Towards Understandable Software

Anna Liberty

2026-10-24 Sat

Outline

1. Introduction
2. Free software as user agency
3. The Curse of Code
4. Paths forward
5. Towards Understandable Software
6. Conclusion

Anna Liberty



- Studies Physics & Applied Computing at UW Bothell
- Researches radio astronomy
- Programs in Zig
- Writes at `https://gracefulliberty.com`

What is free software about?

What is free software about?

Free software is about a lot of things:

- The joy of computing
- Personal itch-scratching
- Copyleft
- Particular projects
- Community
- Developer freedom
- Control over our own computers

Negative freedoms are not enough

We need more than free licenses to control our computers.

- The right to work doesn't mean you can actually find a job.
- The right to free speech doesn't mean people actually get heard
- The right to a program's source code doesn't mean you can actually change it.

Values imply imagination

These values require imagining a better world and creating it.

That takes effort and isn't always fun, but we can find joy in it.

Together, we serve the users.

An archaic language

- Barely 2D communication method
- Easy to create syntax errors
- Historical accident

LLMs are false hope

"If you do not control the source code, you are not in control." — Andrew Kelley, "Don't Take the Black Pill"

LLMs are false hope

If you do not control the source (???), you are not in control.

LLMs are generally harmful to society at large

- Destroying the open web
- Stealing code
- Destroying literature
- Spreading mass ignorance
- Harming the environment
- Deskilling workers

Open models won't save us

- Open models can't compete
- Open models aren't open
- Open models don't help the core issue

Documentation all the way down

Programs described in natural language, not just computer language.

People shouldn't have to read the source to understand how it works.

Software should be self-documenting, with up-to-date documentation that any user can read and understand.

Documentation all the way down

All software should be Emacs but good.

Literate programming

The first step to making understandable software is to make complete software.

We need a change of priorities.

> *Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.*
> — *Donald Knuth - Literate Programming*

Literate programming - example

Both natural and computer language in one document.

```
▼ Problem 2

\[
A = \left[\begin{matrix}
 2 & 3
\end{matrix}
\right],
B = \left[\begin{matrix}
 -1 & 4 \cr
 2 & -1
\end{matrix}
\right],
C = \left[\begin{matrix}
 -1 \cr
 2
\end{matrix}
\right]
\]

For the above matrices, I am to find the matrix product in the order A(BC) and
(AB)C.

name: problem2
src python
  # Problem 2
  A = array([[2,3]])
  B = array([[-1,4],[2,-1]])
  C = array([[-1],[2]])
  A_BC = matmul(A, matmul(B,C))
  AB_C = matmul(A, matmul(B,C))
  print("Problem 2:")
  print(" A(BC) = ", A_BC)
  print(" (AB)C = ", AB_C)
src
```

2 Problem 2

$$A = \begin{bmatrix} 2 & 3 \end{bmatrix}, B = \begin{bmatrix} -1 & 4 \\ 2 & -1 \end{bmatrix}, C = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

For the above matrices, I am to find the matrix product in the order A(BC) and (AB)C.

```
# Problem 2
A = array([[2,3]])
B = array([[-1,4],[2,-1]])
C = array([[-1],[2]])
A_BC = matmul(A, matmul(B,C))
AB_C = matmul(A, matmul(B,C))
print("Problem 2:")
print(" A(BC) = ", A_BC)
print(" (AB)C = ", AB_C)
```

These both produce matrices with the single value 6.

Literate programming - example

ReTangled in ReTangled

````
## Document Parser

Each codeblock needs to be parsed with some information in it. This needs to include
the metadata, the contents, and some information that must be stored within. Because
tags can be made within codeblocks that reference other codeblocks by their ID,
these must be parsed within the codeblocks, along with the indentation of the tag
(to be added back in after).

The actual parsing has four steps:
1. Identify a codeblock (that is not within another codeblock)
2. Parse the codeblocks' metadata (which may be on another line)
3. Identify tags within a codeblock
4. Repeat until end of file

ReTangled supports Pandoc Markdown, GitHub Flavored Markdown (GFM), and Org Mode.

``` {.zig #parser}
pub const Mode = enum {
    pandoc,
    gfm,
    org,
};
```

In Markdown, code blocks are started with ` ``` ` and `~~~`. In Org Mode, they are
started `#+begin_src`.

``` {.zig #parse-document-inner}
const block_starts = switch (mode) {
    .pandoc, .gfm => [_][]const u8{
        "```",
        "~~~",
    },
    .org => [_][]const u8{"#+begin_src"},
};
```
````

```zig
const std = @import("std");
const Allocator = std.mem.Allocator;
const ArrayList = std.ArrayList;

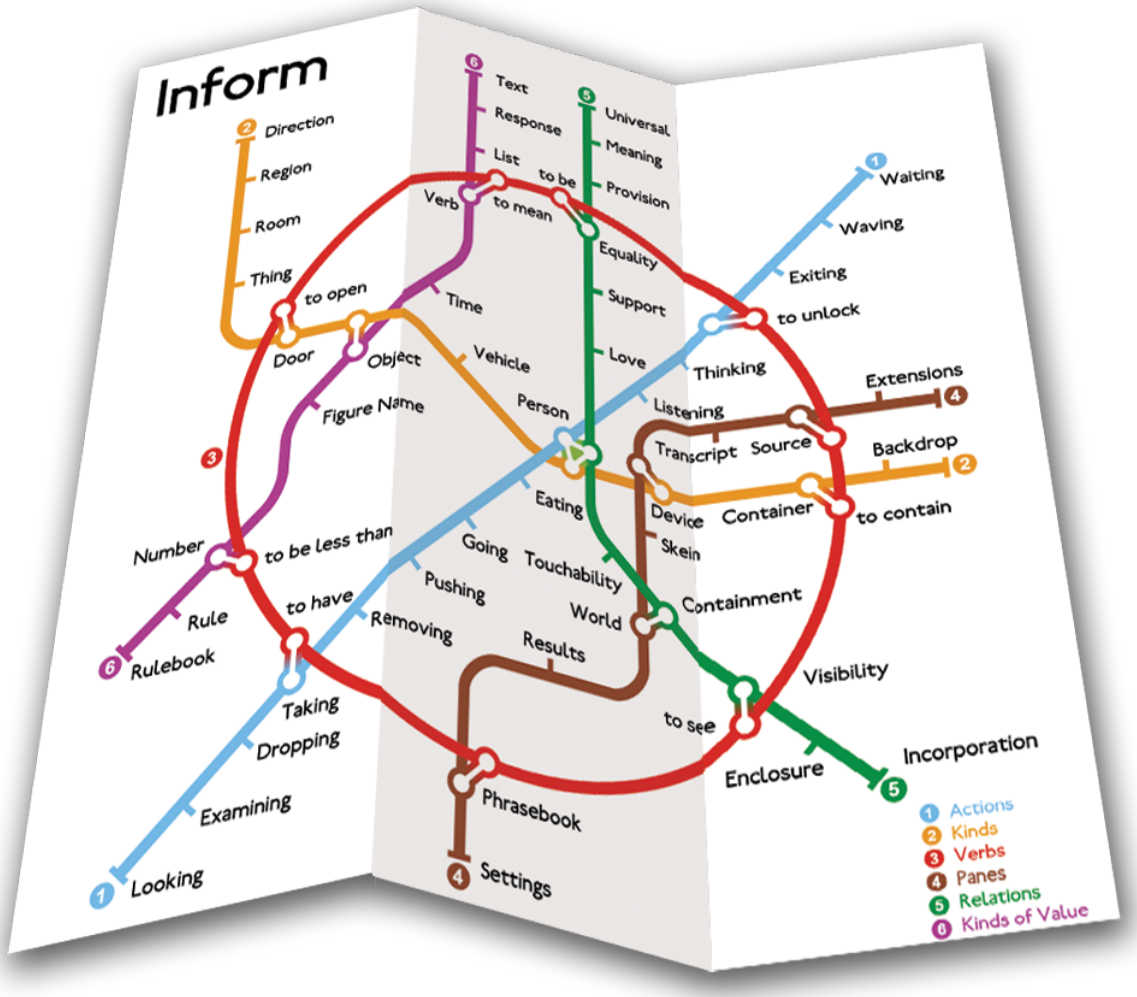
const Parser = @This();

text: []const u8,
sections: ArrayList(Section),
allocator: Allocator,

pub fn create(allocator: Allocator, text: []const u8) error{OutOfMemory}!Parser {
    return .{
        .text = text,
        .sections = try ArrayList(Section).initCapacity(allocator, 2048),
        .allocator = allocator,
    };
}

pub fn destroy(self: *Parser) void {
    for (self.sections.items) |*section| {
        if (section.block) |*block|
            block.contents.deinit(self.allocator);
    }
    self.sections.deinit(self.allocator);
}
```

Literate programming - inform







Literate programming - Eve

Flappy Eve

Setup
Draw the game world!

Game menus
Score calculation
Start a new game

Drawing
Player
Obstacles

Game Logic
Obstacles
Flapping the player
Scroll the world
Collision

Flappy Eve

When a player starts the game, we commit a `#world`, a `#player`, and some `#obstacles`. These will keep all of the essential state of the game. All of this information could have been stored on the world, but for clarity we break the important bits of state into objects that they effect.

- The `#world` tracks the distance the player has travelled, the current game screen, and the high score.
- The `#player` stores his current y position and (vertical) velocity.
- The `obstacles` have their (horizontal) offset and gap widths. We put distance on the world and only keep two obstacles; rather than moving the player through the world, we keep the player stationary and move the world past the player. When an obstacle goes off screen, we will wrap it around, update the placement of its gap, and continue on.

Setup

Add a flappy eve and a world for it to flap in:

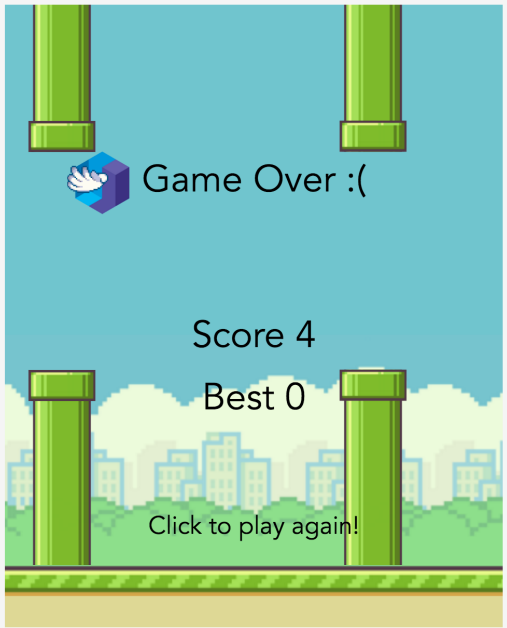
```
commit
  [#player #self name: "eve" x: 25 y: 50 velocity: 0]
  [#world screen: "menu" frame: 0 distance: 0 best: 0 gravity: -0.061]
  [#obstacle gap: 35 offset: 0]
  [#obstacle gap: 35 offset: -1]
```

Next we draw the backdrop of the world. The player and obstacle will be drawn later based on their current state. Throughout the app we use resources from @bhauman's flappy bird demo in clojure. Since none of these things change over time, we commit them once when the player starts the game.

Draw the game world!

```
search
  world = [#world]

commit @browser
  world <- [#div style: [user-select: "none" -webkit-user-select: "none" -
moz-user-select: "none"]  children:
    [#svg #game-window viewBox: "10 0 80 100", width: 480 children:
      [#rect x: 0 y: 0 width: 100 height: 53 fill: "rgb(112, 197, 206)" sort:
```



Natural language programming - I

Imagine:

Natural language programming - I

Imagine:

- Literate programming without the code

Natural language programming - II

Imagine:

- Literate programming without the code
- Language models that are backed by clarity

Natural language programming - II

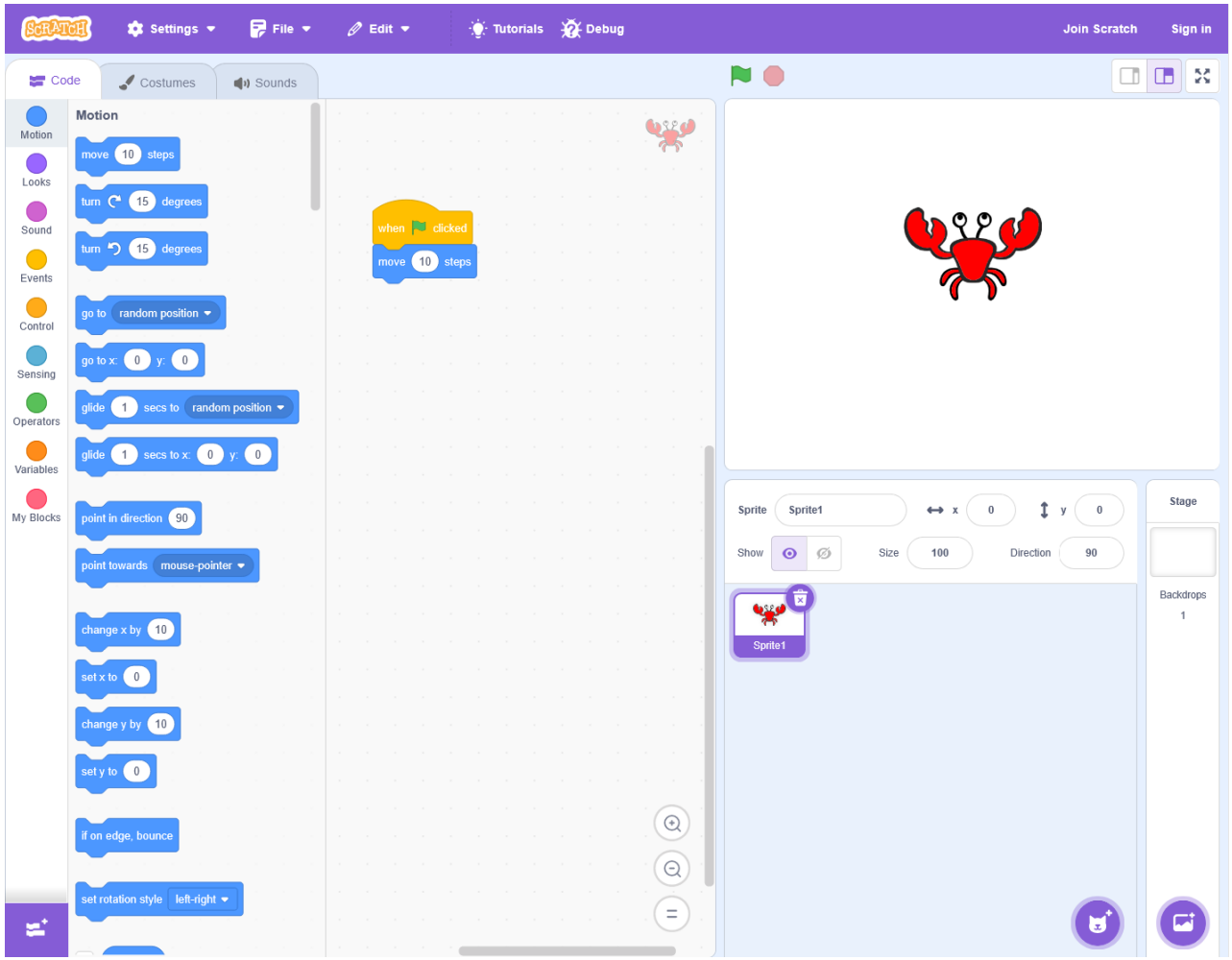
Imagine:

- Literate programming without the code
- Small language models
- Systems of interconnected definitions

Visual programming

But we can go further than text. . .

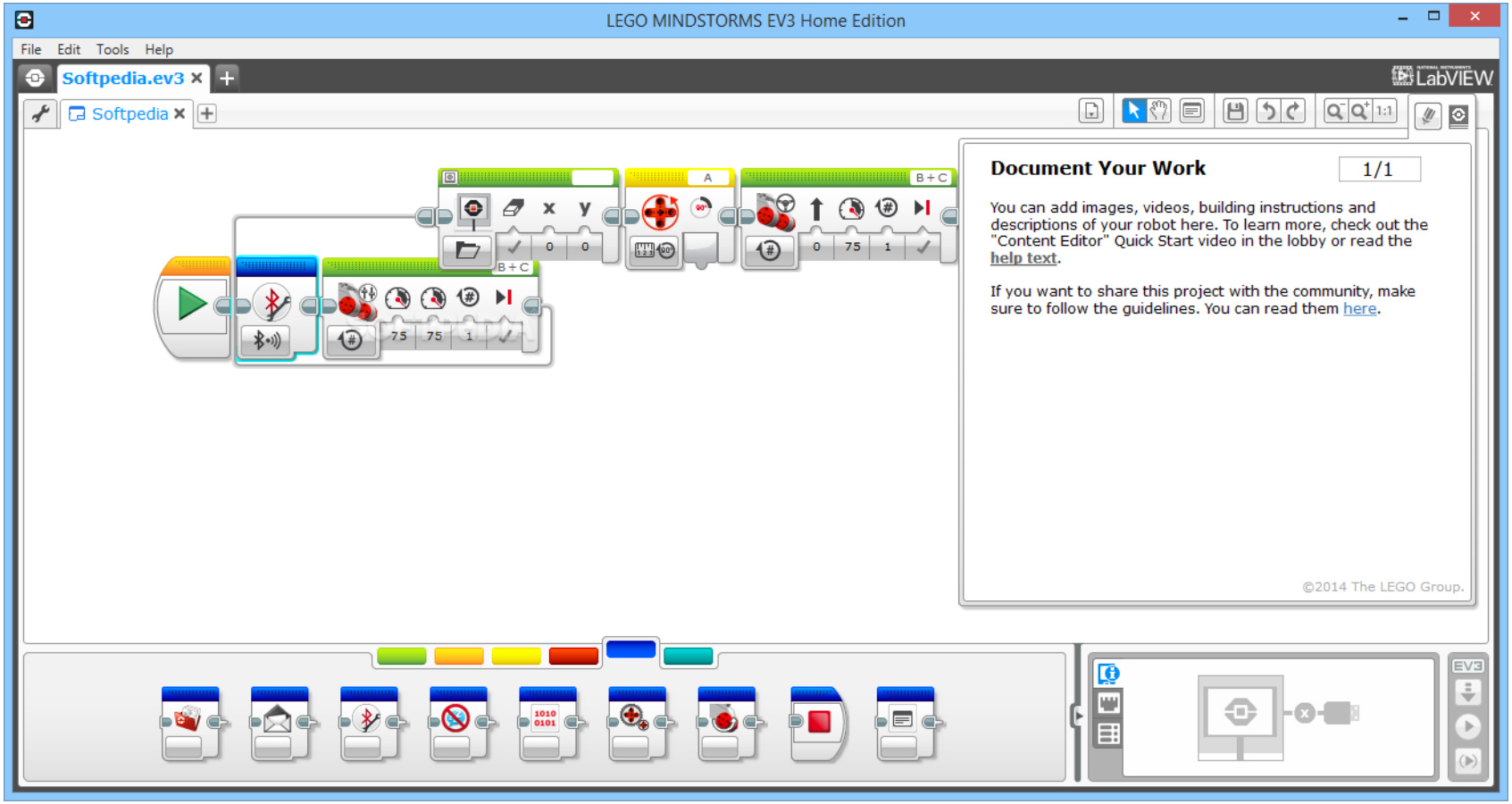
Visual programming - Scratch



By MouseCursor - Own work, CC BY-SA 2.0,

https://commons.wikimedia.org/w/index.php?curid=164027640

Visual programming - Mindstorms



Visual programming

Used

- Widely in educational settings
- Occasionally for scripting larger programs
- Rarely beyond

Benefits

- Syntax errors impossible
- Type errors impossible
- Full graphical expression
- Accessible to more interfaces
- Mitigate logic errors?

Combine them, experiment

The way forward isn't static.

We need many ideas of a better future.

Try a path out!

ReTangled

- Literate programming tool
- Documented with ReTangled

Alonzo

- Functional visual programming
  - See Dyuti De's talk yesterday for an idea for the vision
- Created by Adrian Cochrane

Connect with me!

- Follow my work on ReTangled and similar projects
- Read more of my thoughts on Literate Programming
- Encourage me in the pursuit of understandable software

@liberty@mathstodon.xyz
`https://gracefulliberty.com`
`https://codeberg.org/liberty/retangled`