Software Construction and Software Law

Software Engineering Concerns

- **Non-technical issues**
  - Fundraising
- **Technical issues**
  - Security
  - Database of contacts
  - Where to store data
  - UI/UX
  - Recording friends' contact info
  - Network connectivity
  - Deployment
  - Portability

Software Construction Issues

- File systems use (data)
- Scripting (programming)
- Integration
- Configuration
- Testing
- Versioning/evolution
- Low-level debugging (GDB, linking)
- Client-server model

Application Objectives

One should strive for applications that:

- Make the reasonable distinction between what is **persistent** and what is **volatile**
- Are fast
- Are understandable to developers
- Are understandable to users

> - **Persistent** (aka **nonvolatile**): Describing data that is stored in the flash, drive, etc. (secondary storage). This data continues to exist even when the machine loses power through means like encoding itself on magnetic tape.
> - **Volatile**: Describing data that is stored in RAM. This data can be processed very quickly, but because its state is encoded in the circuitry, this data is lost when power is lost.

Software Philosophies

Software Tools Philosophy

Don't write a big program intended to solve all your problems.

Instead, write your application using a collection of tools, each of them relatively simple, and each tailored to solve one class of problems really well.

One can argue that languages like JavaScript fall under this category because you build programs out of smaller modules. JS itself is definitely not a little language, but the design philosophy has users put together small parts of it that individually do its job well to ultimately construct a more complex program.

Little Languages Philosophy

Design small languages appropriate for each tool.

As something grows and grows, it gets too complicated and people can't figure out how it works, and as new applications come out, it becomes less appropriate for that app, so don't *let* your languages scale.

Basically the opposite of C++, a huge language that attempts to solve all classes of problems (*general-purpose programming languages* in general).

Examples: `sh`, `sed`, and `grep` are specialized programs that all come with their own little languages.

- **Downside:** for each tool you want to become an expert in, you have to learn a new language.
- **Upside:** each language is very simple and does its job well.

Legal Issues of Software

Software has 2 roles:

1. A set of instructions for a computer.
2. A way of collaborating to solve problems.

There are 5 major categories of software law:

(1) Copyright

Originally designed for books + text, it gives the creator an *exclusive* right over making copies for some time. Typically it's the lifetime of an author plus about 75 years.

Copyright protects *form*, not the *ideas*. You could create a social media platform similar to Facebook as long as its *details* aren't exactly like it.

Copyright License Types

0. **Public domain**: you have permissions to do whatever you want with the work. When a copyright expires, it goes into public domain.

1. **Academic**: require that you give credit. Classic examples include:

    1. **BSD** (originating from Berkeley)
    2. **MIT** (originating from MIT)

2. **Reciprocal**: require that you be as generous as the original author. A classic example is **GPL**, the GNU Public License.

3. **Corporate**: "sharing but public". Some examples are Apple, Eclipse. You can copy the source code, but if you make changes, you have to give them back to the owner.
4. **Secret**: the source code is not shown. You have the right to run the program and nothing else. This is the most common license in the commercial world.

When building an application, your individual modules may come with a mixture of licenses, so be sure to respect each of them.

(2) Patent

Originally applied to mechanical inventions, then chemical, and in general, *practical, novel* ideas.

Patents cover not just form but also ideas. Gives a monopoly for typically about 7 years.

(3) Trade Secrets

Applies to ideas that you *don't* publish, things a company keeps secret. A popular example is the formula for Coca-Cola!

These are formalized with **nondisclosure agreements (NDAs)**. Industry workers have to sign such agreements before being hired.

> [You should *always*] read the fine print - there's no standard NDA. If someone tells you there's a standard, don't believe it **- Dr. Eggert**

(4) Trademarks

Trademarks aim to avoid confusion among consumers.

You don't want a certain brand name to be confused for another, like a store named McDonald's that has no relation to the official (trademarked) McDonald's.

(5) Personal Data

For example, HIPPA protects the personal information of medical patients.

Enforcement Rules

Infringement

Determining the infringement is a matter of its own.

The *enforcement mechanism* is a **civil suit**. This is **legal protection**, meaning it's slow and expensive.

Technical Protection

Thus, you also want **technical protection**, technical means to *prevent* infringement before it happens. For example:

- You keep the source code secret. That makes it harder for people to copy your system.

- Use SaaS such that the service provider has the software and runs it, with the source code and executables hidden from the end user.
- **Obfuscation** - deliberately use a terribly efficient/mangled version of the executable so it's very hard to reverse-engineer.

Software Licensing

License

- It's NOT a contract. A contract is an *exchange*.
- It's a *grant* of permission.
- It's often *part* of a contract.
- The grant often has strings attached (read the fine print!).

> Good luck on your final, everyone! It's been a journey. **- Vincent**