Colophon:

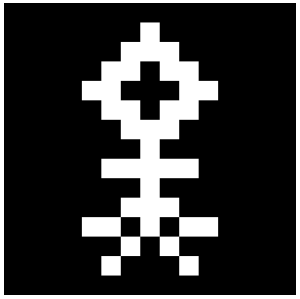
This fanzine was made scraping the Permacomputing principles wiki page: https://permacomputing.net/Principles/

Typefaces: Monospace and Basteleur-moon.

ZineCamp * 2022

permacomputing/ Principles

These **design principles** have been modeled after those of permaculture. These are primarily design/practice principles and not philosophical ones, so feel free to disagree with them or refactor them.



Care for life

This is the ethical basis that permacomputing builds on. It refers to the permacultural principles of "care for the earth" and "care for people", but can be thought of as the basic axiom for all choices.

Create low-power systems that strengthens the biosphere and use the wide-area network sparingly. Minimize the use of artificial energy, fossil fuels and mineral resources. Don't create systems that obfuscate waste.

Everything has a place

Be part of your local energy/matter circulations, ecosystems and cultures. Cherish locality, avoid centralization. Strengthen the local roots of the technology you use and create.

While operating locally and at present, be aware of the entire world-wide context your work takes place in. This includes the historical context several decades to the past and the future. Understanding the past(s) is the key for envisioning the possible futures.

* Nothing is "universal". Even computers, "universal calculators" that can be readapted to any task, are full of quirks that stem from the cultures that created them. Don't take them as the only way things can be, or as the most "rational" or "advanced" way.
* Every system, no matter how ubiquitous or "universal" it is, is only a tiny speckle in a huge ocean of possibilities. Try to understand the entire possibility space in addition to the individual speckles you have concrete experience about.
* **Appreciate diversity**, avoid monoculture. But remember that ?standards also have an important place.
* Strict utilitarianism impoverishes. Uselessness also has an important place, so appreciate it.
* You may also read this principle as: **There is a place of everything.** Nothing is obsolete or irrelevant. Even if they lose their original meaning, programmable systems may be readapted to new purposes they were not originally designed for. Think about technology as a rhizome rather than a "highway of progress and constant obsolescence".
* There is a place for both slow and fast, both gradual and one-shot processes. Don't look at all things through the same glasses.

Respond to changes

Computing systems should adapt to the changes in their operating environments (especially in relation to energy and heat). 24/7 availability of all parts of the system should not be required, and neither should a constant operating performance (e.g. networking speed).

* In a long term, software and hardware systems should not get obsoleted by changing needs and conditions. New software can be written even for old computers, old software can be modifed to respond to new needs, and new devices can be built from old components. Avoid both software rot and retrocomputing.

Care for the chips

Production of new computing hardware consumes a lot of energy and resources. Therefore, we need to **maximize the lifespans** of hardware components — especially microchips, because of their low material ?recyclability.

* Respect the quirks and peculiarities of what already exists and ?repair what can be repaired.
* Create new devices from salvaged components.
* Support local time-sharing within your community in order to avoid buying redundant stuff.
* Push the industry towards **Planned longevity.**
* Design for disassembly.

Keep it small

Small systems are more likely to have small hardware and energy requirements, as well as high understandability. They are easier to understand, manage, ?refactor and ? repurpose.

* Dependencies (including hardware requirements and whatever external software/libraries the program requires) should also be kept low.
* Avoid pseudosimplicity such as user interfaces that hide their operation from the user.
* **Accumulate wisdom and experience rather than codebase.**
* **Low complexity is beautiful.** This is also relevant to e.g. visual media where "high quality" is often thought to stem from high resolutions and large bitrates.
* Human-scale: a reasonable level of complexity for a computing system is that it can be entirely understood by a single person (from the low-level hardware details to the application-level quirks).
* Scalability (upwards) is essential only if there is an actual and justifiable need to scale up; down-scalability may often be more relevant.
* **Abundance thinking.** If the computing capacity feels too limited for anything, you can rethink it from the point of view of abundance (e.g. by taking yourself fifty years back in time): tens of kilobytes of memory, thousands of operations per second — think about all the possibilities!

Expose everything

As an extension of "amplify awareness": Don't hide information!

* Keep everything open, modifiable and flexible.
* Share your source code and design philosophies.
* **State visualization:** Make the computer visualize/auralize its internal state as well as whatever it knows about the state of its physical environment. Regard this visualization/auralization as a background landscape: facilitate observation but don't steal the attention. Also, don't use too much computing resources for this (updating a full-screen background landscape tens of times per second is a total overkill).

Amplify awareness

Computers were invented to assist people in their cognitive processes. "Intelligence amplification" was a good goal, but intelligence may also be used narrowly and blindly. It may therefore be a better idea to amplify awareness.

- Awareness means awareness of whatever is concretely going on in the world/environment but also awareness of how things work and how they situate in their contexts (cultural, historical, biological etc).
- You don't need to twiddle with everything in order to understand it. Yin hacking emphasizes observation.
- It may also often be a good idea to amplify the computer's awareness of its physical surroundings with things like sensors.

Hope for the best, prepare for the worst

It is a good practice to keep everything as resilient and collapse-tolerant as possible even if you don't believe in these scenarios.

- While being resilient and building on a solid ground, be open to positive and utopian possibilities. Experiment with new ideas and have grand visions.
- Design for descent.

Keep it flexible

Flexibility means that a system can be used in a vast array of purposes, including ones it was not primarily designed for. Flexibility complements smallness and simplicity. In an ideal and elegant system, the three factors (smallness, simplicity and flexibility) support each other.

If it is possible to imagine all the possible use cases when designing a system, the design may very well be too simple and/or too inflexible. Smallness, simplicity and flexibility are also part of the "small, sharp tools" ideal of the Unix command line. Here the key to flexibility is the ability to creatively combine small tools that do small, individual things.

- Computing technology in general is very flexible because of its programmability. Programming and programmability should be supported and encouraged everywhere, and artificial lock-ins that prevent (re)programming should be broken.
- Design systems you can gradually modify and improve while running them.

Build on *a* solid ground

It is good to experiment with new ideas, concepts and languages, but depending on them is usually a bad idea. Appreciate mature technologies, clear ideas and well-understood theories when building something that is intended to last.

- Avoid unreliable dependencies, especially as hard (non-optional) dependencies. If you can't avoid them (in case of software), put them available in the same place where you have your program available.
- It is possible to support several target platforms. In case of lasting programs, one of these should be a bedrock platform that does not change and therefore does not cause software rot.
- **Don't take anything for granted.** Especially don't expect the infrastructure such as the power grid and global networking to continue working indefinitely.
- You may also read this as "grow roots to a solid ground". Learn things that last, enrich your local tradition, know the history of everything.