

Please do not redistribute slides without prior permission.



A Semester at University: Teaching Software Engineering in DLang

Social: [@MichaelShah](https://twitter.com/MichaelShah)

Web: mshah.io

Courses: courses.mshah.io

YouTube:

www.youtube.com/c/MikeShah

Presentor: Mike Shah, Ph.D.

11:00-11:30, Wed, Aug. 30, 2023

Audience: Everyone!

Here's the **main takeaway** (tl;dr)

- **Using the D Language substantially improved the software engineering course I taught**
 - Other faculty and trainers should take a close look at D Lang.
- Why?
 - Watch the rest of the presentation

Here's the main takeaway (tl;dr)

- Using the D Language substantially improvement the software engineering course I taught
 - Other fa
- How?
 - Watch th

Slides available on my website
at www.mshah.io after the talk

Your Guide for Today

by Mike Shah

- Associate Teaching Professor at Northeastern University in Boston, Massachusetts.
 - I teach courses in computer systems, computer graphics, and game engine development.
 - My research in program analysis is related to performance building static/dynamic analysis and software visualization tools.
- I do consulting and technical training on modern C++, Concurrency, OpenGL, and Vulkan projects (and hopefully D projects!)
 - (Usually graphics or games related)
- I like teaching, guitar, running, weight training, and anything in computer science under the domain of computer graphics, visualization, concurrency, and parallelism.
- Contact information and more on: www.mshah.io
- More online training coming at courses.mshah.io



Abstract

The abstract that you read and enticed
you to join me is here!

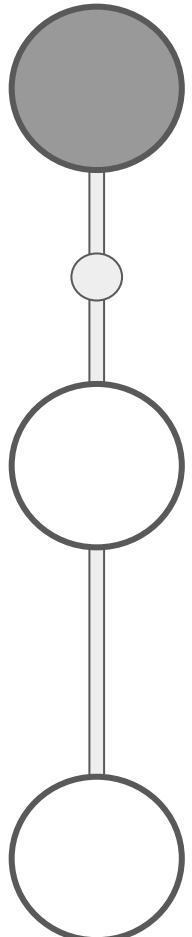
In January of 2023, I excitedly showed a group of over 110 university students that D is the 46th most popular programming language on the Tiobe Index (for whatever it's metrics are)-- and then told those students they would be learning DLang in the software engineering course this semester. In this talk, I will recap my university curriculum of how I taught DLang and why I think DLang should be considered to be taught by more faculty in universities. For this software engineering course we made use of low level access, multiple programming paradigms, built-in profilers, package management (dub) code coverage, ddoc, and unit testing in order to build a half-semester long project. My conclusion is that using DLang at university can enable students a competitive advantage versus other languages, and in this talk I'll reflect on the curriculum, pain points, strengths, and future of D in education.

But don't take my word for it -- you'll hear from some of the students yourself on what they built!

Software Engineering Spring of 2022

Fall of 2022 we got some visitors!

First week of class -- Spring 2023



The Software Engineering Course

- **Titled:** Foundations of Software Engineering
 - Basic idea is that this is a course that will prepare students to go on their first internship/co-op,
 - and/or
 - Give students who have some work experience additional skills (individually and working on a team) and another project for their portfolio
- **Audience:** upper-level undergraduates, and masters students
 - (but predominantly masters level students ~85%)
- **Historically:** I taught this course in Modern C++
 - Other instructors at my university use Java or TypeScript at my university -- the choice is theirs

The screenshot shows a course landing page with the following sections:

- Course Content (L1):** A video thumbnail titled "COE-5500 Module 1: Course Introduction". Below it is a list of topics:
 1. Developer Etiquette
 2. Tools & Resources
 3. Programming Languages
 4. Design Patterns
 5. Design Principles
 6. Managing Projects
 7. Managing Yourself
- Code Examples:** A snippet of DLang code:

```
/// Hello, Dlang!
/// A pragmatic software engineering
/// programming language!
import std.stdio;
void main(){}
    writeln("Hello Class!");
```
- Books:** Links to "The Pragmatic Programmer" by Andrew Hunt and "FSE: Foundations of Software Engineering" by the instructor.
- Quick Links:** Piazza Forum Board | Office Hours Sign up | Teams Channel

Schedule/Road Map:

The following is our tentative syllabus for the course, changes should be expected throughout the semester. I will announce in class, piazza, or through e-mail any major changes.

Week	Date	Lecture and Readings	Assignments	Note(s)
1	Wednesday - January 11, 2023	Module 1 - Course Introduction, DLang	A1 - D Exercises - out (Due Jan. 20 Anywhere on Earth)	Welcome back to class!
2	Wednesday - January 18, 2023	Module 2 - Programming Idioms and Memory	A2 - Code Review and Data Structure (Due Jan. 27 Anywhere on Earth)	
3	Wednesday - January	Module 3 - Debugging in DLang	A3 - Debugging (Due Feb. 3)	

Software Engineering Course Learning Objectives

- At many universities a software engineering course can have a reputation for:
 - Teaching trivia (e.g. “what is agile”)
 - Important to know for software culture, we cover it -- but it can’t be the only thing.
 - (I have been asked that exact questions on an interview before...)
- My course is built to have students build real software in teams -- project-based learning.
 - Team and human skills (ethics) must be built
 - Programming skills (idioms, patterns, tools, debugging, profiling, etc.) must be built
 - Both equally important

The screenshot shows a course website layout. At the top, there's a navigation bar with links like "Course Home", "About", "Syllabus", "Schedule", "Readings", "Assignments", "Gradebook", and "Contact". Below the navigation is a main content area. On the left, there's a sidebar with "Week 1: Course Introduction" and "Week 2: Programming Idioms and Memory". The main content area has several sections: "Course Overview" (with a video thumbnail), "Course Readings" (listing "Hello DLang" and "The Pragmatic Programmer"), "Course Materials" (listing "FSE: Foundations of Software Engineering" and "Software Engineering Conference"), and "Quick Links" (Piazza Forum Board, Office Hours Sign up, Teams Channel). Below these is a "Schedule/Road Map" section with a table:

Week	Date	Lecture and Readings	Assignments	Note(s)
1	Wednesday - January 11, 2023	Module 1 - Course Introduction, DLang	A1 - D Exercises - out (Due Jan. 20 Anywhere on Earth)	Welcome back to class!
2	Wednesday - January 18, 2023	Module 2 - Programming Idioms and Memory	A2 - Code Review and Data Structure (Due Jan. 27 Anywhere on Earth)	
3	Wednesday - January	Module 3 - Debugging in DLang	A3 - Debugging (Due Feb. 3)	

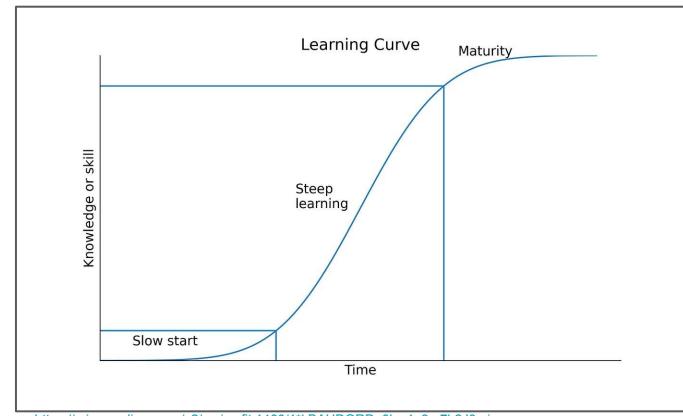
I observed problem(s) in my previous course iterations

- While students successfully were introduced or learned many tools...
 - (e.g. github, gdb, static analysis tools, code coverage, CI/CD, etc.)

The ability to successfully utilize the C++ language was a barrier in a team project

The C++ Learning Curve is Steep

- Most students were excited and motivated to learn C++,
 - Most students had no prior C++ experience.
 - I spent lots of course time teaching C++ foundations to try to get everyone on the same page
 - We were limited to also using the [SFML](#) and [Catch2](#) as our only dependencies in the course
- In student projects (teams of 4), often 1 or 2 programmers who would dominate
 - If students fell behind in learning programming techniques
 - technical contributions to project would be limited, tension could form on the human side.



https://miro.medium.com/v2/resize-fit:1400/1*kBAHDORDq0bw1g9m7h2J0q.jpeg

Interestingly...

- The feedback I got from teaching my course in Modern C++ was:
 - Exceptionally high (e.g. average of 4.8/5.0 over 3 iterations)
 - Good news -- I get to keep my job :)
 - Some subset of students even got C++ development jobs at good tech companies
 - Sounds like I should pat myself on the back -- but not quite!



<https://i.pinimg.com/originals/c1/b2/ef/c1b2ef47207816628de90a60d8ae0b4e.jpg>

Something was not right with the C++ course

1. The efforts from myself and course staff far exceeded other courses I had developed
 - a. **20+ hours per week on one course** -- (often debugging and reteaching C++)
 - i. How can I manage my 2 other courses, research, service, conference talks, etc.?
2. **C++ is a massive language**
 - a. It has its own pros and cons -- but **it was not a great choice of a language for the course.**
 - b. Students who fall behind in any technical aspect of the course, can't really catch up
 - i. This leaves more gaps in learning
 - ii. Students are in a more frantic or high-stress mode when trying to learn.
3. The **original reason(s) for using C++** was the lack of C++ being taught in university
 - a. (...and my own professional background using the language)

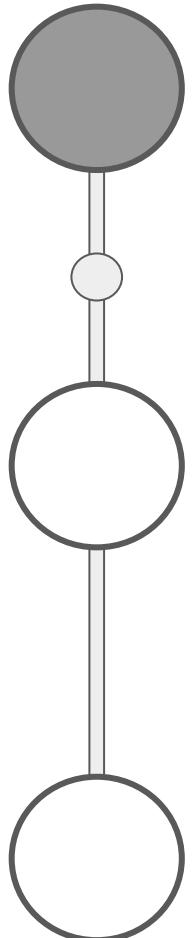
Something was not right with the C++ course

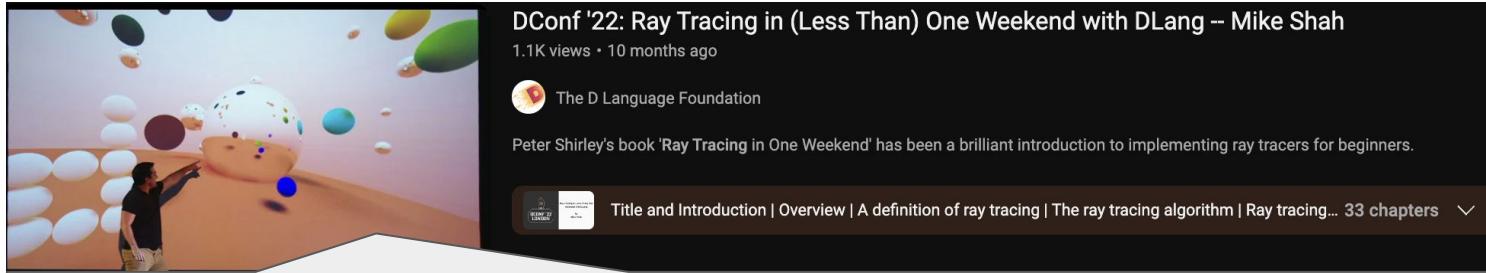
1. Change needed to be made
 - I had some amount of guilt abandoning C++ which I was known for
2. Some subset of students would never learn C++ in university
 - It seemed reasonable to push onwards on C++
 - But 3 semesters of evidence was enough for me to see students clawing their way through syntax versus building software
3. (...and my own professional background using the language)

Software Engineering Spring of 2022

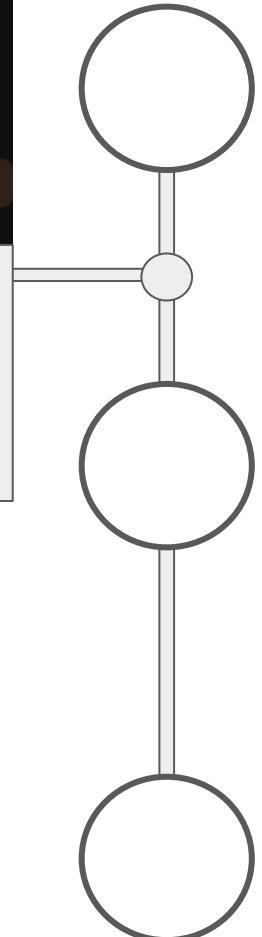
Fall of 2022 we got some visitors!

First week of class -- Spring 2023





- Summer of 2022 I reveal I'm using DLang in a programming language-agnostic course on ray tracers
- My own personal projects and tools also simultaneously being converted from Python to DLang

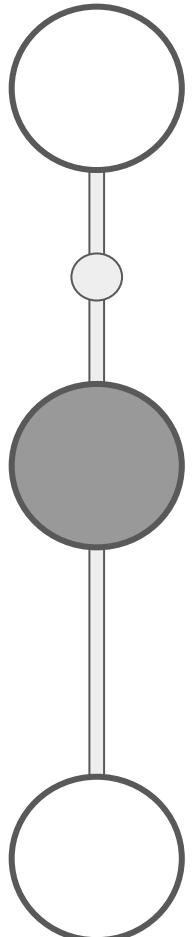


First week of class -- Spring 2023

Software Engineering Spring of 2022

Fall of 2022 we got some visitors!

First week of class -- Spring 2023



Fall of 2022

- Ali Çehreli (and Steven Schveighoffer) visited Northeastern University to meet students and give an Introduction to D
 - We had a very enthusiastic group of students (and faculty) who attended
- This encouraged me further that students would be responsive to a new language for a software engineering course



Invited Speaker Ali Çehreli, Introduction to Dlang

Invited Speaker Ali Çehreli, Introduction to Dlang
https://www.youtube.com/watch?v=0JL9uT_XGZE



“ D shines from low-level control to high-level abstraction ”

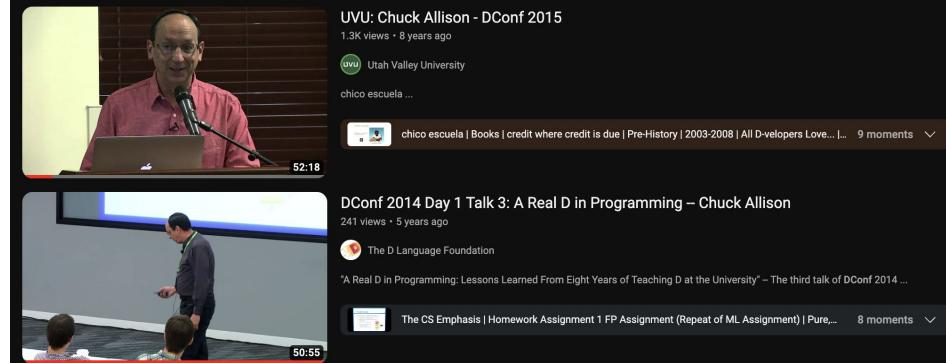
[Success stories](#) [What is D used for?](#)

Further Ruminations on D

- Before committing to the language for Spring of 2023 I reflected on the many potential positives:
 - D ecosystem had nearly everything in one place
 - D had multiple compilers for stability (dmd, ldc, ldc)
 - gdb or lldb available to teach debugging
 - Profiler available (-profile, -profile=gc)
 - Package manager (dub)
 - ddoc (documentation)
 - package manager, code coverage, profiler,
 - unittest for teaching testing
 - Multiple paradigms (could show benefits of functional programming, generic design, OOPP patterns)
 - Can talk about static analysis (dfmt, scanner)
 - Can talk about compile-time versus run-time
- Very importantly, D was industry proven
 - Our university students are pragmatic -- I'm lucky to teach in such a culture.
 - Our students do look on indeed.com , university job postings, etc. expecting Dlang however!
 - This is one area I hope to see more job postings on (more on that later)

Prior Art

- There was also precedent for teaching DLang
 - Chuck Allison (Utah Valley University) led the way (as early as 2014) in his programming languages course using DLang
 - (Aside: I found Professor Paul Buis at Ball State in 2015 also offers a D lang workshop [[link](#)])



All D-velopers Love...

D compiles to *native code*

Universal Function Call Syntax

(Optional) **Garbage Collection**

Compile-Time Function Evaluation

Module System

Mixins (string and template)

Slices

unittest

Associative Arrays

Contract Programming

static if

debug

Critically -- Identified Freely Available Resources

- D being a smaller community, it was important to have resources for students to turn to.

[DLang Tour](#)

Welcome ▾ D's Basics ▾ D's Gems ▾ Multithreading ▾ Vibed ▾ D by Examples ▾ DUB packages ▾

Welcome to D

Welcome to the interactive tour of the *D Programming language*.

The tour gives an overview of this powerful and expressive language which compiles directly to efficient, native machine code.

What is D?

D is the culmination of *decades of experience implementing compilers* for many diverse languages and has a unique set of features:



- *high level* constructs for great modeling power
- *high performance*, compiled language
- static typing
- direct interface to the operating system API's and hardware
- blazingly fast compile-times
- memory-safe subset (SafeD)
- *maintainable, easy to understand* code
- gradual learning curve (C-like syntax, similar to Java and others)
- compatible with C application binary interface
- limited compatibility with C++ application binary interface
- multi-paradigm (imperative, structured, object oriented, generic, functional programming purity, and even assembly)
- built-in error detection (contracts, unittests)

... and many more [features](#).

```
1 import std.stdio;
2 import std.algorithm;
3 import std.range;
4
5 void main()
6 {
7     // Let's get going!
8     writeln("Hello World!");
9
10    // An example for experienced programmers:
11    // Take three arrays, and without allocating
12    // any new memory, sort across all the
13    // arrays inplace
14    int[] arr1 = {4, 9, 7};
15    int[] arr2 = {5, 2, 1, 10};
16    int[] arr3 = {6, 8, 3};
17    sort(chain(arr1, arr2, arr3));
18    writeln("%s\n%s\n%s", arr1, arr2, arr3);
19    // To learn more about this example, see the
20    // "Range algorithms" page under "Gems"
21 }
```



<https://tour.dlang.org/>

<http://ddili.org/ders/d.en/>

Some Final Inspiration -- DLang's fit a teaching language

- **To other faculty:** D has a very interesting potential to scale throughout the university curriculum (different paradigms, systems language, etc.)
 - It could probably be utilized in a CS1 course through advanced graduate courses.
 - The quote below is from Ali's book [Programming in D](#) from Andrei Alexandrescu

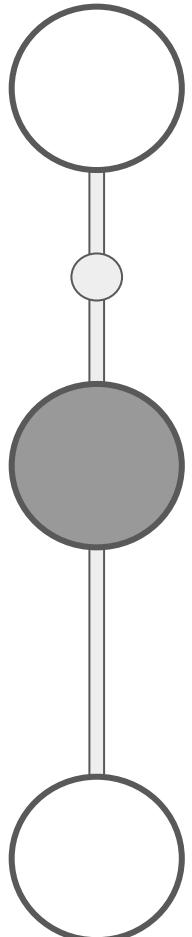
I've long suspected D is a good first programming language to learn. It exposes its user to a variety of concepts – systems, functional, object oriented, generic, generative – candidly and without pretense. And so does Ali's book, which seems to me an excellent realization of that opportunity.

Andrei Alexandrescu
San Francisco, May 2015

Software Engineering Spring of 2022

Fall of 2022 we got some visitors!

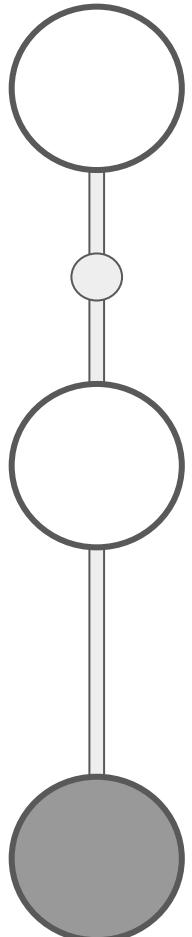
First week of class -- Spring 2023



Software Engineering Spring of 2022

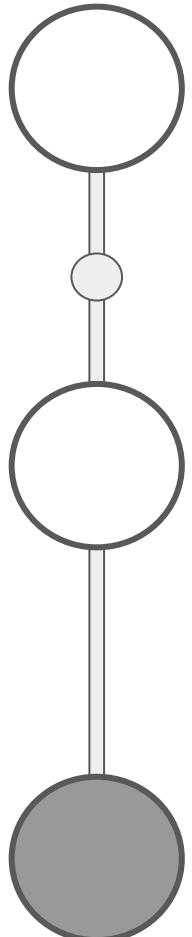
Fall of 2022 we got some visitors!

First week of class -- Spring 2023



Software Engineering Spring of 2022

Fall of 2022
the visitors!



First week of class -- Spring 2023

One of the first slides introducing DLang in my course

- I ask my students if anyone had heard of the D programming language (DLang)
 - There were maybe 1-2 students who raised their hands out of 110 students
 - (A handful had also looked at D from my course page)

Question to the Audience

- What have you heard?
 - Have you heard of DLang?

The next slide...

- I ask my students how many have heard of C++
 - Of course -- many students hands raised up quickly
 - Many students were also aware I had previously taught iterations of the course in Modern C++

Question to the Audience (1/2)

- How many have heard of C++?
 - How many have used C++?

Undergraduate and Masters Students Frame of Mind (1/2)

- So...my next slide I told students they are not going to be learning C++, but the **46th** most popular language
-- DLang
 - The language many students had not heard of!

TIOBE Index for January 2023

January Headline: C++ is TIOBE's Programming Language of the Year 2022!

C++ is TIOBE's programming language of the year 2022. It has won this title because C++ gained most popularity (+4.62%) in 2022. Runners up are C (+3.82%) and Python (+2.78%). Interestingly, C++ surpassed Java to become the number 3 of the TIOBE Index in November 2022. The reason for C++'s popularity is its excellent performance while being a high level object-oriented language. Because of this, it is possible to develop fast and vast software systems (over millions of lines of code) in C++ without necessarily ending up in a maintenance nightmare.

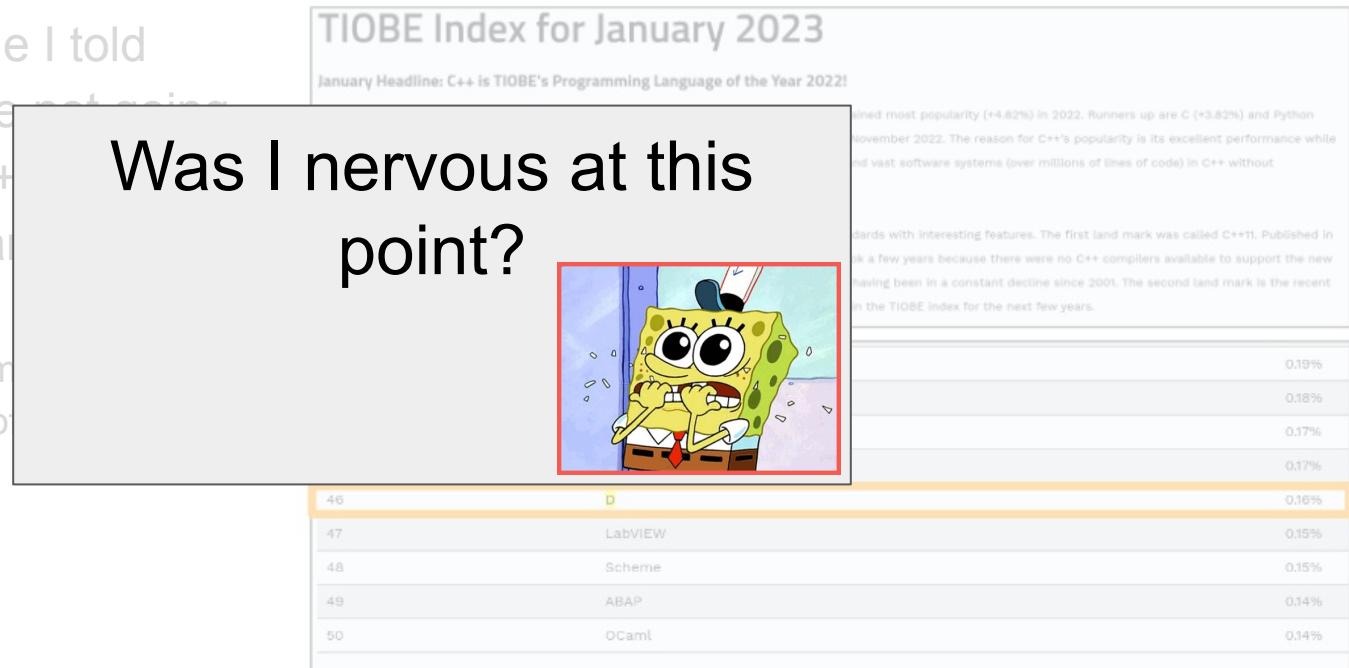
Another reason for C++'s rise is its "recent" and constant publication of new language standards with interesting features. The first land mark was called C++11. Published in 2011, it was the first considerable change since 1988. The adoption of this new standard took a few years because there were no C++ compilers available to support the new language definition. Because of C++11, C++ was slowly going uphill in the TIOBE Index after having been in a constant decline since 2001. The second land mark is the recent C++20 publication, which for instance introduced modules. It will probably lift C++ further in the TIOBE Index for the next few years.

42	Awk	0.19%
43	Prolog	0.18%
44	CFML	0.17%
45	Haskell	0.17%
46	D	0.16%
47	LabVIEW	0.15%
48	Scheme	0.15%
49	ABAP	0.14%
50	Ocaml	0.14%

Undergraduate and Masters Students Frame of Mind (2/2)

- So...my next slide I told students they are going to be learning C++
46th most popular
-- DLang

- The language may have not heard of



- You have to put yourself in the mind of a student:
 - Many are upper-level undergraduates or masters students
 - Students are very job focused -- they are at a school that requires and has a strong co-op/internship program

Something

students they are
to be learning C++
46th most popular
-- DLang

- The language may have had not heard of

January Headline: C++ is TIOBE's Programming Language of the Year 2022!

C++ gained most popularity (+4.82%) in 2022. Runners up are C (+3.82%) and Python (+3.62%). C++ has been the most popular programming language since November 2022. The reason for C++'s popularity is its excellent performance while being able to handle vast software systems (over millions of lines of code) in C++ without problems.

SpongeBob SquarePants was nervous about his first day of school. He was worried about fitting in and making friends. He also didn't know what he wanted to be when he grew up. But with the help of his best friend, Patrick Star, he overcame his fears and found his place in the world.

Standards with interesting features. The first landmark was called C++11. Published in 2011. It took a few years because there were no C++ compilers available to support the new features. Since then, C++ has been in a constant decline since 2001. The second landmark is the recent C++20 standard. It will be released in 2020. It is expected to bring many more changes to the language. The third landmark is the recent C++23 standard. It will be released in 2023. It is expected to bring even more changes to the language. The fourth landmark is the recent C++26 standard. It will be released in 2026. It is expected to bring even more changes to the language. The fifth landmark is the recent C++29 standard. It will be released in 2029. It is expected to bring even more changes to the language. The sixth landmark is the recent C++32 standard. It will be released in 2032. It is expected to bring even more changes to the language. The seventh landmark is the recent C++35 standard. It will be released in 2035. It is expected to bring even more changes to the language. The eighth landmark is the recent C++38 standard. It will be released in 2038. It is expected to bring even more changes to the language. The ninth landmark is the recent C++41 standard. It will be released in 2041. It is expected to bring even more changes to the language. The tenth landmark is the recent C++44 standard. It will be released in 2044. It is expected to bring even more changes to the language. The eleventh landmark is the recent C++47 standard. It will be released in 2047. It is expected to bring even more changes to the language. The twelfth landmark is the recent C++50 standard. It will be released in 2050. It is expected to bring even more changes to the language.

46	LabVIEW	0.19%
47	Scheme	0.18%
48	ABAP	0.17%
49	OCaml	0.17%
50		0.16%
		0.15%
		0.15%
		0.14%
		0.14%

Was I nervous to use DLang for a software engineering course? (1/3)

Was I nervous to use DLang for a software engineering course? (2/3)

- No.

Was I nervous to use DLang for a software engineering course? (3/3)

- No.
- Why?
 - Because as a teacher, professor, or engineer -- you **must** choose the right tool for the right job with the information you have.
 - My **responsibility** is to train future engineers to build reliable and performant systems
 - I believed D unlocked that opportunity to do so
 - It's not a perfect language (no language is)
 - But the mere fact that D has **@safe** for instance makes an interesting discussion in class on writing safe software and motivating testing.
 - Or the mere fact that we can use pointers when building a library
 - Or the mere fact that we can ... (and on and on)

(Aside)

- As of August 2023 -- DLang has moved up to 33 (from 46) by the way!
 - **Choosing a tool is not a popularity contest!**
 - And I want other professors and professionals to know that when using DLang!
 - **Choose the right tool**

Position	Programming Language	Ratings
29	Objective-C	0.52%
30	Lisp	0.51%
31	Scala	0.50%
32	Haskell	0.48%
33	D	0.47%
34	Lua	0.47%
35	Dart	0.43%



Curriculum and Assignments

Assignments and Project

- Six assignments individually completed in the first 7 weeks of the course
 - Goal is to have individuals become competent D programmers in about 2 months time
- One team project (4 programmers) with about 11 milestones throughout the remaining 7 weeks of the semester.
 - Goal is to have individuals come together as a team to build something larger (reflecting the real world)
 - Inspired directly by my Statements of Work (SoW) from my personal 10-12 week consulting contracts

The screenshot shows the DLang Tour interface with several assignment sections:

- Assignment1_dlang**: A folder icon next to the title.
- Assignment2_DataStructure**: A folder icon next to the title.
- Assignment3_Debugging**: A folder icon next to the title.
- Assignment4_SDLC**: A folder icon next to the title.
- Assignment5_Dub_Patterns**: A folder icon next to the title.
- Assignment6_Testing**: A folder icon next to the title.

The main area displays the DLang Tour interface with the following content:

Welcome to D

Welcome to the interactive tour of the *D Programming language*.
The tour gives an overview of this powerful and expressive language which compiles directly to efficient, native machine code.

What is D?

D is the culmination of *decades of experience implementing compilers* for many diverse languages and has a unique set of features.

A cartoon character of a person with a large red letter 'D' for a head is shown.

A bulleted list of D's features includes:

- high level constructs for great modeling power
- high performance, compiled language
- static typing
- direct interface to the operating system API's and hardware
- blazingly fast compilation times
- memory safety (Safety 0)
- maintainable, easy to understand code
- gradual learning curve (C-like syntax, similar to Java and others)
- compatible with C application binary interface
- limited compatibility with C++ application binary interface
- multi-paradigm (imperative, structured, object oriented, generic, functional programming purity, and even assembly)
- built-in error detection (contracts, unittests)

and many more features.

```
1 import std.stdio;
2 import std.algorithm;
3 import std.range;
4 ...
5 void main()
6 {
7     // Let's get going!
8     writeln("Hello World!");
9 ...
10    // An example for experienced programmers:
11    // Take three arrays, and without allocating
12    // temporary variables, sort across all the
13    // arrays inplace.
14    int[] arr1 = {4, 9, 7};
15    int[] arr2 = {5, 2, 1, 10};
16    int[] arr3 = {3, 8, 6};
17    mergeInplace(arr1, arr2, arr3);
18    writeln("%s\n", arr1, arr2, arr3);
19    // To learn more about this example, see the
20    // "Range algorithms" page under "Gems"
21 }
```

First 7 Weeks

- Students ramping up on D language and other critical tools
 - OOP, Functional and generic programming
 - gdb, lldb
 - git
 - unittest, unit-threaded

Week	Date	Lecture and Readings	Assignments
1	Wednesday - January 11, 2023	Module 1 - Course Introduction, DLang	A1 - D Exercises - out (Due Jan. 20 Anywhere on Earth)
2	Wednesday - January 18, 2023	Module 2 - Programming Idioms and Memory	A2 - Code Review and Data Structure (Due Jan. 27 Anywhere on Earth)
3	Wednesday - January 25, 2023	Module 3 - Debugging in DLang	A3 - Debugging (Due Feb. 3 Anywhere on Earth)
4	Wednesday - February 01, 2023	Module 4 - The Software Development Life Cycle (SDLC)	A4 - Software Case Study/User Stories (Due Feb. 10 Anywhere on Earth)
5	Wednesday - February 08, 2023	Module 5 - Design Considerations and Design Patterns 1	A5 - Dub Design Pattern (Due Feb. 19 Anywhere on Earth)
6	Wednesday - February 15, 2023	Module 6 - Structural Testing and Functional Testing	A6 - Testing (Due Mar. 1 Anywhere on Earth)
7	Wednesday	Module 7 - Networking and Final Project Overview	

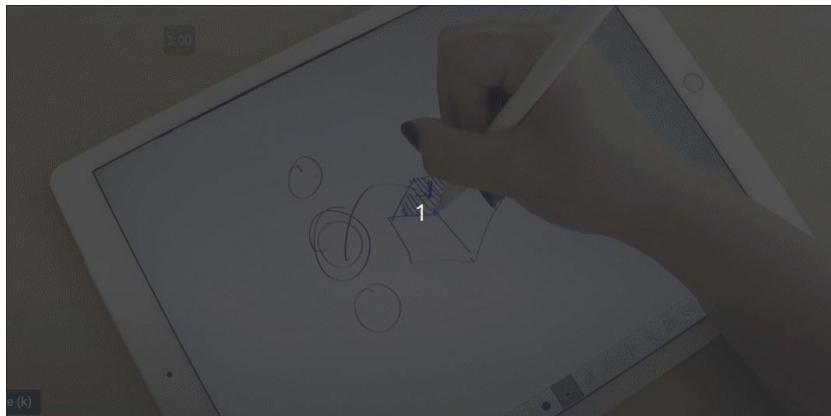
Next 7-8 Weeks

- Students ramping up on D language and other critical tools while building a project
 - Dub - package management
 - Continuous Integration
 - Profiling
 - dscanner, dfmt (d format)
 - SDL (bindbc-sdl)
 - gtkD - GUI library
 - Design Patterns

8	Wednesday - March 01, 2023	 Module 8 - Exam on canvas online -- no in person class
9	Wednesday - March 08, 2023	 Module 9 - No Class
10	Wednesday - March 15, 2023	 Module 10 - Program Analysis and Code Refactoring
11	Wednesday - March 22, 2023	 Module 11 - Design Principles for User Interfaces
12	Wednesday - March 29, 2023	 Module 12 - Testing Strategies
13	Wednesday - April 05, 2023	 Module 13 - Design/Architecture Patterns Tips
14	Wednesday - April 12, 2023	 Module 14 - project work time in class
15	Wednesday - April 19, 2023	 Module 15 - Course wrap up and project work time

(Aside: You'll see more from the students)
Student Project - 7 to 8 weeks

- A collaborative paint Application!
 - Based off of <https://limnu.com/>
 - https://www.youtube.com/watch?v=71L-cuQBgsE&feature=emb_title (34 seconds)
 - A real world startup company that was acquired for \$\$\$ with a real product
 - Good motivation to students
 - **We learned from Ucora earlier -- motivation is everything!**
- Students would exercise skills in
 - Design patterns
 - Networking (today's topics)
 - Non-trivial Testing (e.g. pixel and networking)
 - Algorithmically interesting (i.e. brushes, flood fill, queueing work, etc.)
 - Challenging, but can build an MVP and see feedback!



Where Students Struggled and Excelled

(Professor Perspective)

Where did Students Struggle? (1/2)

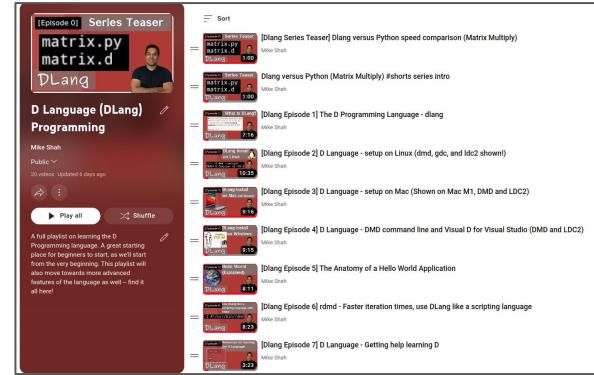
- There were initial struggles with DLang regarding finding specific documentation
 - In some respect I had to vet specific libraries that were actively maintained as best as I could.
 - For myself -- the first run through a course always has some aspect of this
- Some difficulties working across Mac (intel), Mac (M1/M2), Linux, Windows
 - Students have different operating systems that must be supported
 - Using lldb versus using gdb
 - Some resolved by having releasing training videos
 - Personally, this is part of software engineering, but I try my best to align students to similar platforms
- Initial difficulties getting IDE's setup (VSCode and IntelliJ)
 - Many of which I believe have some solution -- though may not have been easy to setup

Where did Students Struggle? (2/2)

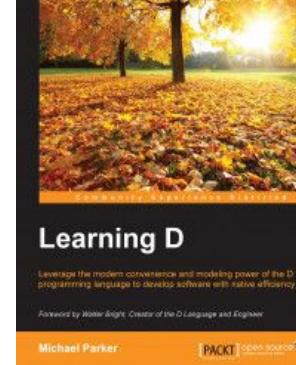
- Some student struggles are natural in the sense of ... well students learning software engineering
 - Learning tools, organizing a team, collaborating using git, etc.
- Some struggles learning how to rely on documentation and the spec as the ground truth when tutorials do not exist (reasonably hard skill to acquire!)
 - The feedback was that this was strictly because of using DLang (and there are indeed often less tutorials)
 - The reality is **I cannot (and did not to the best of my knowledge) tell students they would have this same difficulty with C++, Java, TypeScript, etc.**
 - As an instructor however, you serve as an oracle -- so I must be prepared to fill any gaps however :)

Where I'll Improve **this Fall 2023**

- Assignments/Labs will of course be refined
- Students will have more video resources
 - YouTube: 67 videos produced in 1 year
 - More videos on ecosystem (e.g. debugging, IDE setup, etc.)
 - I expect over 100 videos by next years DConf.
 - I encourage the D community to turn the camera on as they code, review code, etc.
 - Repeats of the same topics from different perspectives or new use cases are useful for beginners!
 - Learning D will be recommended for course reading alongside Ali's Programming in D
 - **Thank you** to those in D community doing bug fixes, making new tools, and maintaining new bindings to libraries.



https://www.youtube.com/watch?v=HS7X9ERdjM4&list=PLvv_0ScY6vfdf9Fso-3cB4CGnSIW0E4btJV&index=1



Course Postmortem

Things that did and did not matter in choosing D (short list)

- D had ‘rdmd’ for quick prototyping
 - This mattered greatly when introducing D to students -- D was as easy as Python to get started.
- D has better error messages than C++
 - This was very, very important -- huge win for the entire course staff (teaching assistants)
- D has a garbage collector
 - No student (to my knowledge) voiced this as a negative.
 - This was a strict win for me in ramping up students -- many coming from Java this was natural.
 - I showed them malloc/free however
 - Introduced another software engineering trade-off (the whole point of the class in fact!)
- D taught students to work with dependencies (C libraries)
 - This taught students how to work with dependencies and cross language boundaries, potentially building abstractions around them -- I really like this as a lesson in a software engineering course
- Students had to do networking in a project without a prior to a networking course
 - They could figure it out, std.socket a reasonable interface with a lecture and example code.
 - Skill I personally learned at a startup -- you just need to learn stuff as you go

It took great patience....

- But as I watched students proceed in the course, I had to wait until the very end of the semester
 - **They were all excelling far beyond previous semesters.**
- **The worst project using DLang was on par (or better) than the best project in the combined three previous iterations of my course in C++**
 - Why?
 - Students on level playing field in new language
 - DLang overall easier to work with and quicker to iterate (DMD builds fast!)
- There were also unexpected benefits for this specific iteration of students
 - Having to navigate the dlang specs, documentation, read library source code, and go into a ‘new course that was not 100% smooth’ emulated the software engineering experience better than I could have planned.

Why I will continue to teach with DLang at University

DLang at Universities

- DLang is a language that can scale throughout the entire university curriculum
 - I **strongly believe** I could teach a first semester course in DLang
 - (e.g. using rdmd or dmd -run)
 - I **believe** the potential to use DLang in nearly any course in our university curriculum.
 - It's a general-purpose language supporting many paradigms and high-level and low-level facilities.
 - I **choose DLang because** I believe it is a complete package for software engineering
 - Support built into DLang: profiling performance and memory, Documentation (DDoc), unit testing (-unittest), code coverage, package management (dub), are amongst just a few of the features every software engineer should have available and know how to use.

Some Advice to other Faculty Adopting DLang

- **Give students notice on the switch:**
 - I displayed on my webpage several weeks ahead of time (around course registration) that I was switching the language to DLang
 - I think it is good to be up front with students as early as possible
- **Know your university culture:**
 - e.g. I had faculty backing me that learning a new language is an advantage even if you do not use it.
- **Reach Out:**
 - If you're a faculty considering DLang -- reach out to me after watching this talk
- **Let's have Academia + Industry work together (That's you DConf!):**
 - We had 2 students hired from my course specifically for D programming -- there will be new and talented D programmers ready in the Fall!

The next presentation...

- At the end of the semester I brought up DConf to one of the best groups in the course
 - I only told them be honest in their experience using the language
 - They will show you the project that was created
 - And I'm very excited to hand the stage over to them....



A Semester at University: Learning Software Engineering in DLang

Presenters: Andrew Briasco-Stewart, Elizabeth Williams, Ben Mallett, and Steven Abbott

11:30 am-12:00 pm, Wednesday, August 30, 2023

Audience: Everyone in the D community

Your Guides for Today



Andrew
Briasco-Stewart



Elizabeth
Williams



Steven
Abbott



Ben
Mallett



Agenda

- Intro
- Initial Reactions
- Learning in D
- Exciting Features
- Project Overview
- Leveraging Dub
- Project Conclusions
- How much of D did we end up experiencing?
- What did we miss out on?
- Going Forward
- Conclusion

Introduction

- Graduate and undergraduate Northeastern University students.
- Took Mike Shah's Foundations of Software Engineering class in the Spring of 2023.
- Foundations of Software Engineering was our first experience using D.



Disclaimer: We were/are still new to D!

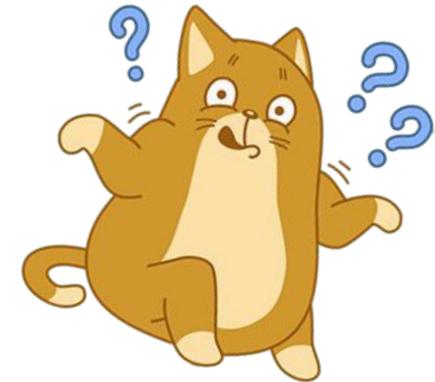


Initial Reactions and Learning D

Initial Reactions

- First thoughts
 - What is D?
 - D resources?
 - Where is it used?
- Hesitations
 - Never heard of D before coming into this class.
 - Realizing that the D community is small.

D



What Does Learning D from Scratch
Look Like for the Average Programmer?

Dlang Tour

Welcome to D



Dlang Tour

Welcome ▾ D's Basics ▾ D's Gems ▾ Multithreading ▾ Vibed ▾ D by Examples ▾ DUB packages ▾

Run ▶ Format ▾ Reset ▾ Export ▾

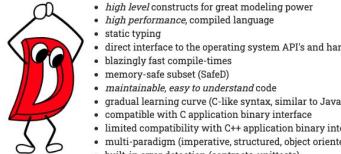
Welcome to D

Welcome to the interactive tour of the *D Programming language*.

The tour gives an overview of this powerful and expressive language which compiles directly to efficient, native machine code.

What is D?

D is the culmination of *decades of experience implementing compilers* for many diverse languages and has a unique set of features:



- *high level* constructs for great modeling power
- *high performance*, compiled language
- *static typing*
- direct interface to the operating system API's and hardware
- blazingly fast compile-times
- memory-safe subset (SafeD)
- *maintainable, easy to understand code*
- gradual learning curve (C-like syntax, similar to Java and others)
- compatible with C application binary interface
- limited compatibility with C++ application binary interface
- multi-paradigm (imperative, structured, object oriented, generic, functional programming purity, and even assembly)
- built-in error detection (contracts, unittests)

... and many more [features](#).

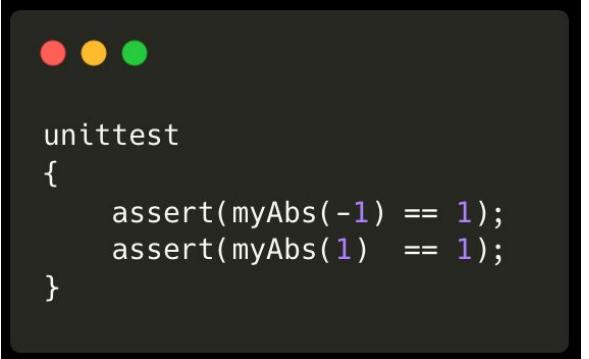
About the tour

Each section comes with a source code example that can be modified and used to experiment with D's language features. Click the run button (or [Ctrl-enter](#)) to compile and run it.

To navigate this tour, either use the '< previous' and 'next >' links at the bottom (or left and right arrow keys), or else go straight to particular sections using the menus at the top.

```
1 import std.stdio;
2 import std.algorithm;
3 import std.range;
4
5 void main()
6 {
7     // Let's get going!
8     writeln("Hello World!");
9
10    // An example for experienced programmers:
11    // Take three arrays, and without allocating
12    // any new memory, sort across all the
13    // arrays inplace.
14    int[] arr1 = {5, 9, 7};
15    int[] arr2 = {5, 2, 1, 10};
16    int[] arr3 = {6, 8, 3};
17    sortInplace(arr1, arr2, arr3);
18    writeln("\n\n\n", arr1, arr2, arr3);
19    // To learn more about this example, see the
20    // "Range algorithms" page under "Gems"
21 }
```

Example 1 - Inline unit tests



unittest

```
{  
    assert(myAbs(-1) == 1);  
    assert(myAbs(1) == 1);  
}
```

Dlang Tour

Example 2 - UFCs, Templates, and Delegates... Oh My!

```
void main()
{
    string text = q{This tour will give
        you an overview of this powerful and
        expressive systems programming
        language which compiles directly
        to efficient, *native* machine code.};

    alias pred = c => canFind(" .\n", c);
    auto words = text.splitter!pred
        .filter!(a => !a.empty);

    auto wordCharCounts = words
        .map!(a => a.count);

    zip(wordCharCounts, words.array())
        .sort().uniq().chunkBy!(a => a[0])
        .map!(chunk => format("%d -> %s",
            chunk[0], chunk[1].map!(a => a[1]))
        .joiner(", ")).joiner("\n").writeln();
}
```

Example 2 - Contract Programming (in/out)

```
long square_root(long x)
in {
    assert(x >= 0);
} out (result) {
    assert((result * result) <= x
        && (result+1) * (result+1) > x);
} do {
    return cast(long)std.math.sqrt(cast(real)x);
}
```

Other Resources We Used



D Programming Language Forum

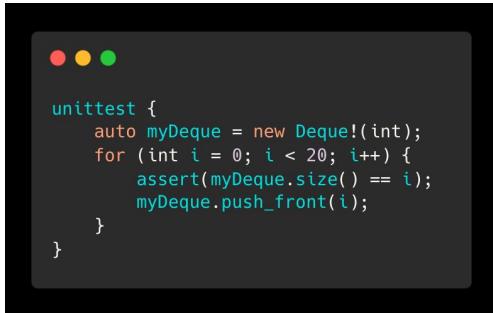


- [Assignment1_dlang](#)
- [Assignment2_DataStructure](#)
- [Assignment3_Debugging](#)
- [Assignment4_SDLC](#)
- [Assignment5_Dub_Patterns](#)
- [Assignment6_Testing](#)

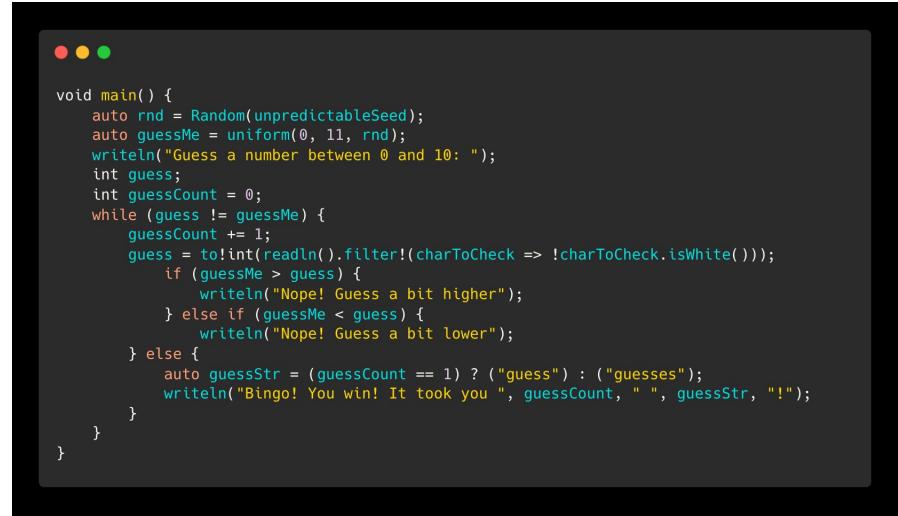
The screenshot shows a YouTube channel page for 'DLang'. The channel has 58 videos and 8,821 views, last updated today. The main video is titled '[Episode 0] Series Teaser matrix.py matrix.d DLang' and shows a man in a blue shirt. Below the video player, there are buttons for 'Play all' and 'Shuffle'. A description below the video reads: 'A full playlist on learning the D Programming language. A great starting place for beginners to start, as we'll start from the very beginning. This playlist will also move towards more advanced features of the language as well - find it all here!' To the right of the main video, there is a list of 6 video thumbnails, each with a title, duration, and view count:

- [Episode 0] Series Teaser matrix.py matrix.d DLang [Dlang Series Teaser] Dlang versus Python speed comparison (Matrix Multiply) Mike Shah • 2K views • 10 months ago
- [Episode 1] Series Teaser matrix.py matrix.d DLang [Dlang versus Python (Matrix Multiply) #shorts series intro] Mike Shah • 1K views • 10 months ago
- [Episode 1] What Is DLang? [Dlang Episode 1] The D Programming Language - dlang Mike Shah • 3.2K views • 10 months ago
- [Episode 2] Dlang Install on Linux [Dlang Episode 2] D Language - setup on Linux (dmd, ldc2, and ldc2 shown!) Mike Shah • 1K views • 10 months ago
- [Episode 3] Dlang Install on Mac [Dlang Episode 3] D Language - setup on Mac (Shown on Mac M1, DMD and LDC2) Mike Shah • 664 views • 10 months ago
- [Episode 4] D Language - DMD command line and Visual D for Visual Studio (DMD and LDC2) [Dlang Episode 4] D Language - DMD command line and Visual D for Visual Studio (DMD and LDC2)

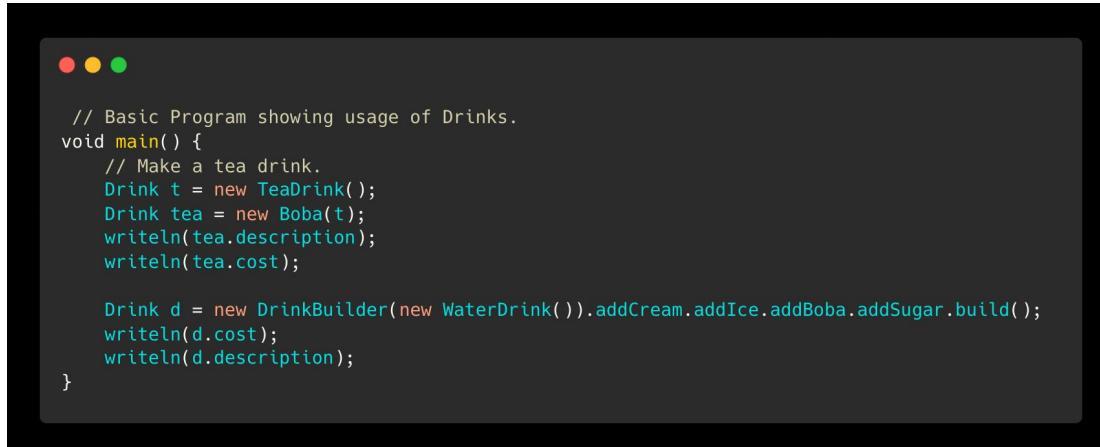
Early Programs in D!



```
unittest {
    auto myDeque = new Deque!(int);
    for (int i = 0; i < 20; i++) {
        assert(myDeque.size() == i);
        myDeque.push_front(i);
    }
}
```



```
void main() {
    auto rnd = Random(unpredictableSeed);
    auto guessMe = uniform(0, 11, rnd);
    writeln("Guess a number between 0 and 10: ");
    int guess;
    int guessCount = 0;
    while (guess != guessMe) {
        guessCount += 1;
        guess = to!int(readln().filter!(charToCheck => !charToCheck.isWhite()));
        if (guessMe > guess) {
            writeln(" Nope! Guess a bit higher");
        } else if (guessMe < guess) {
            writeln(" Nope! Guess a bit lower");
        } else {
            auto guessStr = (guessCount == 1) ? ("guess") : ("guesses");
            writeln(" Bingo! You win! It took you ", guessCount, " ", guessStr, "!");
        }
    }
}
```

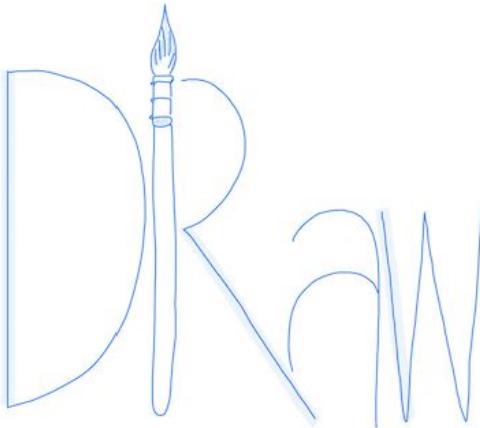


```
// Basic Program showing usage of Drinks.
void main() {
    // Make a tea drink.
    Drink t = new TeaDrink();
    Drink tea = new Boba(t);
    writeln(tea.description);
    writeln(tea.cost);

    Drink d = new DrinkBuilder(new WaterDrink()).addCream.addIce.addBoba.addSugar.build();
    writeln(d.cost);
    writeln(d.description);
}
```

Takeaways From Initial Learning

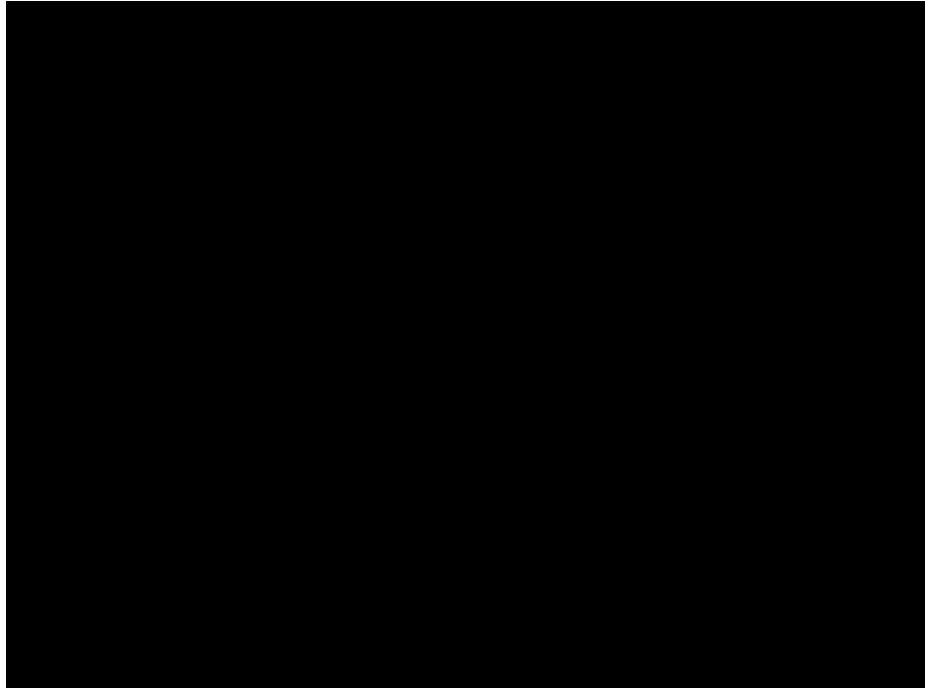
- Growing Excitement.
- Garbage Collection - Easy.
- Memory Safety - Somewhat challenging.
- Several Different Paradigms - Easy.
 - OOP, Imperative, Functional.
- Interop with C - Nice to have.
- Localized Imports, interesting.
- Mixins - complex, but powerful.



Our Project

Project Overview

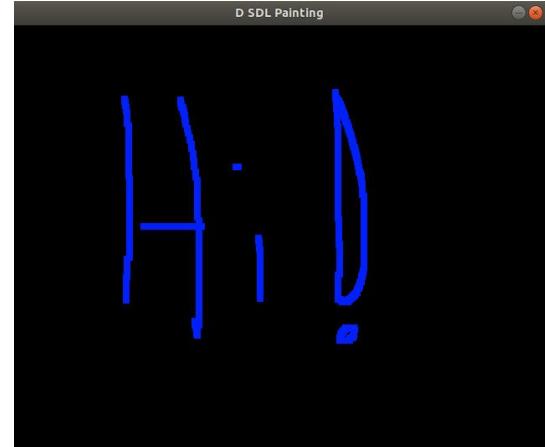
- Networked drawing app.
 - Real-time editing.
- Programmed in D.
- 7-8 Weeks.
- Agile development methodology.
- Simple install and run.



MVP: SDL → GtkD

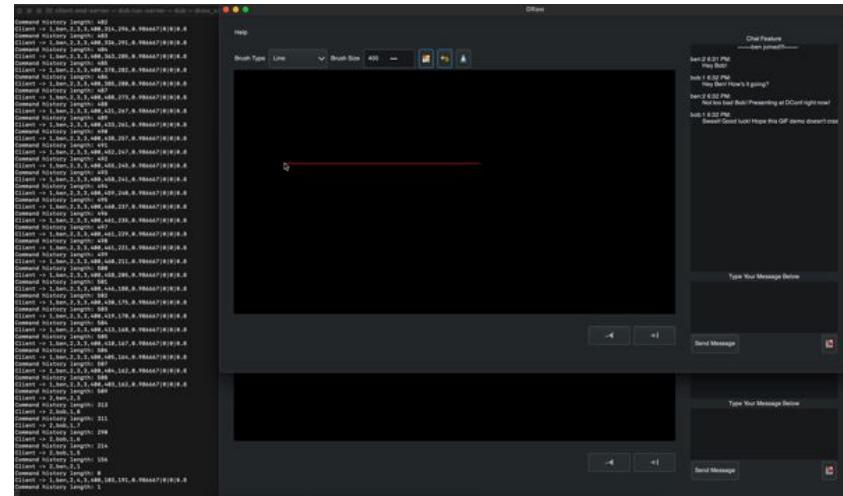


- Started with a single person blank canvas for basic drawing (MVP) in SDL.
- Inability to combine SDL window with GtkD on Macs.
- GtkD has built-in widgets.
- D and GtkD work well with object-oriented programming.
- Learning GtkD was no harder than learning any other new GUI tool.



Networking

- Simple Client-Server Architecture.
- Client renders in one thread, handles packets in another.
- Server leverages SocketSet to efficiently manage connections and incoming packets.
- Networking in D is incredibly simple.
- Message passing and abstracted data structures enable simple, rapid development.





```
char[PACKET_LENGTH] message;
auto recv = 0;
if (this.socketOpen)
{
    recv = this.sock.receive(message);
    //
    // error code checking + handling
    //
    return Tuple!{char[PACKET_LENGTH], long}(message, recv);
}
```

Client Reception



```
for (bool active = true; active && network.isSocketOpen();)
{
    // Checks briefly for messages from main thread.
    auto recv = receiveTimeout(TIMEOUT_DUR, (string packet) {
        // If we get packet info, we will send that along to the server.
        network.sendToServer(packet);
    }, (immutable bool shutdown) {
        // If we receive a shutdown request, we will shutdown this thread.
        active = false;
    }, (OwnerTerminated error) {
        // If our owner thread fails, we will shut down this thread as well.
        active = false;
    });

    // Receives information from the server if there is any.
    auto msgAndLen = network.receiveFromServer();
    // When we receive anything from the server, notify our parent thread.
    if (msgAndLen[1] > 0)
    {
        string encodedMsg = to!string(msgAndLen[0]);
        immutable long recvLen = msgAndLen[1];
        send(parent, encodedMsg[0 .. recvLen], recvLen);
    }
}
```

Message Passing

```
void pollForMessagesAndClients()
{
    if (Socket.select(this.sockSet, null, null))
    {
        if (this.sockSet.isSet(this.sock))
        {
            Socket newSocket = this.sock.accept();
            handleNewConnection(newSocket);
        }

        int[] curKeys = this.connectedClients.keys();

        foreach (key; curKeys)
        {
            Socket client = this.connectedClients[key];
            if (this.sockSet.isSet(client))
            {
                handlePacketReception(client);
            }
        }
    }
}
```

Server via SocketSet

Priority Number 1: Good Development Practices, CI/CD and Dub

CI/CD - What did we want?

1 - Set up a CI/CD system via GitHub actions #3

 Merged ben-mallett merged 39 commits into `master` from `1-set-up-a-cicd-system-via-github-actions` on Mar 22

 Conversation 4

 39

 Checks 6

 Files changed 16



StevenAbbott commented on Mar 21 • edited ▾

Configured Github Actions to:

- Enforce code formatting via dfmt
- Run unit tests and enforce that all unit tests pass
- Generate coverage files and enforce that all files have 100% test coverage (we can lower this later if needed)
- Generate ddocs
- Generate deliverables for windows, ubuntu-20.04, and macOS

How Did We Achieve That?

DUB Setup

- dub.json file for client, server, and client-and-server.
- Helped us manage dependencies:
 - gtk-d
 - unit-threaded
- Simple install and run.



Installing DUB

DUB is the D language's official package manager, providing simple and configurable cross-platform builds. DUB can also generate VisualD and Mono-D package files for easy IDE support.

To install DUB, search your operating system's package manager or [download](#) the pre-compiled package for your platform. The Windows installer will perform all installation steps; for other archives, you will want to ensure the DUB executable is in your path. Installation from source on other platforms is as simple as installing the dmd development files and your system's libcurl-dev, then running `./build.sh` in the repository's folder.

A screenshot of a terminal window with a dark background. It displays a JSON configuration file for DUB. The file defines build types for "docs", which include build requirements like "-allowWarnings", build options like "-syntaxOnly", and flags like "-O-", "-w", "-C", "-Ddocs", and "-op". It also lists excluded source files for both client and server configurations. The "configurations" section defines two configurations: "docs" and "client". The "docs" configuration is set to be an executable targeting both client and server source paths, with excluded source files matching the build type. The "client" configuration is set to be an executable targeting client source only, with excluded source files matching the build type. Finally, the "dependencies" section lists several packages with their required versions: gtk-d:gstreamer (~>3.10.0), gtk-d:gtkd (~>3.10.0), gtk-d:peas (~>3.10.0), gtk-d:svl (~>3.10.0), and gtk-d:vte (~>3.10.0).

```
{
  "authors": [
    "Team DRaw"
  ],
  "buildTypes": {
    "docs": {
      "buildRequirements": [
        "allowWarnings"
      ],
      "buildOptions": [ "syntaxOnly" ],
      "dflags": [
        "-O-",
        "-w",
        "-C",
        "-Ddocs",
        "-op"
      ],
      "excludedSourceFiles": [
        "client/source/app.d",
        "server/source/app.d",
        "client/source/client_cov_app.d",
        "server/source/server_cov_app.d"
      ]
    }
  },
  "configurations": [
    {
      "name": "docs",
      "targetType": "executable",
      "sourcePaths": [
        "client/source",
        "server/source"
      ],
      "excludedSourceFiles": [
        "client/source/app.d",
        "server/source/app.d",
        "client/source/client_cov_app.d",
        "server/source/server_cov_app.d"
      ],
      "dependencies": {
        "gtk-d:gstreamer": "~>3.10.0",
        "gtk-d:gtkd": "~>3.10.0",
        "gtk-d:peas": "~>3.10.0",
        "gtk-d:svl": "~>3.10.0",
        "gtk-d:vte": "~>3.10.0"
      }
    }
  ]
}
```

Unit Testing

```
/*
 * Testing the isValidUsername() method with valid usernames.
 */
@("Testing isValidUsername valid")
unittest
{
    assert(isValidUsername("Mike Shah"));
    assert(isValidUsername("Rohit"));
    assert(isValidUsername("Bob"));
    assert(isValidUsername("User12"));
}

/**
 * Testing the isValidUsername() method with invalid
 usernames.
*/
@("Testing isValidUsername invalid")
unittest
{
    assert(!isValidUsername(null));
    assert(!isValidUsername(" Mike"));
    assert(!isValidUsername(""));
    assert(!isValidUsername(" "));
    assert(!isValidUsername(" d"));
    assert(!isValidUsername("User!!!"));
    assert(!isValidUsername("\n"));
    assert(!isValidUsername("\t\r"));
}
```

```
static bool isValidUsername(string username)
{
    if (username is null)
    {
        return false;
    }

    auto r = regex(r"^-a-zA-Z0-9-()]+(\s+[-a-zA-Z0-9-()]+)*$");
    return !username.equal("") && matchFirst(username, r);
}
```

DUB Packages Documentation About Download Log in

unit-threaded 2.1.7

Advanced multi-threaded unit testing framework with minimal to no boilerplate using built-in unittest blocks

To use this package, run the following command in your project's root directory:

dub add unit-threaded

Manual usage

Put the following dependency into your project's dependences section:

dub.json

"unit-threaded": ">2.1.7"

dub.sdl

dependency "unit-threaded" version=">2.1.7"

Time taken: 9 ms, 273 µs, and 8 hnsecs
24 test(s) run, 0 failed.

OK!

CI/CD - What Worked and What Did Not?

- Code formatting → dfmt
- Unit tests → unit-threaded
- Code coverage threshold → wrote our own script
- Generate docs → harbored-mod
- Generate cross-platform deliverables - GtkD and github actions foiled this

Conclusion

Project Wrap-Up and Closing Remarks

Room For Improvement

DUB Pros and Cons



- Pros 😊
 - Made package management easy.
 - More professional project.
 - Allowed us to create documentation easily.
 - Nice to use – it aided our development.
- Cons 😞
 - Lack of documentation and resources on how to properly set up a project with DUB.
 - Someone should write a coverage-aggregation tool.

Dub Docs - The Bane of My Existence

dub.pm/package-format-json     

Build settings

Build settings influence the command line options passed to the compiler and linker. All settings are optional.

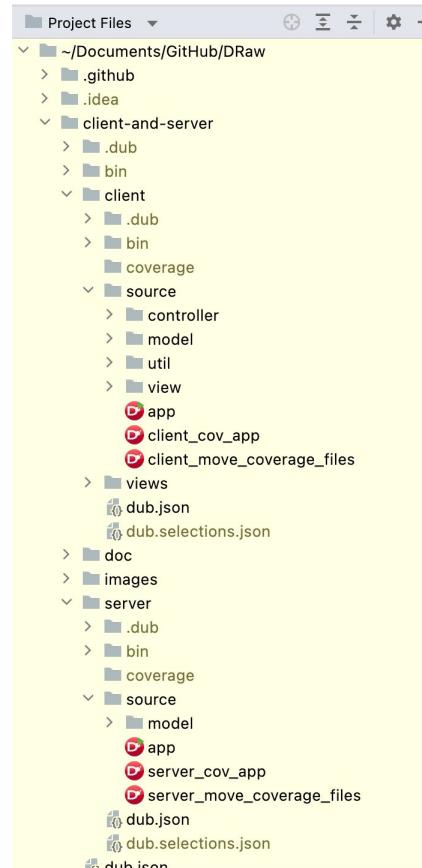
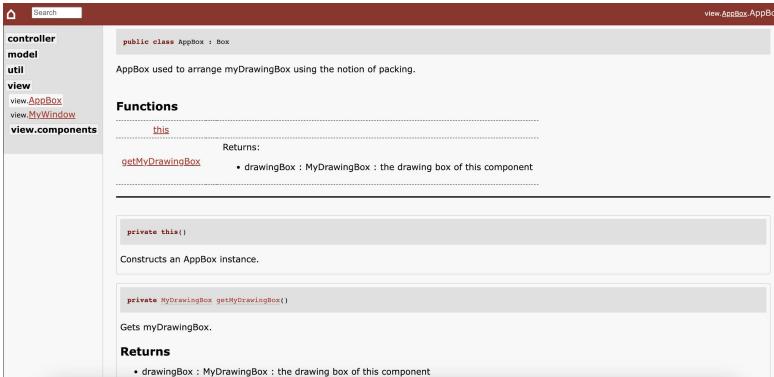
Platform specific settings are supported through the use of field name suffixes. Suffixes are dash separated list of operating system/architecture/compiler identifiers, as defined in the [Language reference](#), but converted to lower case. The order of these suffixes is `os-architecture-compiler`, where any of these parts can be left off. Additionally on Windows the architectures `x86_omf` and `x86_mscoff` can be used with dmd to differentiate between 32 bit object formats used with the `--arch` switch. Examples:

```
{  
    "versions": ["PrintfDebugging"],  
    "dflags-dmd": ["-vtls"],  
    "versions-x86_64": ["UseAmd64Impl"]  
    "libs-posix": ["ssl", "crypto"],  
    "sourceFiles-windows-x86_64-dmd": ["libs/windows-x86_64/mylib.lib"],  
    "sourcefiles-windows-x86_omf-dmd": ["libs/windows-x86_omf/mylib.lib"],  
    "sourceFiles-windows-x86_mscoff-dmd": ["libs/windows-x86_mscoff/mylib.lib"],  
}
```

Name	Type	Description
dependencies	<code>T[string]</code>	List of project dependencies given as pairs of " <code><name></code> : <code><version-spec></code> " - see next section for what version specifications look like - this setting does not support platform suffixes
systemDependencies	<code>string</code>	A textual description of the required system dependencies (external C libraries) required by the package. This will be visible on the registry and will be displayed in case of linker errors - this setting does not support platform suffixes
targetType	<code>string</code>	Specifies a specific target type - this setting does not support platform suffixes
targetName	<code>string</code>	Sets the base name of the output file; type and platform specific pre- and suffixes are added automatically - this setting does not support platform suffixes
targetPath	<code>string</code>	The destination path of the output binary - this setting does not support platform suffixes

Classes and Module System

- Modules based on directories and file/class names.
 - Example: module controller.commands.Command
- Used classes (DrawArcCommand) and abstract classes (Command).



Learning GtkD



gtkDcoding

About Posts in Date Order Blog Posts by Topic

GtkD coding

a site for sore skode

WINDOW

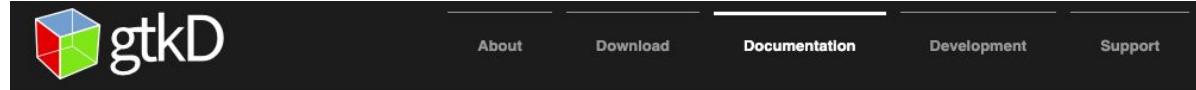
0000: Introduction to GtkCoding
An introduction to GTK 3 and how it can be used to create Graphical User Interfaces (GUI) for applications - a D language tutorial.

Friday, January 11, 2019

WINDOW

0001: Introduction to the Test Rig
Introduction to the GTK Trailing window and how to install syntax highlighter files for PSPad and CodeBlocks - a D language tutorial.

Tuesday, January 15, 2019



Documentation

- [API Reference](#)

Comprehensive API reference documentation for GtkD libraries.

- [Wiki](#)

The public wiki is located on the github project page. Its content is still very limited, but anyone can quickly contribute interesting links or knowledge there.

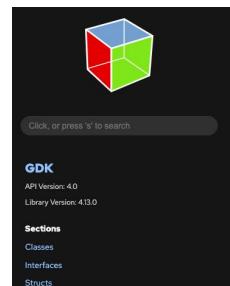
Tutorials

- [gtkDcoding](#)

An ongoing blog series by Ron Tarrant with numerous small tutorials on various gtkD subjects.

- [GExperts](#)

Blog posts about GtkD by Gerald Nunn made while developing the successful [Tilix](#) terminal emulator.

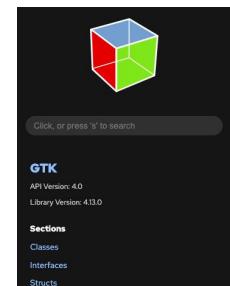


Namespace Gdk - 4.0

The GTK toolkit
Version 4.13.0
Authors GTK Development Team
License LGPL-2.1-or-later
Website <https://www.gtk.org>
Source <https://gitlab.gnome.org/GNOME/gtk/>

Build

C headers gdk/gdk.h
pkg-config files gtk4



Namespace Gtk - 4.0

The GTK toolkit
Version 4.13.0
Authors GTK Development Team
License LGPL-2.1-or-later
Website <https://www.gtk.org>
Source <https://gitlab.gnome.org/GNOME/gtk/>

Build

C headers gtk/gtk.h
pkg-config files gtk4

Lack of Tooling - Coverage Aggregation

Search results for: *coverage*

Package	Latest version	Date	Score	Description
doveralls	1.4.1	2020-Nov-02	2.8	Upload D code coverage results to coveralls.io.
covered	1.0.2	2019-Mar-17	1.6	Processes output of code coverage analysis
liblsparse	1.1.2	2020-Sep-20	0.6	A simple parser for LST coverage files
coverd	1.0.0	2016-Jun-12	0.5	Code coverage HTML reporter for language D listings
d-cobertura	1.0.0	2021-Aug-30	0.4	A program to automatically convert d-style coverage reports to XML cobertura-style ones.
uncovered	0.1.0	2020-Jul-20	0.0	Summary tool for coverage listing files.

Found 6 packages.

What Did We Love?

So How Much of D Did We Actually Touch?

- Delegates
- Functional Programming
- UFCS
- **Message Passing**
- Version Tags
- Parallel (In-built)
- Classes (Networking)
- Library Ecosystem (GtkD, SDL, etc)
- Unit Testing
- **Dub Ecosystem** (Package management, CI/CD, etc)
- **Memory Management**
- ...



Message Passing

- Simplifies code dramatically.
- Promotes safety and SOC.
- Implementation is quick.
- Fewer hungry philosophers.



Dub Ecosystem - The Good

- Great for package management.
- Great for CI/CD utilities.
- Flexible.
- Simple when configured properly.

Welcome to DUB, the D package registry.
Total 2379 packages found.
Select category: —

Most popular

 **silly** 1.20-dev.2
Better test runner for D
by Anton Fediushin

 **ultralight** 0.2.0
Lightweight, high-performance HTML renderer for game and app developers
updated 50 minutes ago

 **vibe-d** 0.9.7
Event driven web and concurrency framework
by Sönke Ludwig

 **vibe-d** 0.9.7
Event driven web and concurrency framework
updated 12 hours ago

 **unit-threaded** 2.17
Advanced multi-threaded unit testing framework with minimal to no boilerplate using built-in
by Attila Neves

 **intel-intrinsics** 1.11.15
Use SIMD intrinsics with Intel syntax, with any D compiler, targetting x86 or arm. Like simde
updated 2 days ago

 **ardsd-official** 0.0.1
Subpackage collection for web, database, terminal ui, gui, scripting, and more with a
by Adam D. Ruppe

 **gettext** 1.0.7
Internationalization compatible with the GNU gettext utilities.
updated 2 days ago

Recently updated

Newly added

 **ultralight** 0.2.0
Lightweight, high-performance HTML renderer for game and app developers
created 50 minutes ago

 **sol-d** 0.1.0
a tiny crypto library
created 2 days ago

 **libfive** 0.1.0
D bindings to libfive, a library and set of tools for solid modeling, especially suited for parametric
created 3 days ago

 **sd** 0.0.2
D library for versatile object serialization/deserialization, supporting integral types, arrays,
created 5 days ago

Memory Management

- Flexibility in when we want to take control of memory management.
- Safer code.
- Almost necessary for beginners.
- Allows me to relax a little.



How Much of D Did We Not Touch?

- @safe, @trusted, @system, ...
- Mixins
- C style &s and *s
- Local imports
- Templates
- Whatever's in DIP1000



Wrap Up

Learning D is an incredibly valuable experience. The language is expressive, powerful, and can be quite fun to use at times.

What's holding D back in our opinion is complexity through inconsistency and lack of clarity.

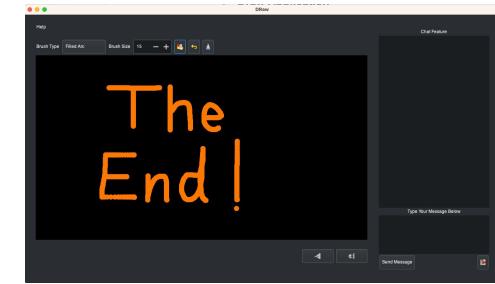
While the language and ecosystem themselves are encouraging, inconsistencies in documentation and a lack of examples for examples sake bring the barrier to entry a bit too high for the casual programmer without a specific reason that makes them want to try the language.

Moving Forward

- Publish examples/toy programs outside of a project context.
- Write articles that are not bound by the forums.
- Update documentation.
- Keep programming in D.

Code For Our Project

- Located here: <https://github.com/abstewart/DRaw>



abstewart / DRaw Public

Code Issues Pull requests Actions Projects Security Insights

master 1 branch 1 tag Go to file Code

abstewart Merge branch 'master' of github.com:abstewart/DRaw	f3b4ec5 on Apr 19	455 commits
.github/workflows	Fix unit tests not running any tests	4 months ago
FinalProject	Update README.md	4 months ago
client-and-server	Merge branch 'master' of github.com:abstewart/DRaw	4 months ago
executables/mac	mac release executables for client and server	4 months ago
generated-docs	adding Adrdox Documentation Generation	4 months ago
media	Added DRaw Whimsical Diagram to the media folder.	5 months ago
support	run_clients for windows	4 months ago
.gitignore	Updated .gitignore.	4 months ago
DEV-README.md	Add documentation about project structure and pull requests, begin i...	5 months ago
README.md	Update README.md	4 months ago

About

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks Report repository

Releases 1

DRaw v1.0.0 Latest on Apr 19

Packages

Help

Chat Feature

Brush Type

Filled Arc



Brush Size

15



thanks!

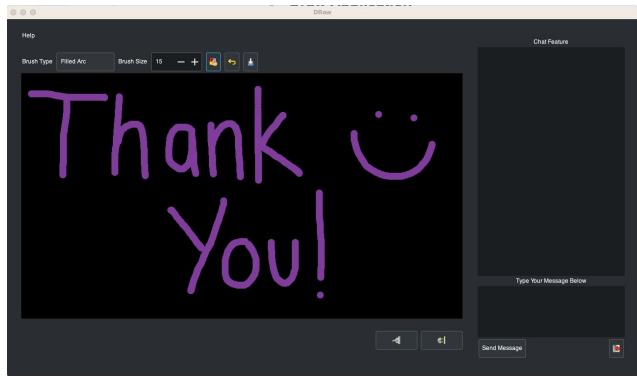
Questions?

Type Your Message Below



Send Message





A Semester at University: Learning Software Engineering in DLang

Presenters: Andrew Briasco-Stewart, Elizabeth Williams, Ben Mallett, and Steven Abbott
11:30 am-12:00 pm, Wednesday, August 30, 2023
Audience: Everyone in the D community