator is a dashboard where the programming environment is accessible as well as the video player. Most importantly, however, the channel navigator presents a data-driven navigation system for channel scripts between users. By using a data visualization that, like the movie-map [17], allows for exploration through links as data objects, we simplify exploration from an intentional process with many decisions to a more serendipitous one that acts more like a rabbit hole.

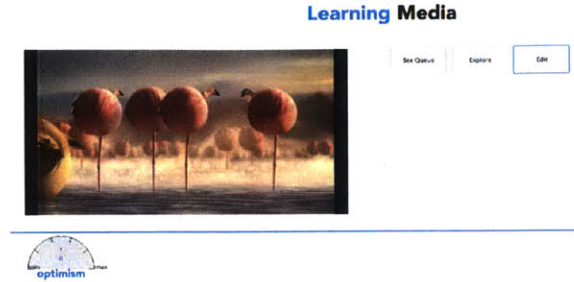


Figure 30: Video player with the user interface activated

By dragging and dropping channel nodes into one's own channels area, the channel script becomes a reference to the author's channel. In order to copy the channel script and save it in the state it's currently in without no updating from the original author, the channel node is selected after the drag and drop. The selection causes a small menu to appear with the option to either edit the script, create a full copy, play now, or delete. This is illustrated in Figure 31.

1. Choosing to edit the script opens the channel script in the editor and is saved in that state as well as all changes as a new channel script in place of the old dragged and dropped channel node.
2. Choosing to create a full copy simply copies over the channel script associated with the node as it currently is into one's channel node set. The copy is therefore detached from the origin author and will not update as the original author updates the script.
3. Choosing to play now will run the channel script, load the video player immediately, and begin playing videos within the generated queue.
4. Choosing to delete simply removes the channel node from one's set.

### 5.3.5 Socket Messaging Structure

In order to make speedy and frequent requests to the server and serve out data as it comes in according to the events structure, Me.TV relies on a socket communication system when channel scripts are executed. As a user logs into a system, the front-end connects to the server through the socket when a script is played, and retrieves the necessary data to produce the user's video player or dashboard depending on which page is being requested. If the script includes event listeners, the client will have





## 6 Evaluation

In evaluating the Me.TV prototype, a small-scale, anonymous survey was conducted to understand the legibility of the visual language, the level of detail required in the tutorial to get an understanding of the visual language elements, and the satisfaction with the type of media fetched in a given example.

The survey consisted of an introduction to Me.TV, a tutorial on the different elements of the Me.TV language, and then followed with eight questions. The tutorial only described Me.TV as a visual language for crafting unique media experiences. It did not divulge information on other features of Me.TV, such as how the channels are populated with media at each execution or that scripts are shareable. This maintains strict focus on the language itself.

The first two questions asked respondents to write their interpretation of a given Me.TV script and then rate the ambiguity of each of the main components. Respondents understood the media objects very well but were slightly confused by what to expect from the event given. The event, in the survey, was a user input event. The rating for the event object in terms of ambiguity on its function and output was a strong 'Medium Ambiguity', indicating some unclear components but overall not entirely ungraspable. This result indicates that event objects and dials will require more well designed tutorials or coding preview features that explain through experience.

The next question looks to determine the overall opinion of the visual mix - iconic and diagrammatic. The icons were rated to be 'Very Helpful' in aiding respondents' understanding of the code, but the use of connectors to create a diagram of data flow left a quarter of the correspondents less impressed. This indicates that either stronger structural rules should be in play to help communicate the data flow or they should be done away with in favor of icons and spatial arrangements as with the media objects.

The next question asked for a rating of the media object's fetched content compared to known media providers' systems. A little under three quarters of the respondents liked the results of the media object more than both traditional television and current on-demand systems like Netflix and Hulu. However, when it was compared to the act of creating a playlist on youtube, it was rated to have a similar level of satisfaction. This indicates that the level of engagement required in creating the Me.TV script felt equal to that of creating a playlist on Youtube. Since respondents had no idea that this channel script fetched new content at each execution, this may have felt like as much work as picking and choosing content to queue up on Youtube.

Then, to understand the respondents a little more, we asked about their experience with programming, visual languages, and the different media systems (traditional television and on-demand). There was a nearly even split of expert programmers, intermediate programmers, and novice/non programmers. This meant our results' trends are not strongly influenced by the background the respondent.

In determining the amount of experience respondents had with visual languages, including the ubiquitous flowchart, all had at least some experience. But once again, the respondents were nearly evenly split amongst the expert, intermediate, and novice categories.

The last questions tried to determine familiarity and habits around media from traditional television and on-demand. On-demand was a much more familiar concept to respondents with most having used at least one provider extensively. Traditional television was a familiar concept to all but use was infrequent for most respondents. This suggests that this sampling of the population was much more open to working towards a media experience and making decisions to have one or the other.

These survey results inform us that future work is necessary in improving the event object representation and interaction to limit the misunderstandings about their functionality and output. A redesign of the communication of data flow may also be necessary.

## 7 Future Work

Beyond the indications of work from the survey results, future work with Me.TV includes a more expansive channel navigation system, control constructs for the language, more event objects, and an improved video player.

1. **Expanding the channel navigation system** Currently, the channel navigation system sets up a bubble visualization of channels. To properly explore channels based on attributes of the code, a more complex visualization is required or more options for interactivity with the bubble chart is required. For instance, a user may want to copy over all channels related to their favorite 'short-horror-long-comedies' channel. Currently, there also is not a hookup within the navigation system to find your friends explicitly and always have their channels available nearby to explore.
2. **Adding control constructs within the language** During the design and prototyping of Me.TV, the question of function in the scope of media retrieval for programming constructs like for-loops and conditionals were difficult to answer. Any future work should explore the possibility of additional programming constructs to push the boundaries of how we understand a media experience and how we might consider crafting one. At present, an event-based system satisfies the obvious desires.
3. **Increase the number of event objects** In addition to increasing the number of event objects, Me.TV must also be able to accept event objects developed by users. The organization and process for such a system will need to be designed with scale in mind. Questions of how many events will be overwhelming and how users can add to existing events to maintain the sense of categorical functionality of an event object have yet to be answered.
4. **Improved video player** The ability to start off at a full-screen, explore and edit without obscuring the current picture, and play items other than video are all improvements in the future of Me.TV. The playback experience could be enhanced even further by offering non-screen controls, taking advantage of keyboards and connected smartphones and smartwatches.

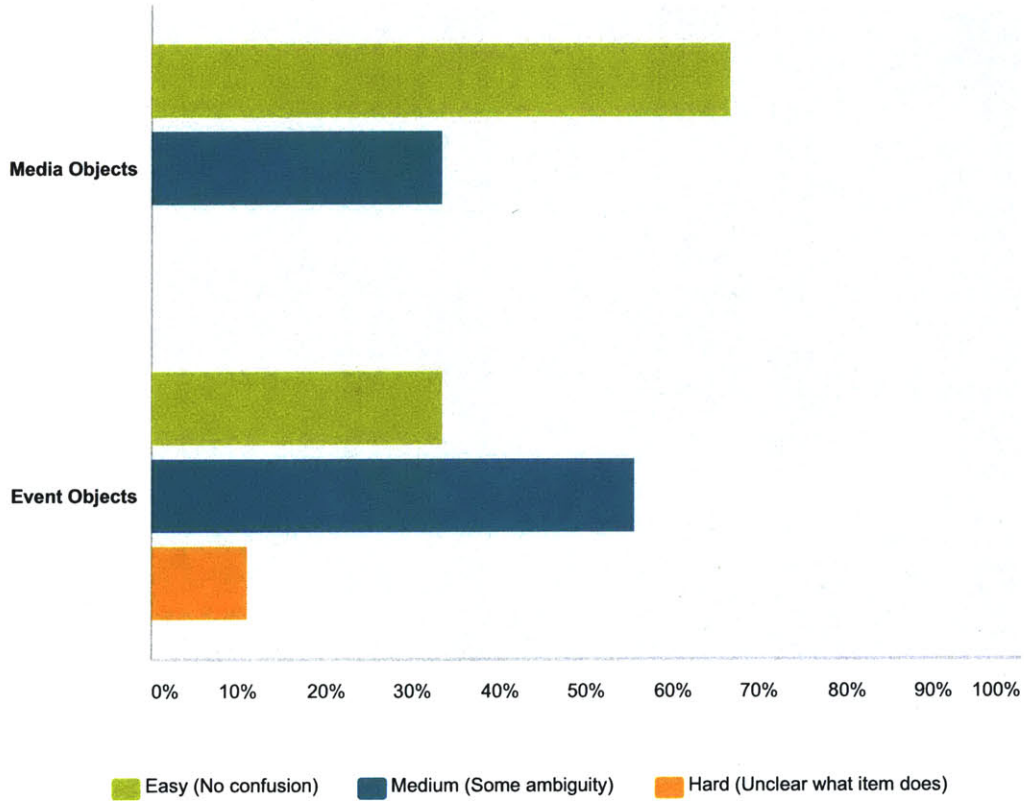
## 8 Survey Results

### Q1 Write your understanding of what the code is requesting here:

Answered: 9 Skipped: 0

### Q2 How difficult was it to understand the script's components?

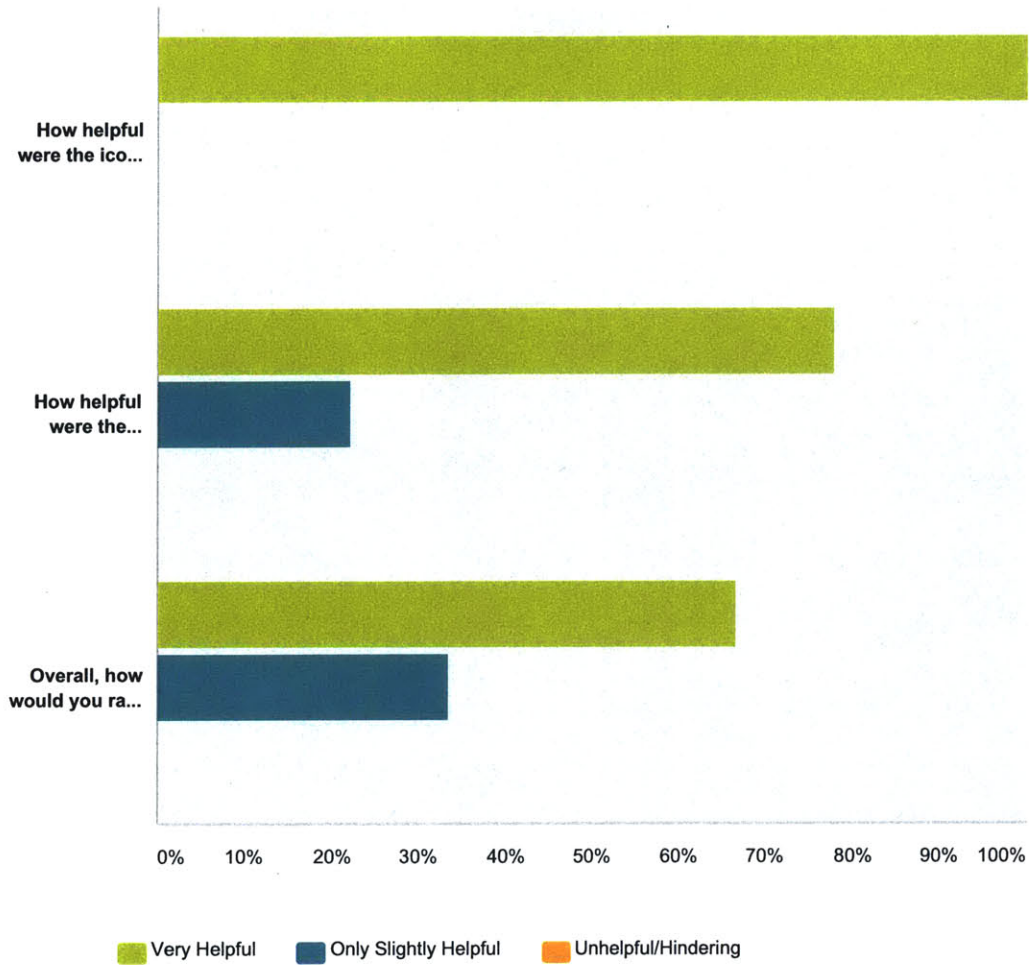
Answered: 9 Skipped: 0



	Easy (No confusion)	Medium (Some ambiguity)	Hard (Unclear what item does)	Total
Media Objects	66.67% 6	33.33% 3	0.00% 0	9
Event Objects	33.33% 3	55.56% 5	11.11% 1	9

### Q3 How helpful were the icons/diagram elements?

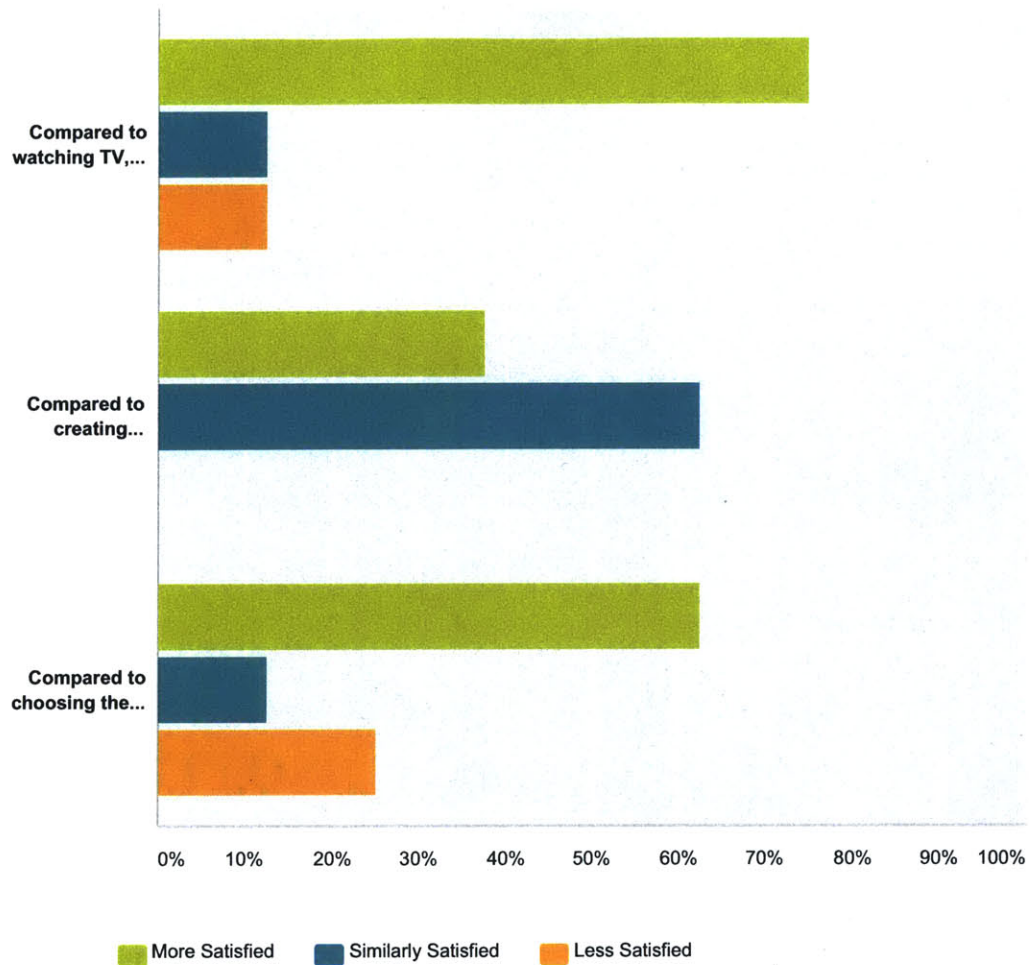
Answered: 9 Skipped: 0



	Very Helpful	Only Slightly Helpful	Unhelpful/Hindering	Total
How helpful were the icons in aiding your understanding of the code?	100.00% 9	0.00% 0	0.00% 0	9
How helpful were the connecting lines between the event object and media object in aiding your understanding of the code?	77.78% 7	22.22% 2	0.00% 0	9
Overall, how would you rate the visual elements in terms of helpfulness in understanding the code?	66.67% 6	33.33% 3	0.00% 0	9

**Q4 Rate the resulting channel queue as someone with interest in cats, heavy metal, and non-terror news:**

Answered: 8 Skipped: 1

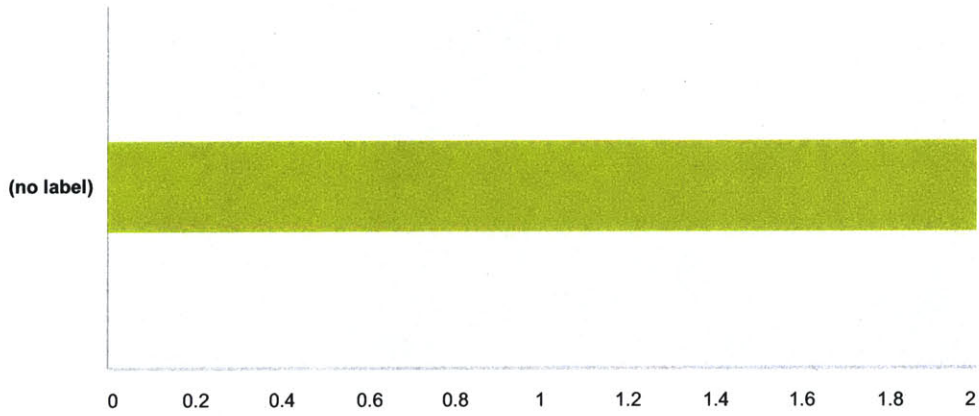


	More Satisfied	Similarly Satisfied	Less Satisfied	Total
Compared to watching TV, this queue would make me:	75.00% 6	12.50% 1	12.50% 1	8
Compared to creating playlists on Youtube, this queue would make me:	37.50% 3	62.50% 5	0.00% 0	8
Compared to choosing these types of items from on-demand providers like Netflix and Hulu, this queue would make me:	62.50% 5	12.50% 1	25.00% 2	8

### Q5 How familiar with programming (in any computer programming language) are you?

Answered: 8 Skipped: 1

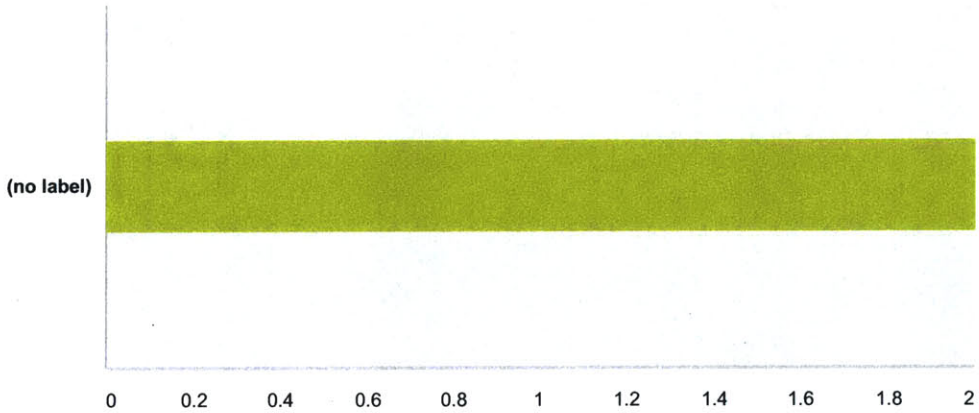




	Very Familiar	Some Experience But Not Expert	Very Novice/ No Experience	Total	Weighted Average
(no label)	37.50% 3	25.00% 2	37.50% 3	8	2.00

**Q6 How familiar are you with diagrams like flowcharts and even modeling languages like UML?**

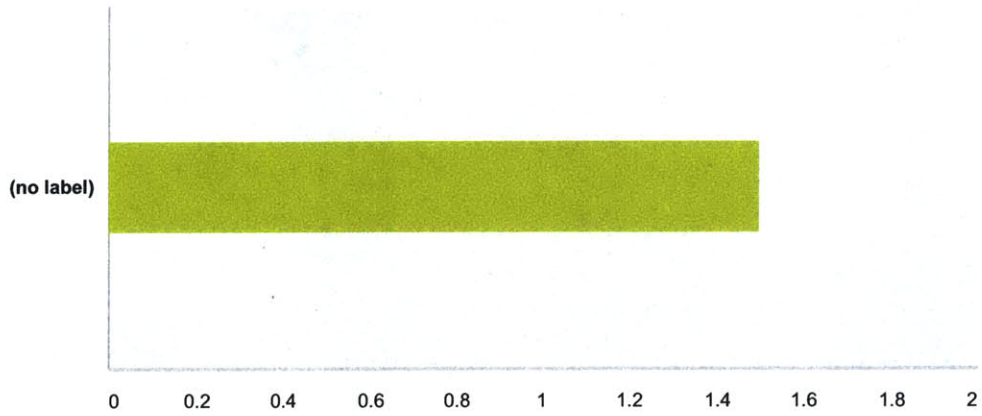
Answered: 8 Skipped: 1



	Very Familiar - I Have Created Them Before	Familiar - I Have Interacted With Them Before	Not Very Familiar - I Rarely Use Them	Unfamiliar - I Have Never Used Them	Total	Weighted Average
(no label)	37.50% 3	25.00% 2	37.50% 3	0.00% 0	8	2.00

**Q7 How familiar are you with on-demand media systems like HBO, Netflix, Hulu, and Youtube?**

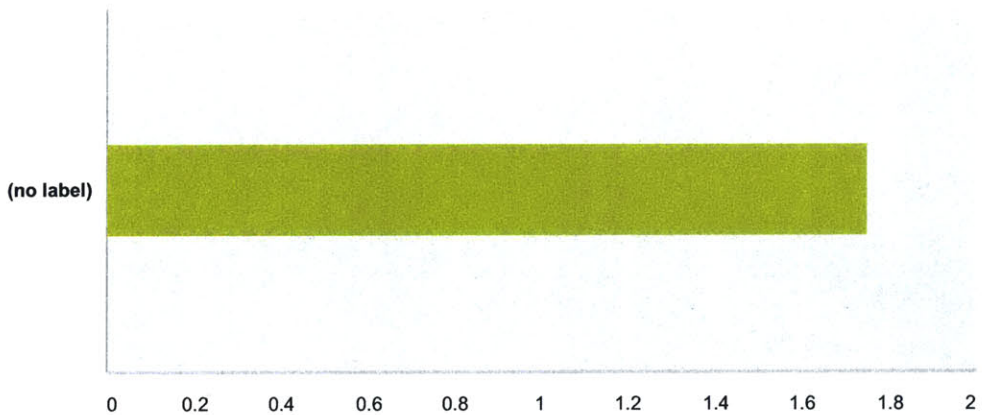
57  
Answered: 8 Skipped: 1



	Very Familiar - I Have Used More Than One System Before Extensively	Familiar - I Have Used One System Before Extensively	Not Very Familiar - I Have On-Demand Systems Only Briefly	Unfamiliar - I Have Never Used an On-Demand System	Total	Weighted Average
(no label)	50.00% 4	50.00% 4	0.00% 0	0.00% 0	8	1.50

### Q8 How familiar are you with traditional television?

Answered: 8 Skipped: 1



	Very Familiar - I Watch Television Frequently	Familiar - I Have Watched Television Before But Not Frequently	Not Very Familiar - I Do Not Watch Television Frequently or Have Not Recently	Unfamiliar - I Do Not Watch Television and Do Not Remember What The Experience is Like	Total	Weighted Average
(no label)	37.50% 3	50.00% 4	12.50% 1	0.00% 0	8	1.75

## 9 Bibliography

### References

- [1] T. Onion. Netflix introduces new 'browse endlessly' plan. [Online]. Available: [https://www.youtube.com/watch?v=3\\_Bm2WUYBxU](https://www.youtube.com/watch?v=3_Bm2WUYBxU)
- [2] N. C. Shu, *Visual programming*. New York : Van Nostrand Reinhold, c1988., 1988.
- [3] (2007, April) Flowchart example. [Online]. Available: <https://en.wikipedia.org/wiki/File:Tax.PNG>
- [4] A. Kay. Th early history of smalltalk. [Online]. Available: [http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk\\_III.htm](http://www.smalltalk.org/smalltalk/TheEarlyHistoryOfSmalltalk_III.htm)
- [5] R. Jacob, "A state transition diagram language for visual programming," *Computer*, vol. 18, no. 8, pp. 51–59, 1985.
- [6] [Online]. Available: <http://www.ni.com/labview/>
- [7] [Online]. Available: <http://www.grasshopper3d.com/>
- [8] [Online]. Available: <http://nodered.org/>
- [9] K. Lodding, "Iconic interfacing." *IEEE Computer Graphics & Applications*, vol. 3, no. 2, p. 11, 1983.
- [10] P. Langely, "Machine learning for adapative user interfaces," in *KI-97 Advances in Artificial Intelligence*, 1997, pp. 53–62.
- [11] D. Chamberlain, "Television interfaces," *Journal of Popular Film and Television*, 2010.
- [12] d.-r. Trindade, Daniel<sup>1</sup> and a.-r. Raposo, Alberto<sup>1</sup>, "Improving 3d navigation techniques in multiscale environments: a cubemap-based approach." *Multimedia Tools & Applications*, vol. 73, no. 2, pp. 939 – 959, 2014.
- [13] K. H. Y. H. N. S. F. Hirabayashi, "Media-based navigation for hypermedia systems," in *Hypertext '93 Proceedings*, 1993.
- [14] M. RESNICK, J. MALONEY, A. MONROY-HERNÁNDEZ, N. RUSK, E. EASTMOND, K. BRENNAN, A. MILLNER, E. ROSENBAUM, J. SILVER, B. SILVERMAN, and Y. KAFAI, "Scratch: Programming for all." *Communications of the ACM*, vol. 52, no. 11, pp. 60 – 67, 2009.
- [15] [Online]. Available: <https://facebook.com>

- [16] V. Luckerson, "Ifttt has big plans for the internet of things." *Time.com*, p. 1, 2014. [Online]. Available: <http://libproxy.mit.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bthlive>
- [17] Movie-map. [Online]. Available: <http://www.movie-map.com/national+lampoon-27s+van+wilder.html>
- [18] How netflix is changing the tv industry. [Online]. Available: <http://www.investopedia.com/articles/investing/060815/how-netflix-changing-tv-industry.asp>
- [19] M. Snider. Cutting the cord: A new user interface coming to netflix on the web. [Online]. Available: <http://www.usatoday.com/story/tech/2015/06/14/cutting-the-cord-new-netflix-interface/28024595/>
- [20] Complete list of netflix genres. [Online]. Available: <http://www.bestmoviesonnetflix.com/netflix-hacks/complete-list-of-netflix-genres/>
- [21] K. Bell. (2015, March) Netflix 'god mode' gets rid of the site's terrible horizontal scrolling. [Online]. Available: <http://mashable.com/2015/03/18/netflix-god-mode/>
- [22] M. Rhodes. (2015, May) Netflix's redesign will finally ditch the slow carousels. [Online]. Available: <http://www.wired.com/2015/05/netflixs-redesign-will-finally-ditch-slow-carousels/>
- [23] J. Thibeault. (2014, July). [Online]. Available: <http://blog.limelight.com/2014/07/using-second-screen-to-build-deep-and-meaningful-relationships/>
- [24] W. Stockard. (2010, October). [Online]. Available: <http://www.reuters.com/article/2010/10/20/idUS223149+20-Oct-2010+MW20101020>
- [25] Icomoon app. [Online]. Available: <https://icomoon.io>
- [26] Socket.io. [Online]. Available: <http://socket.io/>
- [27] Toolbars. [Online]. Available: [https://msdn.microsoft.com/en-us/library/windows/desktop/Dn742395\(v=VS.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/Dn742395(v=VS.85).aspx)
- [28] G. Linden, B. Smith, and J. York, "Amazon.com recommendation - item-to-item collaborative filtering." *IEEE INTERNET COMPUTING*, vol. 7, no. 1, pp. 76 – 80, n.d.