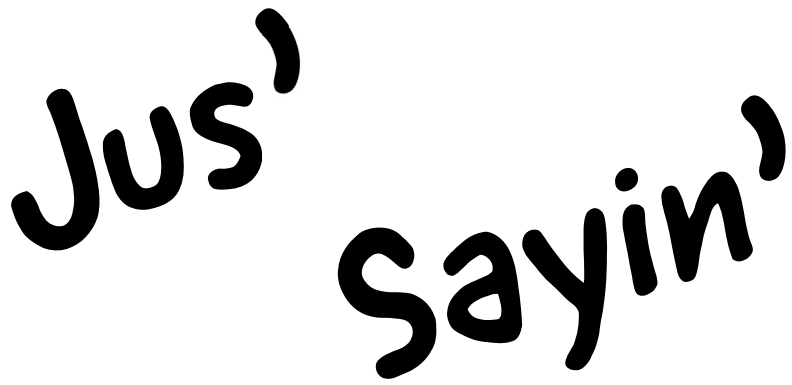
# CREATIVE NUDGES




## UNcommon Sense about Design, Creativity, and Software

Mary Shaw

November 2025

Published by Shaw-Weil Associates



ISBN 978-0-9727324-8-2

The card deck is available at https:// creative-nudges.com where It has its own ISBN.
This booklet and the card deck are version 4.1125
For comments and permissions, contact jus-sayin@creative-nudges.com

André made me do this. Marian backed him up. At ICSE24 in Lisbon, André van der Hoek asked for 25 of "those things you say" that help get a discussion focused. Marian Petre suggested that these could be pithy sayings, insights, precepts, adages, hints, aphorisms, epigrams, slogans, maxims, whatever. I started a list, then exceeded expectations, with closer to 70 than to 25.

Just for fun, I converted the "things" to a deck of cards, to break the constraints of fixed layout and to see how people would rearrange them and play with them.

https://creative-nudges.com

# Jus' Sayin'

*Sometimes ya get stuck in a creative rut and*
*it takes a li'l nudge to get out, jus' sayin'*

These *creativity nudges* capture some of the insights that I offer in technical discussions to remind people to take a fresh point of view, identify dissonance, look for structure, and more. They capture what should be common sense about creativity and design but often isn't recognized as common sense. You can use them in several ways.

The nudges emerged from my decades of experience in software engineering research, so they're largely set in a software context. That research has often drawn on other areas, though, so most of the nudges have broader applicability. In many cases I recall the events that triggered the insight behind the nudge, and these nudges cite the source. In other cases, more recent work has resonated strongly with the nudge, and these are also cited. Sometimes, though, the nudges express opinions that formed gradually over time, so there's some stuff I figured out along the way.

The nudges are clustered by topic (design, engineering, etc). Some are observations about the nature of the world (blue declarative statements), and some are advice on what to do in practice (green imperative statements). Each nudge is elaborated on the back (yellow) with a source and sometimes more detail.

Echoing *Hints for Computer System Design* [Lampson 83], "these are not novel (with a few exceptions), laws of system design, precisely formulated, consistent, always appropriate, approved by all the leading experts, or guaranteed to work. They are just hints". In this vein, these nudges continue a tradition that includes Lampson's *Hints*, the [Perlis *epigrams*], Bentley's *programming pearls* [Bentley 88], and Petre and van der Hoek's *ways experts think* [Experts 16].

Now that I've captured these "things", how can we use them to good advantage? Send ideas to jus-sayin@mary-shaw.org

## How to use these cards

*When you're stuck on a design*

If you get stuck on a creative project, rehashing the same ideas, deal yourself half a dozen cards. Look at each one and ask how it applies to your project, perhaps to suggest a new design alternative or a new way to think about the problem.

*In project meetings*

At the beginning of a discussion, deal several cards to each person. During the discussion, when people get new insights into the topic, they hold up the relevant card and explain the insight. When discussion gets stuck, the leader calls on people for new ideas.

*In open discussion*

Deal three cards to a participant, who then uses them to frame a story about how the nudges could apply in something they've done – perhaps to give insight into why a decision worked out or did not work out. Either distribute the cards before the discussion or give them to the participants just before they start their stories (so that other participants aren't distracted by their own cards).

*In a lessons-learned reflection*

Keep a deck of nudges on the table. As each element that could have been done differently comes up, think about whether it evokes a nudge that would have improved the product or development process.

*As a game*

Choose (or invent) some design or creative scenarios that fit your situation. Deal five cards to each player, then take turns with players playing a card and saying what it suggests about the scenario. The dealer of the round selects the best response, players replenish their hands from the deck, and the winner deals the next round.

*As daily inspiration*

Each day select a card at random and think about it over morning coffee. Or in the evening look for a card that resonates with something that happened during the day. Can you apply the idea to something in your life today? Does it resonate with something you're working on? Is it dissonant with something you believe? Why?

*How things are*

# CREATIVE NUDGES

## Jus' Sayin'

UNcommon sense about design, creativity, and software

Mary Shaw

November 2025

*What to do*

These creativity nudges capture some of the insights Mary Shaw offers in technical discussions to remind people to take a fresh point of view, identify dissonance look for structure, and more.

The nudges are clustered by topic (design, engineering, etc). They distinguish observations about how things are in the world (blue) from advice on what to do (green). Each nudge is elaborated on the back (yellow) with a source and more detail.


version 4.1125
Published by Shaw-Weil Associates
ISBN 978-0-9727324-7-5
http://creative-nudges.com
jus-sayin@mary-shaw.org

## Jus' Sayin'

***Sometimes ya get stuck in a creative rut and it takes a li'l nudge to get out, jus' sayin'***

These cards capture what should be common sense about creativity and design, especially in software, but often isn't recognized as such. You can use them in several ways.

### How to use these cards

***When you're stuck on a design***

If you get stuck on a creative project, rehashing the same ideas, deal yourself half a dozen cards. Look at each one and ask how it applies to your project to suggest a new design alternative or a new way to think about the problem.

***In project meetings***

At the beginning of a discussion, deal five cards to each person. During the discussion, when people get new insights into the topic, they hold up the relevant card and explain the insight. When discussion gets stuck, the leader calls on people for new ideas.

***As a game***

Make up some design or creative scenarios that fit your situation. Deal five cards to each player, then take turns with players playing a card and saying what it suggests about the scenario. The dealer of the round selects the best response, players replenish their hands from the deck, and the winner deals the next round.

Design

**Everyone designs**

"Everyone designs who devises courses of action aimed at changing existing situations into preferred ones."

[Simon 96]

Design

**Everyone creates**

Everyone can conceive of new things. Creativity is not limited to a select few. Technology, including AI, can help people bring their conceptions to fruition.

[Dagstuhl 25]

Design

**No amount of technology can fix a bad idea**

If you're struggling with the implementation, re-examine your goal.

[Harger 25]

Experts are alert to evidence that challenges their theory.

[Experts 16]

Design

**Just because you can, doesn't mean you should**

Consider the implications of your technology.

[variously attributed]

Design

**Problem setting is as important as problem solving**

Simon focused on design as problem-solving. [Simon 96]
Schön focused on design as problem-setting. [Schön 84] [Visser 06]

Design

**Distinguish essential from accidental complexity**

The mental crafting of the conceptual construct of software is the essence; the implementation process is the accident. [Brooks 86]
Experts focus on the essence. [Experts 16]
AI is better at the accident than at the essence.

Design

**Your system probably doesn't have a unique structure or representation**

The alternatives will all be incomplete, and they will interact. [Stuff I figured out]
The physical parts of a motorcycle aren't the same as its functional systems – the same part can contribute to more than one system. [Zen Moto 74]

Design

**It's easier to divide than it is to conquer**

"Divide and conquer" doctrine says to break your problem into smaller parts, solve the parts, and assemble the parts into the solution.
But it's a lot easier to "divide" than to "conquer".
Independently developed parts are often incompatible. [Stuff I figured out]

Design

**Identify the scarce resource and manage it carefully**

In most designs there is a scarce resource.
The key to successful design is to identify this resource and allocate it carefully. [Brooks 10]
Safety first.
In allocating resources, strive to avoid disaster rather than to attain an optimum. [Lampson 83]

Design

**Consider many alternatives**

Experts keep options open. [Experts 16]
Explore the design space broadly; compare high-level alternatives, don't just dive deep on the first idea. [Shaw&Petre 24] [Cross 07]

Design

**Specify "what", not "how"**

Specify the properties, not the realization. Distinguish the model of the problem in the world from the model of the solution in the machine. [Jackson 95]
Distinguish the outer and inner environments; you can only affect the latter. [Simon 96]

Design

**Reframe your problem; reinterpret the situation from a new point of view**

Experts explore different perspectives. [Experts 16]
Rigid adherence to a narrow model or uncritical acceptance of conventional wisdom deters you from exploring a variety of alternatives. Defaulting to a familiar solution can lead to misfit solutions. [Shaw 23]

Design

**Consider affordances**

Consider all the things your materials and resources can contribute, not just their explicit uses. [Norman 88]
Experts do not feel obliged to use things as intended. [Experts 16]

Design

**Import ideas from other fields; adopt and adapt their problems or techniques**

"Have you seen a similar problem? Can you use its method? Can you use its result? What are the unknowns?" etc. [How to Solve It 45]
Look for similar structures in other fields – they may not fit exactly, but they'll make you think. [Shaw 23]

Design

**Look for dissonance between what everyone says and what everyone does**

Opportunities lie in this gap. [Stuff I figured out] [Shaw 23]
When the terrain disagrees with the map, trust the terrain. [Swiss Army Proverb]

Design

**Question authority; rethink assumptions**

Experts generate alternatives. [Experts 16]
"sometimes he feels that there really is another way, if only he could stop and think of it." [Edward Bear, bumping down the stairs, Pooh 26]
[Timothy Leary, maybe Socrates]

Design

**Take different perspectives intentionally and contrast them critically**

Look for contrary as well as confirming evidence. Look for structural resonance or dissonance between problems and solutions. Alternate between expanding the design space and limiting attention to essential details. [Petre&Shaw 25]

Design

**There are two kinds of change: anticipated and unanticipated. Use parameters for the former, use editors for the latter**

Parameters capture the designer's intentions for anticipated variability.
[Butler Lampson]

Engineering

**No amount of tool, process, or method can replace actually understanding the problem and caring about its solution**

Engineering is about how to make the important decisions. Tools, processes, and methods (including AI) can make this more efficient but they do not replace thinking.
[Stuff I figured out]
Neither abstraction nor simplicity is a substitute for getting it right.
[Lampson 83]

Engineering

**The prototype is not the product**

Prototypes are built to explore particular design questions; they are usually not suitable as the primary basis for a product.
[Stuff I figured out]
Plan to throw one away.
[Brooks 95]
*cf* vibe coding

Engineering

**"It depends"**

There isn't a single correct answer, it depends on the client, the context, ...
[Stuff I figured out]

Engineering

**Correctness is a social construct**

Other than formal verification of formally specified code in a sound language, the standard for correctness is negotiated among the participants.
[Lakatos 76] [Proof 25]
[DeMillo&Lipton&Perlis 79]

Engineering

**Consensus is not correctness**

The blind leading the blind is not a substitute for adequate definition and documentation. Crowdsourcing and Q/A communities provide suggestions, but their consensus might be wrong. [Stuff I figured out]

Engineering

**Time is not money**

Costs can be tracked in time, space, attention, bandwidth, etc. Their measurement properties differ (fungibility, rivalry, perishability, ratio vs ordinal scale, ...). Converting everything to money is convenient, but it loses nuance and may introduce errors.
[Poladian&al 03]

Engineering

**You can't get security into a system by sprinkling it like pixie dust on your completed code**

... nor privacy, nor reliability, nor usability, nor performance, nor ... These properties must be designed into the structure and fabric of the system, into its very architecture.
[Stuff I figured out]
Retrofitting reliability to an existing design is very difficult. [Lampson 83]

Engineering

**Moving from a program to a product increases the complexity and effort by a factor of 3; so does moving from a program to a system**

Doing both, moving from a program to a system product, costs a factor of 9 – about an order of magnitude.
[Brooks 95]
Each step has higher demands for reliability, security, usability, correctness, safety, performance, and other quality attributes.
*cf* vibe coding

Engineering

**Less than 10% of the code goes to the overt function of the system; the rest goes to housekeeping**

... such as system code, I/O, standard dialogs, UI, data validation, audit trails, communication, and so on.
[Stuff I figured out] [Shaw 95]
[Bentley 88] [many places online]
We can hope that AI will take over a lot of the remaining 90%.

Engineering

**Seek cost-effective solutions to practical problems**

Engineering creates cost-effective solutions to practical problems by applying systematic knowledge to building things, in the service of mankind. It entails decisions constrained by limited knowledge, time, and resources.
[Shaw 90] [Manifesto 05]

Engineering

**Satisfice – fitness for the task at hand is often a better goal than formally verified correctness**

There's a spectrum from "good enough" to "must be correct", choose the right point. [Stuff I figured out] [Shaw 20]
This is a form of satisficing. [Simon 96]

Engineering

**Allow the engineer to be utterly faithful to the engineering integrity of the engineered artifact**

Many project failures arise from management getting in the way. [Squires 86]

Engineering

**Think, don't just count**

Evaluate things critically and thoughtfully. Counting things is often a poor proxy for what you really care about. [Stuff I figured out]
*cf* bibliometrics

Engineering

**Measure what matters**

If you can't measure your true objective directly, beware of proxy measures. "When a measure becomes a target, it ceases to be a good measure" – people will pursue the target (instead of the real goal, if they are different). [Goodhart's Law]
*cf* the H-index

Engineering

**Don't confuse ONE way with THE way**

There's a difference between ∃ and ∀. Many methods, tools, and processes are sometimes useful, and it's usually good for the whole team to be on the same page. But no method, tool, or process is uniquely the best. [Stuff I figured out]

Engineering

**Reuse proven solutions when you can**

In engineering, *routine design* selects established solutions to well-known problems. *Innovative design* finds novel solutions to unfamiliar problems. Choose routine engineering when it works for you. [Shaw 90] Experts prefer solutions that they know work. [Experts 16]

Engineering

**Make deliberate tradeoffs among resources and among quality attributes**

Experts make tradeoffs.
[Experts 16]
Pay special attention to the generality/power tradeoff.
"Faster, better, cheaper, choose two"?
No, choose some of each. [Stuff I figured out]

Engineering

**Exhaust less drastic, reversible options before turning to more drastic, irreversible ones**

Try physical therapy before surgery.
[Stuff I figured out]

Engineering

**Satirize: ridicule absurdity**

Humor, especially satire, can be a safe way to say things that are otherwise unacceptable.
[medieval court jesters]

Systems and Society

**Problems embedded in society are not often solved using the ordinary methods of analysis**

Many stakeholders, competing implicit goals, and irreversible actions rule out clear problem definitions and success criteria. These are *Wicked Problems*.
[Wicked 73]

Systems and Society

**New technology doesn't necessarily invalidate old theory, though it may require adaptation**

Don't chase fads.
[Stuff I figured out]
The information economy was said to need a whole new economic theory. But no, the old economics works just fine, just with new parameters.
[Info Rules 99]
AI will not make SE obsolete. [Kang&Shaw 24]

Systems and Society

**There are two ways to deal with the possibility of Bad Things: prevention and remediation**

Choose prevention when consequences are dire – remediation is often preferable the rest of the time.
[Stuff I figured out] [Shaw 06]

Systems and Society

**Correlation is not causation**

Doesn't everyone know this?

Systems and Society

**Automate things that are well-understood, tedious, and error-prone when left to humans**

... and where the consequence of failure is manageable. Everything else, plus oversight, is human responsibility.

[Stuff I figured out]

[Kang&Shaw 24]

Systems and Society

**Be reasonable**

Assume everyone is reasonable until proven otherwise, and they are open to being reminded of this. Assume unusual action is probably reasonable. Favor personal discussion over administrative action. This is the *Reasonable Person Principle*.

[long CMU CS tradition]

Systems and Society

**Align rights with responsibilities**

Holders of rights, privileges, and authority incur a corresponding obligation to be responsible for the outcomes of exercising those rights.

[Stuff I figured out]

Systems and Society

**Don't try to fix specific problems with sweeping policy changes – deal directly with the specific problem instead**

The policy changes will have troublesome side effects, and they will impose new burdens on the non-problems.

[Stuff I figured out]

Systems and Society

**Participate in early design phases**

Major changes get harder as design progresses. Speak up early. Stay engaged, lest your concerns get eclipsed. Waiting for the public meetings is often too late.

[Stuff I learned in public advocacy]

Research

**Asking the right question is as important as getting the right answer**

[Stuff I figured out, or maybe Jim Horning]

Research

**Computer science is the study of the phenomena surrounding computers**

This is the "big-tent" inclusive view of computer science. The phenomena surrounding computers are rich and complex. There is, as with many sciences, a related engineering discipline. [Newell&Perlis&Simon 67]

Research

**There are three nested classes of results: findings** (fully validated research), **observations** (reports of facts), **& rules-of-thumb** (generalizations, even unvalidated, believed by investigators willing to sign them)

The appropriate criteria for quality should be *truthfulness and rigor* for findings; *interestingness* for observations; *usefulness* for rules-of-thumb; and *freshness* for all three. [Brooks 88]
These cards are rules-of-thumb. [Stuff I figured out]

Research

**Concentrate on doing great science**

The rest will take care of itself. [Allen Newell]

Research

**Understand your own research strengths; choose projects where this gives you a comparative advantage**

If you're a bit better than most at a particular kind of analysis or a specific domain, choose research topics where you can use that. [Herb Simon]

Research

**Understand the paradigms of your field**

Disciplines can be distinguished by the kinds of questions they ask, the kinds of answers they accept, and their methods to find and validate answers.
Software engineering does not explain this very well. [Shaw 03]
[cf Schön 84 pp 191ff]

Research

**Establish traceability between your claims and your results**

Show how *all* elements of your claim about the research are supported by the results you're reporting and that you properly applied your methods to get the results. Don't include irrelevant results.
[Stuff I figured out]

Abstraction / Specification

**Progress in programming languages can be tracked by the increasing size of programming abstractions**

As time passes, each line of code invokes more power.
[Shaw 80] [Shaw&Klein&Ross 25]
This is why "lines of code" was always a terrible measure of productivity: it measured input, not output.
[Shaw 02]

Abstraction / Specification

**Abstraction is suppressing details; good abstraction is suppressing the right details**

Keep secrets.
[Lampson 83]
Experts design elegant abstractions.
[Experts 16]
The key to creating a powerful abstraction is identifying the domain-related concept that is lurking in the implementation details.
[Stuff I figured out]

Abstraction / Specification

**Programming to a specification only works if you know in advance what you want**

... including the extra-functional properties (aka quality attributes). Exploratory programming uses early implementations to figure out what you actually want. [Shaw 20]

Abstraction / Specification

**The real value of formal specification and verification is taking a hard look at the code from two very different points of view**

In the early days of program verification we were debugging the specification as much as the code. [Stuff I figured out]

Abstraction / Specification

**Specifications are not formal, static, and complete – they are heterogeneous, evolving, and incomplete**

They capture the best, most appropriate information available at a given time. We call these *credentials*.
[Shaw 96] [Scaffidi&Shaw 07]

Abstraction / Specification

**Seek systematic underlying structure**

Classification systems or taxonomies can be principled (top-down) or ad hoc (bottom-up). Both will capture domain structure; both are affected by cultural norms and politics. Both will have buckets for "other stuff that doesn't fit."

[Bowker&Star 99] [Shaw 23]

Abstraction / Specification

**Generalize just enough, but not too much**

Generalization makes your model useful in more cases, but it often sacrifices power for specific cases.

[Stuff I figured out]

Everything should be made as simple as possible, but no simpler.

[attr Einstein]

Abstraction / Specification

**Define the correspondence between a new abstraction layer and its implementation carefully**

You must show that data and operations at the abstract level are faithfully represented in the implementation.

[Hoare 72]

Abstraction / Specification

**Don't assume that formal specifications can capture everything you care about**

The walled-garden tragedy of formal specifications: they support, in principle, verifying things within the formal system, but they are inherently unable to specify things outside that system. You must find other ways to specify the rest.

[Stuff I figured out]

[Kang&Shaw 24]

Abstraction / Specification

**Don't assume programmers are highly-trained professionals with math skills**

Vernacular programmers vastly outnumber highly-trained professionals. They are poorly served by the software and programming languages communities. [Shaw 20]

[Scaffidi&Shaw&Myers 05]

Abstraction / Specification

**Distinguish carefully among the thing, the name of the thing, what the thing is called, and what the thing is**

Recall from *Alice* the Knight's Song, and the name of the song, what the name of the song is called, what the song is called, and what the song is. [Alice 60]

Education

**In curriculum design, the scarce resource is curriculum space**

Manage it carefully – curriculum design is at heart a resource allocation problem.
[Stuff I figured out] [Manifesto 05]

Education

**Curricula, as systems, should provide traceability from the overall objectives to the details**

... certainly to the course objectives (and thence to the course content, of course) and also to other activities. This should show how overall curriculum objectives are achieved and how all required activities contribute.
[Stuff I figured out]

Education

**Teach them to think like engineers, to learn on their own, to be informed citizens**

... plus enough current practice to get started.
[Carnegie Plan]

Education

**Make the student responsible – there are two sides to the lectern**

"Learning results from what the student does and thinks, and only from what the student does and thinks. The teacher can advance learning only by influencing the student to learn." [Simon memorial]

Education

**Help students read and study good examples before asking them to write their own**

Students in other fields study and critique good exemplars before creating their own. They study not just the exemplars' appearance but also their structure, their meaning, and critiquing techniques. Programmers must likewise read and evaluate code written by others, including AI.
[Shaw&Hilton&Fairbanks 26]

Education

**Teach enduring principles in the context of current competence**

These foundations provide a base for learning new technologies as they emerge.
[Manifesto 05] [Carnegie Plan]

## References to Books


[Alice 60] Martin Gardner (annotating Lewis Carroll). The Annotated Alice. Clarkson Potter 1960.

[Bentley 88] Jon Louis Bentley. *More Programming Pearls: Confessions of a Coder*. Addison Wesley 1988.

[Bowker&Star 99] Geoffrey C. Bowker and Susan Leigh Star. *Sorting Things Out: Classification and its Consequences*. MIT Press 1999. ISBN 978-0262024617

[Brooks 95] Frederick P. Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Second edition, Addison Wesley 1995.

[Brooks 10] Frederick P. Brooks Jr. *The Design of Design*. Addison Wesley 2010. ISBN 978-0201362985

[Cross 07] Nigel Cross. *Designerly Ways of Knowing*. Board of International Research in Design. Birkhäuser Architecture 2007.

[Experts 16] Marian Petre and André van der Hoek. *Software Design Decoded: 66 Ways Experts Think*. MIT Press 2016. ISBN 978-0262035187

[How to Solve It 45] George Polya. *How To Solve It.* Princeton University Press 1945.

[Info Rules 99] Carl Shapiro and Hal R. Varian. 1999. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business Review Press, 1999. ISBN 978-0875848631

[Lakatos 76] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press 1976.

[Norman 88] Donald Norman. *The Psychology of Everyday Things*. Basic Books 1988.

[Pooh 26] A. A. Milne. *Winnie-the-Pooh*. Dutton, 1926.

[Proof 25] Adam Kucharski. Proof; *The Art and Science of Certainty*. Basic Books 2025.

[Schön 84] Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books (1984). ISBN 978-0465068784.

[Simon 96] Herbert A. Simon. 1996. *The Sciences of the Artificial*. MIT Press 1996. ISBN 978-0262691918.

[Squires 86] Arthur M. Squires. *The Tender Ship: Governmental Management of Technological Change*. Birkhauser 1986. ISBN 978-0817633127

[Visser 06] Willemien Visser. 2006. *The Cognitive Artifacts of Designing*. CRC Press 2006. ISBN 978-0805855111.

[Zen Moto 74] Robert Pirsig. *Zen and the Art of Motorcycle Maintenance*. Bantam 1973 ISBN 0-553-10310-5

## References to Papers


[Brooks 86] Frederick P. Brooks Jr. No silver bullet: Essence and accident in software engineering. *Proc IFIP Tenth World Computer Conf*, pp 1069-1076. Reprinted in [Brooks 95]

[Brooks 88] Frederick P. Brooks, Jr. Grasping Reality Through Illusion—Interactive Graphics Serving Science. *Proc 1988 ACM SIGCHI Human Factors in Computer Systems Conf (CHI '88)* pp. 1-11.

[DeMillo&Lipton&Perlis 79] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Commun. ACM* 22, 5 (May 1979), 271–280. https://doi.org/10.1145/359104.359106

[Hoare 72] C Anthony Hoare. Proof of Correctness of Data Structures. *Acta Informatica* 1972.

[Jackson 95] Michael Jackson, The World and the Machine. *ICSE '95.*

[Kang&Shaw 24] Eunsuk Kang and Mary Shaw. tl;dr: Chill, y'all – AI will not devour SE. *Onward! Essays*, 2024.

[Lampson 83] Butler W. Lampson. Hints for computer system design. In Proceedings of the ninth ACM symposium on Operating systems principles (SOSP '83),1983.. 33–48. https://doi.org/10.1145/800217.806614

[Manifesto 05] Mary Shaw (ed). Software Engineering for the 21st Century: A basis for rethinking the curriculum. *Technical Report CMU-ISRI-05-108*, Carnegie Mellon University, Institute for Software Research International, March 2005.

[Newell&Perlis&Simon 67] Allen Newell, Alan J. Perlis, and Herbert A. Simon. What Is Computer Science? Science 1967(157) 1373-74. doi: 10.1126/science.157.3795.1373.c

[Petre&Shaw 25] Marian Petre and Mary Shaw. Contrasting to Spark Creativity in Software Development: Tactics Used By High-Performing Teams, *IEEE Software*, vol. 42, no. 3, pp. 67-74, May-June 2025, doi: 10.1109/MS.2025.3538670.

[Poladian&al 03] Vahe Poladian, Shawn A. Butler, Mary Shaw, and David Garlan. Time is Not Money: The Case for Multi-dimensional Accounting in Value-based Software Engineering. Position paper for *Fifth Workshop on Economics-Driven Software Research (EDSER-5),* affiliated with the 25th *Int'l Conf on Software Engineering*, 2003. doi: 10.1184/R1/6626234.v1

[Scaffidi&Shaw 07] Christopher Scaffidi and Mary Shaw. Developing confidence in software through credentials and low-ceremony evidence. Presented at workshop, not in digital library. Online at NON-ARCHIVAL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.407.4357&rep=rep1&type=pdf (accessed December 4, 2021).

[Scaffidi&Shaw&Myers 05] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the numbers of end users and end user programmers. *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '05),* 2005, pp. 207–214, https://doi.org/10.1109/VLHCC.2005.34

[Shaw 80] Mary Shaw. The Impact of Abstraction Concerns on Modern Programming Languages.*Proc IEEE special issue on Software Engineering* 68, 9 (1980), 1119–1130. https://doi.org/10.1109/PROC.1980.11811 (invited).

[Shaw 90] Mary Shaw. Prospects for an Engineering Discipline of Software. *IEEE Software* 7, 6 (November 1990), 15–24. https://doi.org/10.1109/52.60586

[Shaw 95] Mary Shaw. Architectural issues in software reuse: it's not just the functionality, it's the packaging. *SIGSOFT Softw. Eng. Notes* 20, SI (Aug. 1995), 3–6. https://doi.org/10.1145/223427.211783

[Shaw 96] Mary Shaw. Truth vs Knowledge: the difference between what a component does and what we know it does. In *Proc. of the 8th International Workshop on Software Specification and Design (IWSSD).* https://doi.org/10.1109/IWSSD.1996.501165

[Shaw 02] Mary Shaw. The Tyranny of Transistors: What Counts about Software? *Proc Fourth Workshop on Economics Driven Software Engineering Research*, 2002. Nonarchival workshop presentation. Archived at https://web.archive.org/web/20170721211912/https://www.cs.cmu.edu/~Compose/ftp/shaw-sw-measures-fin.pdf

[Shaw 03] Mary Shaw. Writing Good Software Engineering Research Papers. Mini-tutorial for ICSE 2003, *Proc. Int'l Conf on Software Engineering (ICSE-2003),* Portland OR, May 2003. doi: 10.1109/ICSE.2003.1201262

[Shaw 06] Mary Shaw. Strategies for Achieving Robustness in Coalitions of Systems. In *NATO Workshop RTO IST-064, Building Robust Systems with Fallible Construction*. https://www.sto.nato.int/publications/STO%20Meeting%20Proceedings/RTO-MP-IST-064/$$MP-IST-064-ALL.pdf

[Shaw 20] Mary Shaw. Myths and Mythconceptions: What Does it Mean to be a Programming Language, Anyhow?. In HOPL IV, Fourth ACM SIGPLAN History of Programming Languages Conference (*Proc. ACM Program. Lang.*, Vol. 4). 44. https://doi.org/10.1145/3480947

[Shaw 23] Mary Shaw. Aha! Strategies for Gaining Insights into Software Design**.** Workshopped at PLoP23: Pattern Languages of Programs Conference, October 2023. Revised version *Proc. PLoP 2023: Proc 30th Conf on Pattern Languages of Programs*, ACM, 2023 https://dl.acm.org/doi/10.5555/3721041.3721043

[Shaw&Hilton&Fairbanks 26] Mary Shaw, Michael Hilton, and George Fairbanks. AI Tools Make Design Skills More Important than Ever (The Pragmatic Designer Column). *IEEE Software*, Jan/Feb 2026, Vol 43, no 1

[Shaw&Klein&Ross 25] Mary Shaw, Daniel V. Klein and Theodore L. Ross. Revisiting Abstractions for Software Architecture and Tools to Support Them," *IEEE Transactions on Software Engineering*, vol. 51, no. 3, pp. 768-773, March 2025, doi: 10.1109/TSE.2025.3533549

[Shaw&Petre 24] Mary Shaw and Marian Petre. Design spaces and how software designers use them: a sampler. In *Designing '24: 2024 International Workshop on Designing Software Proceedings*. 8 pp. https://doi.org/10.1145/3643660.3643941

[Wicked 73] Horst W. J. Rittel and Melvin M. Webber. Dilemmas in a general theory of planning. *Policy Sciences* 4, 155–169 (1973). https://doi.org/10.1007/BF01405730

## References to Other Sources


[Carnegie Plan] Carnegie Plan for Education. See, e.g, the last page of [Manifesto 05]

[Dagstuhl 25] Dagstuhl Perspectives Workshop on Creativity, GenAI, and Software Development, October 5-10, 2025. From discussions.

[Goodhart's Law] Charles Goodhart. Problems of Monetary Management: The UK Experience. *Papers in monetary economics* 1975; 1; 1. - [Sydney]. - 1975, p. 1-20. Vol. 1. Sydney: Reserve Bank of Australia. Usually paraphrased as "When a measure becomes a target, it ceases to be a good measure".

[Harger 25] Brenda Bakker Harger, of CMU Entertainment Technology Center, in lecture at CMU 3/26/25

[Shaw 25 cards] Mary Shaw. Jus' Sayin' : UNCommon sense about design, creativity, and software (cards). https:/creative-nudges.com version 4.1125j, Shaw-Weil Associates, 2025.

[Simon memorial] Quote and report on creation of the memorial at http://doi.library.cmu.edu/10.1184/pmc/CMM/CMM_2002_021_01_2002.

[Stuff I figured out] (and variants of this) I can't figure out who else to attribute this to, so I apparently figured it out for myself, and sometimes I wrote it down

Jim Horning, Butler Lampson, Allen Newell, Herb Simon, I recall these insights from meetings but can't put a date or citation on them.

Various other (possibly folkloric) attributions: CMU tradition, Swiss Army, Timothy Leary, court jesters, Albert Einstein, Socrates. online



ISBN 978-0-9727324-8-2

9 780972 732482