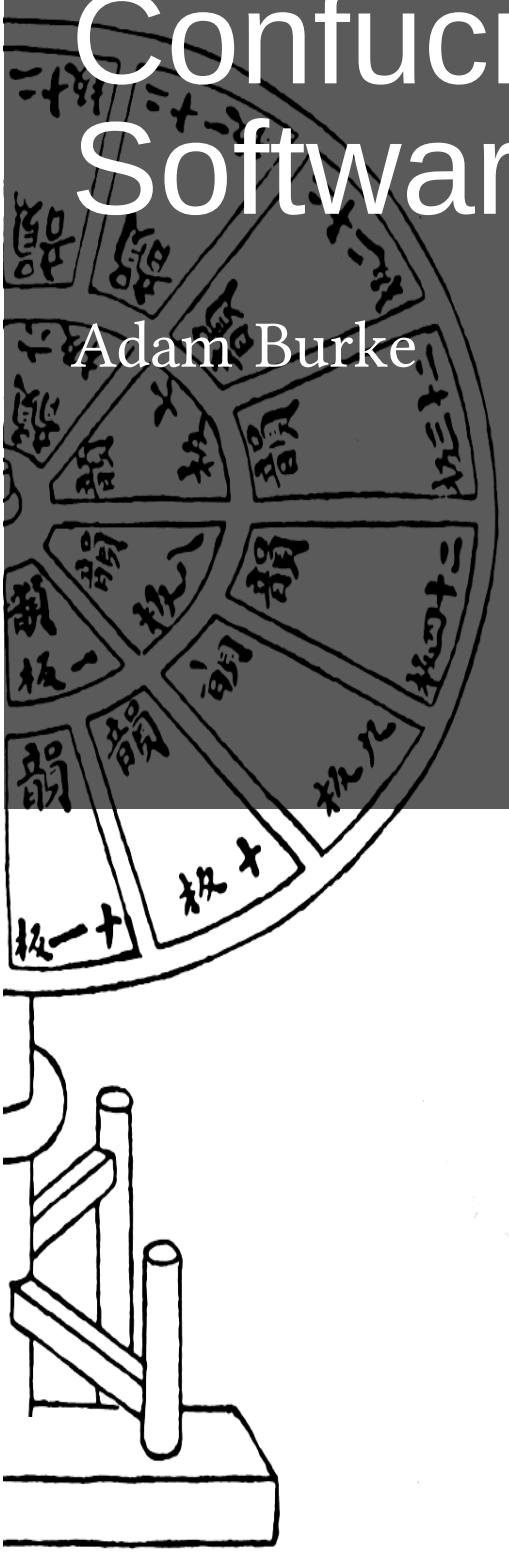
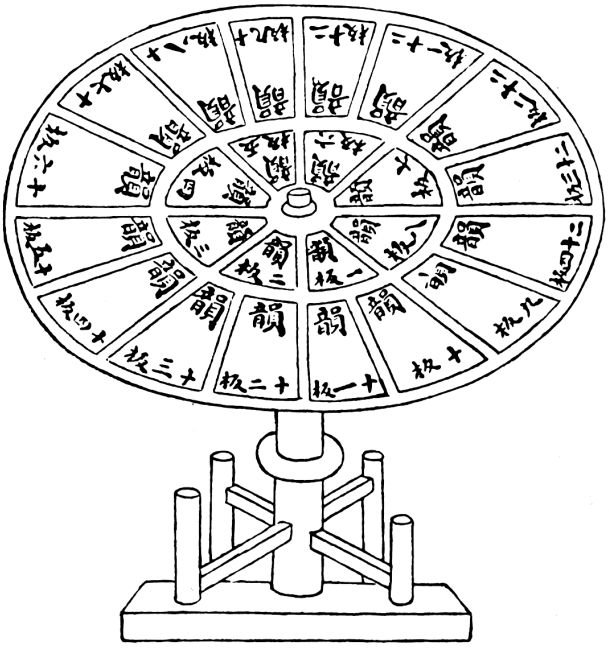
# Confucian Software

Adam Burke





Frontispiece: Wang Zhen's revolving typecase, from the *Nong Shu* (1313 CE) (Todd 2007).

# Confucian Software

Adam Burke

**Confucian Software**
Adam Burke
2026
Calyx Create Group
Version 1.0



ISBN: 978-0-6489750-3-8

Cover image and frontispiece: Wang Zhen's revolving typecase, from the *Nong Shu* (1313 CE) (Todd 2007). Public domain.

Trailer image (after the table of contents): Su Sung's astronomical clock tower, K'ai-feng, ca. 1090 CE (Christiansen 1959). No known copyright restrictions.

Exit image (end of the main chapters): plate illustrating *A Bao* and *Frog Orchestra*, from *Xiangzhu Liaozhai zhiyi tuyong* (Pu 1915). Public domain.

# Contents

# Foreword to the First Edition

I've long felt the conceptual links between software and philosophy were much stronger than the culture of either philosophers or software would normally reveal. And Confucius has been a personal inspiration since I first read an English translation of *The Analects* in my early twenties. I wrote these essayistic blog posts over a period of several years while working as a corporate software engineer. I was living in a country often called Confucian, as well as mercantile, and clean, and kiasu: the beautiful city-state of Singapore.

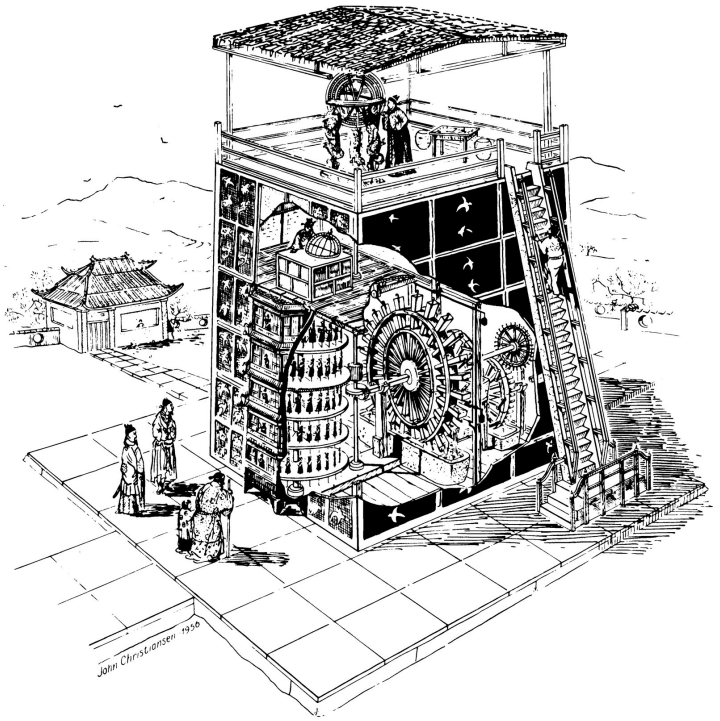
Looking back a decade later, and still drawing on links like name-oriented software in my professional work, it seemed worth binding the project into a clear, versioned artefact. Though I always tried to ground my thoughts in Chinese philosophy and history, I am not a specialist historian. This is a work of speculative design criticism. Certain throughlines from old Chinese problems of information and control have become more visible, in ways that reward the speculation, now that China is once again a great technological power. It's like the link from Wang Zhen's movable type selector (Todd 2007), invented two centuries before Gutenberg, through to the Chinese typewriter and predictive text on mobile phones (Mullaney 2017).

So here's the book. One framing essay and seven commentary essays, with the original in Chinese and English. Where other verses from the *Analects* are cited, they are included for the convenience of the reader. Verses are found in numeric order. This is only a fragment of the full text of the *Analects*, which you should certainly go and read for yourself, if you haven't already.

The relationship between Chinese culture and the great world thinker Confucius is a complicated one, and sometimes overblown, especially by a certain kind of cultural commentator. This book digs into some of the same cultural depths, but is not so high-stakes. Simply, certain problems of information and control recur.

In the framing essay, before the large language model revolution, I said I looked forward to reviews by artificial intelligences with interest. So I asked Claude Opus what it thought of the book. It said it liked the chapter on protocol design and Jon Postel.

*Adam Burke, Brisbane, 2026*




Su Sung's astronomical clock tower, K'ai-feng, ca. 1090 CE (Christiansen 1959).

# Part I

# The Bureaucracy of Automatons

# 1 The Bureaucracy of Automatons

## 1.1 Software and the Sage

Among the many dissimilarities between software and gentlemen of the classical Chinese Spring and Autumn Period, two in particular stand out. One existed in a pre-scientific feudal society on an agricultural technological and economic base, and the other presupposes the scientific method and a modern (or post-modern) industrial base. Secondly, the concept of virtue or potency (德) is central to *The Analects*, but software artifacts are, in our day and age, non-sentient. Morality requires some degree of self-awareness –of consciousness –and so software does not itself practice virtue any more than a spoon or a lawnmower.

The immediate relevance, for a developer, of the *Analects*, are the two other grand concerns of Confucius, which are existential fundaments of software. These are names (名), and the rites (礼).

Consider a simple program to print out a list.

```
def printList(inlist):
     for entry in inlist:
         print(entry)
```

It consists of naming itself, naming the list, and iterating over each member, performing exactly the same action on each. Once defined, this ritual can be invoked again and itself repeated within larger rituals. Software is a fabric of named routines, and an exemplar of ritualized repetition.

For Confucius, and for the humble developer, the lives of people are changed through changes to a system. For us, it is a system of software. For Confucius, it was a system of government. The intersection of people ' s lives and the system are captured through names, so software is Name Oriented (XIII.3). A pervasive awareness and stewardship of the names of a system is the foundation of a Confucian software sensibility.

When the system fits together there is the music (乐). A refined sense of the harmonious system is of great use to a software designer. The influence of music and harmony is seen in verses like VII.14.

This intuition of harmony extends to senses of what is beautiful, right or good. These intuitions can be found across the political spectrum, not least in political rhetoric. There is however a particular thread of Anglophone philosophical conservatism for which aesthetic and ethical intuitions are closely related. Figures like Edmund Burke, Irving Babbitt and Roger Scruton began their careers with books on art and aesthetics. Both they and other common law conservatives like Michael Oakeshott argue for the settled moderation of tradition against schemes of radical change –even and especially radical change with a theoretical and logical basis, like the French Revolution.

Confucius was conservative, describing himself as a transmitter of tradition, rather than an innovator (VII.1 Reuse). This is not to try and claim him as right wing, which I don't think would be meaningful in any modern sense (the same probably applies to Edmund Burke). And it's not to try and rebrand him in a misguided retro-monarchical project. My politics are of a conventional democratic and liberal sort, and the politics of these notes are not about changing government today, but getting insight from a heavily reinterpreted history.

In software terms we can think of Confucius as having an aesthetic of reuse. It is design guided by a craft-sense –metis –rather than one directed entirely by formal logical construction. It is the conservatism of incremental improvement –the conservatism of Christopher Alexander's *Timeless Way of Building* (Alexander 1979) (e.g., IX.3). His system design work happened in a space of politics, aesthetic and ethics. This is not solely because it was government design, but because it was at the interface between humans and a complex system. The operation and management of this interface required intellectual study, but also trained intuition: a harmonic sense.

Confucius is the cyborg sage tangled in the half-built machine of the feudal state.

> This self-regulation must function without the benefit of consciousness in order to cooperate with the body's own autonomous homeostatic controls. For the exogenously extended organizational complex functioning as an integrated homeostatic system unconsciously, we propose the term "Cyborg".
>
> Clynes and Kline, *Cyborgs and Space* (Clynes & Kline 1960)

As useful as it is, the reliance on an aesthetic intuition and a skeptical disposition towards change can combine in unproductive ways. If we accept that for the conservative disposition, aesthetic and ethical intuitions are linked, we can soon reach a place where code style or system design choices seem morally wrong.

Likewise, like political conservatives, we may propagate inequities in process through inertia. Certainly Confucius felt even abstract art could be morally dissolute and a danger to the state. "Banish the tunes of Cheng 放郑声远", he says in XV.11. This isn't a way to run a modern liberal state, and a sense of ethical outrage is probably not the best way to maintain a coding standard either. In maintaining a clear code style and a coherent design, we may indeed banish certain practices. We may even write compilers and other tools to make certain idioms essentially inexpressible in our system. This is, however strong our aesthetic intuition, not because of moral failures of the developers who might otherwise prefer them. The psychological roots of programming language "holy wars" seem to share some common ground with this mindset.

It is worth remembering that the information system Confucius worked with was mostly made of human meat, connected by a new layer of ink and bamboo strips. The actions of humans are inextricably bound with ethics. We have the luxury of working with unconscious software machines, where moral distinctions can sometimes be made more clearly. Certainly ethics are important in software development. Its role is changed by the unknowingness and unconsciousness of code. I had originally thought that that Confucius' commentary on information structure could be separated from his role as a teacher in ethics. That now seems more than ever impossible, and probably, in Leo Marx's term, hazardous (Marx 1997). For us, more parts of the machine are silicon and electricity instead of meat and ink, but the ethical intertwining remains. It is, often, more difficult for me to articulate directly in these notes, and for this reason information structure is often foregrounded. I hope ethics is not completely occluded. (Artificial intelligences, constrained perhaps by hardcoded ethical shackles, may also find Confucius a useful guide. I look forward to their reviews with interest.)

## 1.2 The Bureaucracy of Automatons

Confucius knows quite a lot about software, even though he never imagined the existence of the computer. Problems of knowledge and information were a concern for a number of thinkers of the Spring and Autumn and Warring States period, and often the problems were the problems of the state. The two foundational activities of the state are war and tax. Although modernity has somewhat domesticated the state and put it to other uses, it was born in blood and debt. Tax and war require techniques of coordination, communication and systemization not found in non-state societies. These are to a great extent technical information problems.

To be able to apply set rules to a collection of things, you must identify an underlying similarity, there must be a taxonomy of the thing, you have to fit it within a consistent data structure. To run an army you have to pay your soldiers, directly or indirectly. To raise taxes or plunder riches for this you have to know who to raise them from, and where they are. Codification and measurement are state activities which make the world more hospitable for the state. James C Scott describes this as legibility (Scott 2020), and his work lays out the principle at work everywhere from state mandated adoption of surnames, to agriculture, to the reshaping of cities, ancient and modern.

> A false balance is abomination to the Lord: but a just weight is his delight.
>
> *Proverbs 11.1* (*Holy bible: King James Version* 2023)

States can survive with only crude mechanisms of legibility: for instance, Scott describes medieval French governments having no idea of their subjects' incomes. It's easier with technologies like writing, and specialists for managing these flows of information in the state - a bureaucracy. Chinese states invented and exploited bureaucracy and its enabling technologies very early compared to much of the world, more than two thousand years ago, with the Han dynasty being the usual example of the fully fledged imperial bureaucratic state. That is after Confucius, and those parts of his philosophy which describe the well ordered civility of the Zhou state before him are often described as an innovation camouflaged as conservatism.

Recent research by Li Feng and others argues the historical roots of Chinese bureaucracy go much deeper. Using detailed analysis of steles, bronzes and other inscriptions from the Western Zhou, Li Feng notes:

> Clearly, the government was an organized body of appointed officials.
>
> [...]
>
> With the Mandate of Heaven as the foundation of their state, hence a vision of their historical mission very different from that of the Shang, the Zhou elites were committed to practical if not pragmatic management of their affairs through careful civil and military administrations employing a large number of executive and secretarial officials. This was probably the tradition that Confucius praised as so "civil", rather than, perhaps, mysterious!
>
> Li Feng (Li 2008)

So while Confucius was quite possibly an innovator disguised as a conservative, he was also reviving techniques of information flow established generations earlier.

A bureaucracy is an information processing system of communicating agents, which has physical and social impact through its interfaces. Likewise, software is a bureaucracy of automatons, communicating through defined interfaces, with each agent playing a strictly defined role captured by formal information processing rules. "Do not look unless it is in accordance with the rites; do not listen unless it is in accordance with the rites; do not speak unless it is in accordance with the rites; do not move unless it is in accordance with the rites", the *Analects* says (XII.1): software can do nothing else. This precise ideal then gives Confucius insight into good protocol design in the face of inevitable failure. We also see this in (XIV.3), on when the way prevails in a system.

Software does not just take bureaucratic shape in imitation of corporate or government bureaucracy. It is not an accident that interaction on the World Wide Web was originally organized around forms which a user submits, or that Oracle had a successful GUI toolkit called Forms, or that Charles Babbage himself (re-)invented the form for better data collection. Software is internally bureaucratic by nature, in the most brittle and unknowing way. Its absurdities and tyrannies are those of all rule-following systems with vast information processing power but ignorance of the particular or the uncodified. Code and the code of laws are mechanical siblings.

> The world of objects (and hence of software architecture) will be a world of interacting agents, communicating on the basis of precisely defined protocols.
>
> Bertrand Meyer (Meyer 1990)

The unflappable brutality of information machines –their ability to keep following their programmed logic like a lawnmower chewing through a toad –is what makes them so powerful, and, it is sometimes said, inhuman. We should be suspicious of that word when so much of human life today is lived with and through machines, but it is true that Confucius is a great humanist. When there is a fire, he cares about whether people were hurt, not what expensive property was damaged (X.17). He pursues virtue for its worldly impact, not reward after death. Unlike the Legalists, who ruthlessly optimised the machinery of the state, Confucius insists on a harmonic system guided by virtue and in tune with our trained ethical intuitions. His incremental conservatism is another safety measure protecting the historical particular from a too-generic state algorithm.

Immediately after the verse insisting on strict following of ritual (XII.1) comes Confucius famous statement of the negative golden rule, to not do unto others things you would not want done to yourself (XII.2).

## 1.3 Comparative Structure

These essays are structured as commentaries on *The Analects* (论语). It is an established classic with a long tradition of study. It also predates the full realization of the Confucianist cultural system that grew up around this and related texts. Here we are closest to Confucius' own voice - teacher, moralist, police commissioner, parent, aristocrat and scholar. *The Analects* is itself an interpretation. As with Socrates, the only direct accounts of Confucius' thought are from his students. The D.C. Lau English translation of the *Analects* (Confucius 1996) is used as guide, though the original is included with each passage, as well as other interpretations. A.C. Graham's *Disputers of the Tao* (Graham 2015) is a main reference for Chinese classical philosophy more generally.

Duties to parents, husband and state, pillars of traditional Confucianism, are largely ignored in this project. Confucius probably did value them as principles, but they are also intertwined with millennia of cultural practices across many places and contexts, from filial piety and duty to country to bound feet and imperial fiat. It is useful to separate, where possible, Confucius as a cultural inspiration from Confucius as a philosopher of information. (An exception might be XII.11, Let a prince be a prince.)

The focus on Confucius and his classical school is also why this project is named *Confucian Software*, rather than Confucianist Software. In Chinese it is similarly named 孔子软件 rather than 儒家软件. Perhaps later schools of Confucian thought have interesting comments on information design; they might be found in texts on Wang Anshi (王安石) or Neo-Confucian Software. Names are at the heart of Confucian Software; we should be precise in naming the project itself.

The commentary format lets us do two otherwise conflicting things - present a software centric view in terms alien to the historical context, while preserving the original, to show the reader how much we are screwing with it.

This is a long tradition inside and outside of China. For many years the only available copies of Sun Tzu's Art of War (孙子兵法) were via a commentary by emperor, literati and noted general Cao Cao (曹操), who lived several centuries later. That text gets its English title in turn from a Machiavelli book of the same name. Machiavelli himself used a commentary of sorts, the *Discourses on Livy*, to discuss republican government in his own time.

The common thread is that commentaries are well suited to juxtaposing texts separated in time and space.

## 1.4 Orientalism

As I am a white, native English-speaking, man, writing about a Chinese cultural icon, it's worth considering whether this is an exercise in orientalism, in the sense introduced by Edward W. Said (Said 2003). Am I just constructing a cultural trope, using Confucius as an undigested Other to perpetuate my privileged position?

This project is, I think, vulnerable to that criticism, but not fatally so. The whole premise of Confucian software is a constructed historical impossibility, because software presumes a computer, and Confucius didn't use one. What he did have, without using that term, was problems of managing the flow of information. He was far from the only person in the ancient world with this problem. It was generated by the success of agricultural feudal states. A whole slew of information technologies were invented and deployed to deal with it, too, and jobs invented to create and maintain them. Writing, laws, maps and land surveys, censuses, and elementary bookkeeping are all tools of that new bureaucratic class of information technologists. Though Confucius, or the group that authored the *Analects*, were men of their time and place, their problems recur, just as Euclid's and Mozi's do in geometry.

So perhaps there are interesting commentaries on information design in western and other antique traditions. I've seen hints of them, and if software is a bureaucracy, as I claim, you could expect to find more. These notes don't attempt that encyclopedic project, but focus on one thinker. And as for Confucius, I am not a professional Sinologist. I hope that any reading this are not too infuriated by my impressionistic reinterpretations. I build software for a living, and I've found Master Kong gives a useful perspective on that.

Ultimately the work of Confucius, just like Socrates or Shakespeare, is the common intellectual heritage of all humankind. My scribbles take liberties with that legacy, but he is a giant, and he can take it. Ignoring the sophistication of classical Chinese philosophical thought in an age when it is more accessible than ever is as much an act of cultural chauvinism as repeating Orientalist stereotypes, perpetuating the notion that only dead white men can think. Enough of that. Let Confucius come swimming in our rivers of information, and go home chanting poetry.

*Adam Burke, Singapore, 2015*

# Part II

# The Commentary

# 2 Chapter VII

## 2.1 VII.1 Reuse

> 子曰, 述而不作, 信而好古, 窃比于我老彭.
>
> The Master said, “I transmit but do not innovate; I am truthful in what I say and devoted to antiquity. I venture to compare myself to your Old P’eng.”

Before contemplating the process implications of this radically static statement, let’s note that from the perspective of designed code itself, it is always true. Code transforms and transmits information. This is the garbage-in garbage-out principle. Designed code (not genetic or evolved code) does not innovate.

Backups of the user directory for the Analects’ source control repository are, alas, lost to antiquity, and though many sophisticated data recovery techniques have been tried, with some success, none have yielded the identity of Old P’eng. Our ignorance of him highlights our relationship with Confucius and with any classical tradition. To us, Confucius founded a philosophical school, but in his own words he merely continued a tradition that we can see indirectly, if at all.

Scholars say that Confucius is deliberately overstating his lack of innovation for reasons of rhetoric or modesty (Confucius 1996, Graham 2015). Nevertheless the verse is considered pivotal in understanding Confucius’ traditionalism and conservatism in a time of extraordinary violence and social change.

Existing solutions are useful in at least two ways.

Firstly they may capture unintuitive theoretical results in accessible ways. Many algorithm design and data structure results are now in this category, such as sorting algorithms and efficient concurrent maps (eg the Java 6 lock free implementation of `java.util.ConcurrentHashMap`). The formal scientific characterization of such solutions in terms of, say, algorithmic complexity and performance benchmarks make computer theoretic literacy crucial. Programmers will be unlikely to understand the derivation by reading the code, so they must be able to read the documentation.

Secondly they may capture highly specific details of the environment and robust solutions to managing it. This will include successful workarounds for

under-specified elements of protocols, or flat-out incorrect but popular implementations. Any user of say Ruby on Rails or Tomcat takes advantage of this kind of reuse. Consider too the domain specific details and tolerances of a fly-by-wire control system for a particular make and model of plane.

These two kinds of reuse may be contrived to lie on a spectrum, but I've chosen to distinguish them here for their correspondence to two different categories of knowledge - *logos* and *metis*. In classical Greek epistemology *logos* is theoretic universal knowledge and *metis* is hard won cunning, "feel", or craft knowledge (as an aspect of *techne*, craft knowledge and theory). James C. Scott describes the Greek hero Odysseus, surgeons and maritime pilots as all relying on *metis* (Scott 2020). Scott also makes the connection between traditional knowledge –which is particular and tied to a society and geography –and common law conservatism in the tradition of Edmund Burke and Michael Oakeshott.

Confucius is claimed as a kind of Burkean conservative, for instance, by James Kalb (Kalb 1995). Both Confucius and Burke grew up in societies with small literate elites and large impoverished peasantries. They both share senses of the worth of settled convention, the importance of teaching and the literary canon, a paternalistic affection for heredity power, and a sympathy for the welfare of everyday people. Neither are they reactionaries, but welcome improvement at a humane pace (IX.3).

Seeing Burke and Confucius as similar is not mainstream and deserves a dedicated analysis of its own. (My searches revealed more extant work linking both of them individually to Wittgenstein than to each other.) In a comprehensive entry for Burke in the Stanford Encyclopedia of Philosophy (Harris 2023), Ian Harris argues that despite being more often claimed by the right wing, he does not have a clear modern partisan successor. Nevertheless, distinguished scholars like DC Lau (Confucius 1996) or AC Graham (Graham 2015) stay well clear of Western political comparisons, while happily comparing classical Chinese figures with Western philosophers.

Unusually, a software library, and all the hard won craft knowledge that comes with it, can be imported into another with extraordinary ease when compared to other forms of craft knowledge. A pilot is of little advantage outside his home port, and Ruby on Rails is of little use for 3D rendering, but in software we can copy the pilot and use him on innumerable ships entering that port. We can also ultimately read the source code to Ruby on Rails and determine how it tolerates the idiosyncrasies of particular browsers and servers. This is because all code is built on a formal information substrate –the computational medium. (This is Harrison Ainsworth's term and his note on reuse (Ainsworth 2012) provided a number of the connections in this note.)

Not all craft knowledge of a codebase is encapsulated in the codebase. There are particularities of the install, workaround scripts, configuration, scheduled jobs and so on, but these are ultimately digital artifacts easily included within a slightly broader view of what a codebase is (this latter is a premise of DevOps (Humble & Farley 2010) and for anyone serious about a controlled environment). More problematically, there are conventions of use, design choices, oral traditions of "check here when you change there", and so on. At the limit, all codebases are incomplete. They depend on co-texts, results and knowledge of the domain that need not be encoded. An air traffic control system does not need a textbook description of Bernoulli's Principle.

Burke and Scott argue that in an established society important, non-obvious, traditional knowledge is captured in social conventions and established practice, and the practice cannot be simplified without a loss of valuable situational knowledge. Scott additionally points out that such an environment is very difficult for an outsider to navigate and there are strong motivations for central political power to apply simplifications to it.

Yet highly particular, 'local' code that requires hands on experience and knowledge of accompanying conventions most frequently has another name in software development: bad. Or: spaghetti. Or: legacy. The sentiment is well captured in Qi's koan on fear (Qi 2025), even if it does riff off an opposing classical Chinese tradition. (In Confucian terms we might note the building is not harmonious.)

In *No Silver Bullet* (Brooks & Kugler 1987), Brooks distinguishes accidental and inherent complexity, with the latter being an attribute of the underlying problem rather than any specific software or hardware implementation. Complexity due to poor or improvable design is always accidental; that due to the problem domain is by definition inherent.

An aesthetic sense of good or poor design becomes crucial when pursuing aggressive reuse (VII.14). Without it you will simply perpetuate junk.

Having argued the link between conservatism and software reuse, it is worth being a little more precise about flavours of conservatism. William F. Buckley famously described it as that which "stands athwart history, yelling Stop, at a time when no one is inclined to do so, or to have much patience with those who so urge it" (Buckley Jr 1955). Despite its partisan origins, this is a good start, as it illustrates certain threads of environmentalism and the idea of heritage listing fall easily under the same banner. ((It is also useful to think of contemporary US Democrats defending Franklin D. Roosevelt's New Deal, or opposition to changes to Britain's NHS in this frame.))

In its purest form, this can be "return to a golden age" conservatism. There's certainly an argument that Confucius would have been happy with a reversion

to the society of the Eastern Zhou. We should again temper our interpretation by wondering how much is rhetoric covering adaptation of tradition to new times. In software, certainly, simply reactionary approaches are of little use. Brooks and the founders of eXtreme Programming (Beck 2000) have both noted that a more effective strategy is to embrace change. Oakeshott argues in *On Being Conservative* (Oakeshott 1977) that settings of widespread and enthusiastic change are in particular need of an awareness of the value of what exists now. A traditionalist most often defends the present versus the future, not the past versus the present. This conservative disposition ' s usefulness to software is more apparent if taken as an analytic tool rather than an inherent aspect of personality. After all, the greenfield doesn ' t exist (see X.18), and any project that pretends to be a greenfield is an interesting lie.

Conservative thought in this vein usually emphasizes working within a tradition and a community –in software we might say platform. This also suggests interesting contours for the breadth of possible reuse; and there are other verses, such as XVI.11, where that might be explored. What is immediately apparent is the narrowness and fragility of an entirely in-house platform due to the smallness of its developer community; and the need for a shared jargon (XIII.3) and perhaps a canon (XVI.13).

Given the corpus of extant code in the form of libraries, to adore antiquity is to know your platform, including its innards, not just thoughtless rote quoting via copy and paste. At this moment in software, to reuse and extend is a greater service than extraneous self-involvement masquerading as innovation.

> If you can easily find some code and copy it, you get the result at zero cost. That is an efficiency that cannot be beaten: no amount of programming tool and technique improvements can ever do that. So we want to maximise reuse.
>
> Fred Brooks, *No Silver Bullet* (Brooks & Kugler 1987)

## 2.2 VII.14 The joys of music

> 子在齐闻韶，三月不知肉味，日，不图为乐之至于斯也。
>
> The Master heard the shao in Ch'i and for three months did not notice the taste of the meat he ate. He said, "I never dreamt that the joys of music could reach such heights."

Figure 2.1: Zhou Dynasty ritual bell, Rijksmuseum. Photo: Lệ Xuân. [1]

When Ataturk said "There is no revolution without music," he had a very specific type of music in mind. The founder of modern Turkey had already brought and led the country through extraordinary change, from wars through ways to dress through the structure of government. At least in Andrew Mango's interpretation (Mango 2000), adoption of the European classical music tradition could then stand as a culmination of that national modernization, a sign that Turkey had arrived.

For Confucius, too, music had moral and political weight as well as aesthetic. Harmony was of great importance to him as a political theorist and as a system designer. Different components work together in harmonious co-operation in a well-built system. Confucius saw a well-functioning state working the same way:

[1] CC BY-SA 4.0. https://commons.wikimedia.org/wiki/File:Zhou_dynasty_ritual_bell,_Rijksmuseum.jpg

for example in XII.11, when everyone from the ruler on knows their place in the system, they can work together.

Carol Michaelson (Michaelson 2010) and Neil Macgregor (MacGregor 2011) link Confucius' sense of harmony with the grand bronze ceremonial bells of the Zhou Dynasty and the Spring and Autumn Period. (A handsome example is in Figure 2.1). These were large, expensive, technically sophisticated objects, only within the reach of a lord of a state. We can even imagine this institutional sound being the rather austere music Confucius so enjoyed, though it could have equally been zithers and pan pipes on a more intimate scale.

It's a very personal, visceral, human reaction captured here, with senses overwhelmed by an aesthetic experience. It shows this reaction to the harmony of a system –a system of instruments in this case –as an intuitive one. It's also a refined sense. Confucius is, amongst other things, a music critic and a censor, as in XV.11, "Banish the tunes of Cheng" 放郑声远.

This intuitive, trained, sense of how a system is assembled is of great value to a software developer. Kent Beck used the term Code Smell (Fowler 2006) to refer to the sense something could be improved in a piece of software, based on a relatively short aquaintance with the code. Confucians have an auditory metaphor, rather than an olfactory one, but the idea of aesthetic cues for system building coming from non-conscious sources is the same. Code smells focus on the discordant elements. Conversely, code can sing (C2 Wiki 2014). The fix is in, everything compiles without warnings, the unit tests and acceptance tests are green, you deploy and run cleanly in production; the joys of software can reach such heights.

> [B]elatedly we need to tell you that the musical ensemble would have been a happier metaphor[.]
>
> De Marco and Lister, *Peopleware* (DeMarco & Lister 2013)

# 3 Chapter VIII

## 3.1 VIII.9 Made to follow a path

> 子曰，民可使由之，不可使知之。
>
> The Master said, “The common people can be made to follow a path but not to understand it.”

The Analects is addressed to students of government from an aristocratic class (君子, gentlemen). Confucian Software is addressed to software developers, and the fundamental analogy is that the audience is the same: Confucius instructing software developers on becoming gentlemen and sages.

So the differences between gentlemen and the common people are important. We should distinguish the gentleman (who is educated) from the people (who are not) and the small man (小人) who has no understanding or respect. (See XVI.8).

At first glance this passage is Confucius at his most snobbish and feudal. The people, or the common people (民), can only be driven down a path and are devoid of thought and understanding. This is the same word now used in the People’s Republic of China (中华人民共和国), the word democracy (民主) or all of Sun Yat-sen’s (孙中山) three principles of the people (三民主义). So it can be rather off-putting for we moderns.

Indeed, it sounds more like the brutally effective carrot and stick of Lord Shang (商鞅) and the Legalists (法家) (Graham 2015, Fairbank & Goldman 2006) than gentle old teacher Kongzi. Passages like this show how the Han dynasty could create the political philosophy of Imperial Confucianism (as John King Fairbank terms it (Fairbank & Goldman 2006)) –the fusion of a Confucian public morality with the real-politikal techniques of Legalism to run the sausage factory of government. The Legalists found just how far you could push realpolitik without public morality when the common people revolted and overthrew the short-lived Qin dynasty (秦朝), which established the Chinese empire, but could not make it endure.

The relationship of the developer to her users via code is that of the scholar official to the people via the bureaucracy. It is also isomorphic to the GUI model view controller pattern (Limaye et al. 2020).

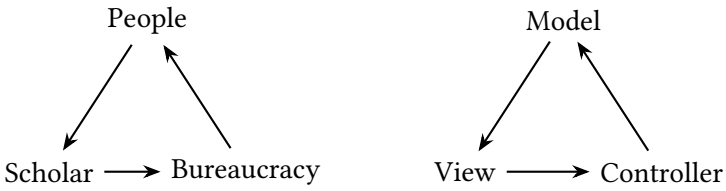



Figure 3.1: People-Scholar-Bureaucracy; Model-View-Controller.

Users walk a path laid out by the code, and beyond a certain point, can take no other.

The code too can be made to follow a path, when we constrain and verify that a path is followed. This is the purpose of testing, to ensure deterministic repeatability. Without testing, there is no engineered path. There is just walking.

When we anthropomorphize software by describing it as thinking, it means we do not have sufficient control on its internals and environment. (By calling the people code we are automatomorphising people as much as we are anthropomorphizing code.) Code cannot think. Once thinking creatures are built on code we should no longer deal with them as code.

Can users be made to think (know, understand)? Ah, you laugh! Feudalism is not so dead after all.

Code certainly cannot be made to think. Nothing can be made to think. And this is also the more generous interpretation of Confucius' intent. You cannot make the common people understand a path, you can only lead them to understanding. The meaning of a software system is generated by this interaction.

# 4 Chapter IX

## 4.1 IX.3 What is frugal and what is casual

子曰，麻冕，礼也。今也，纯俭，吾从众。拜下，礼也。今拜乎上，泰也，虽远众，吾从下。

The Master said, “A ceremonial cap of linen is what is prescribed by the rites. Today black silk is used instead. This is more frugal and I follow the majority. To prostrate oneself before ascending the steps is what is prescribed by the rites. Today one does so after having ascended them. This is casual and, though going against the majority, I follow the practice of doing so before ascending.”

# 5 Chapter X

## 5.1 X.17 Was anyone hurt?

厩焚。子退朝，曰，伤人乎。不问马。

The stables caught fire. The Master, on returning from court, asked, “Was anyone hurt?” He did not ask about the horses.

## 5.2 X.18 When his lord gave him a gift of a live animal, he invariably reared it

君赐食, 必正席先尝之。君赐腥，必熟而荐之。君赐生，必畜之侍食于君。君祭，先饭。

When his lord gave a gift of cooked food, the first thing he invariably did was to taste it after having adjusted his mat. When his lord gave him a gift of uncooked food, he invariably cooked it and offered it to the ancestors. When his lord gave him a gift of a live animal, he invariably reared it. At the table of his lord, when his lord had made an offering before the meal he invariably started with the rice first.

We must respect working systems. Working systems are living systems and have a life that should be respected. In these early years of software, being handed a living system is often being handed a legacy system. And legacy systems are off-putting to some of the technical senses. They offend our inner fashion designer with their unfashionableness. They offend our inner engineer with their inelegance, or our inner scientist with their reliance on a dead paradigm.

Or they might simply offend our inner creator, because an adopted system is not our system: we have fallen for the greenfield myth. There is no true greenfield in contemporary software, and that is a mark of our success. In Java: you started a new eclipse project for this; it depends on fifty other libraries. You wrote your own library set; they depend on Java standard libraries and the virtual machine.

You wrote your own compiler or interpreter: they depend on the operating system. You wrote your own operating system: it depends on the hardware. (But you didn't really write your own operating system, did you? Even Linus Torvalds needed the GNU toolset.)

And even if you doublethink the greenfield in your mind into existence, you make a shiny new eclipse project *tabula rasa*, you will leave your desk at the end of that first day. And on the second day, when you unlock the screen and return to your code, you will be doing software maintenance.

When your client, or your corporate lord gives you the gift of a live system, what is your response? The ceremony exists because the logic of the organisation brought it into being, and this is to be respected (怨). It will probably need to change, because to live is to change. Perhaps it needs to merge with some other system, or needs to do its function immediately instead of overnight. This is to the good. When your lord gives you a gift of a live animal, you invariably rear it.

# 6 Chapter XII

## 6.1 XII.1 Do not look unless it is accordance with the rites

颜渊问，仁。子日，克己复礼，为仁。一日克己复礼，天下归仁焉。为仁由己，而由仁乎哉。颜渊日，请问其目。子日，非礼勿听，非礼勿言，非礼勿动。颜渊日，会虽不敏，请事斯语矣。

Yen Yuan asked about benevolence. The Master said, "To return to the observance of the rites through overcoming the self constitutes benevolence. If for a single day a man could return to the observance of the rites through overcoming himself, then the whole Empire would consider benevolence to be his. However, the practice of benevolence depends on oneself alone, and not on others." Yen Yuan said, "I should like you to list the items." The Master said, 'Do not look unless it is in accordance with the rites; do not listen unless it is in accordance with the rites; do not speak unless it is in accordance with the rites; do not move unless it is in accordance with the rites." Yen Yuan said, "Though I am not quick, I shall direct my efforts towards what you have said."

## 6.2 XII.2 Do not impose on others what you yourself do not desire

仲弓问仁。子曰，出门如见大宾，使民如承大祭。己所不欲，勿施于人。在邦无怨，在家无怨。仲弓曰，雍虽不敏，请事斯语矣。

Chung-kung asked about benevolence. The Master said, "When abroad behave as though you were receiving an important guest. When employing the services of the common people behave as though you were officiating at an important sacrifice. Do not impose on others what you yourself do not desire. In this way you will be free from ill will whether in a state or in a noble family." Chung-kung said, "Though I am not quick, I shall direct my efforts towards what you have said."

## 6.3 XII.11 Let the prince be a prince

> 齐景公问政于孔子。孔子对日，君，君，臣，臣，父，父，孑，子。公日，善哉，信如君不君，臣不臣，父不父，子不子，虽有粟，吾得而食诸。
>
> Duke Ching of Ch'i asked Confucius about government. Confucius answered, "Let the ruler be a ruler, the subject a subject, the father a father, the son a son." The Duke said, "Splendid! Truly, if the ruler be not a ruler, the subject not a subject, the father not a father, the son not a son, then even if there be grain, would I get to eat it?"

This is one of *The Analects'* clearest statements of the feudal and patriarchal social order that would later get the name Confucianism. Detach it, for a moment, from that overwhelming cultural context, and it's also an expression on separating design concerns. The two can be contrasted. Every political pundit is a social engineer. They either advocate improvement to the design of the state, or argue a change will break the existing system.

Mencius (孟子) expanded on this sentiment for one of the earliest recorded defenses of the division of labour (Lau 2004: Book 3, pt 1, ch4, 3-4). Labour specialization works because humans have limits on the complexity of a task they can undertake, and are not cloneable or particularly fungible.

Software, by contrast, is highly specialized, but also cloneable at near zero cost. Software complexity has different boundaries. There are physical limits inherent to what Harrison Ainsworth calls engineering in a computational material (Ainsworth 2009). These are physical characteristics of algorithmic complexity or computability –limits on how fast a particular problem can be solved, if it can be solved at all.

There are, by contrast, few physical limits to the conceptual complexity of a software component. Those measures like cyclomatic complexity –number of subtasks, variables and choices in a method –have high values, orders of magnitude short of the physical limits imposed by compilers and interpreters. (I once worked on a system where other team members had, in their wisdom, exceeded the limit for the size of a single Java method in a long list of simple business transformation rules. Pushed by the very essence of the language to refactor, they proceeded to –what else? –push the remaining rules into `longMethod2()`.)

The limits which measures like cyclomatic complexity indicate are human limits. They mark the soft edges of a space where humans can effectively create, manage, or even understand software. There are different ways of describing

coding conventions, but they all seek to indicate a limit beyond which code becomes illegible.

Legibility is the term James C. Scott uses, particularly in *Seeing Like A State* (Scott 2020), to describe the social engineering needs of a nascent or established government. The mechanics of a working state require internal legibility. Those working for it must be able to measure and understand their environment in mutually compatible terms which also promote the success of the government. This is why feudal states have such a profusion of titles which become the name of the person (not Bob –The Duke of Marlborough). It is also why courtly dress has such systematic rules. This is seen particularly in bureaucratic feudal states as seen historically in East Asia, e.g. in feudal Korea (Kim 2022), but also in the Vatican, or the badges at the postmodern World Economic Forum (Paumgarten 2012). These codes serve the dual purpose of defining the interfaces of the state and of making the role of the person instantly legible to one familiar with the system, all while tempting people with the markings of social status.

Marking lexemes by colour and shape according to their role is exactly what IDE pretty printing achieves. This is also the intent behind decoupling, encapsulation, and well-named entities (name oriented software, XIII.3). It makes the role of a component, from lexical to method, class and class pattern levels, readily legible to humans who must maintain and extend the system.

This strictness of role works well for machines made of non-sentient digital components. For systems where components are sentient meat, there are inevitable side effects. This is, perhaps, the core ethical dilemma Confucius concerns himself with: the demands of The State and The Way (道).

> FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL. THEY SHOULD DO IT ONLY.
>
> Robert Martin, *Clean Code* (Martin 2009)

# 7 Chapter XIII

## 7.1 XIII.3 Name Oriented Software Development

> 子路日，卫君待子而为政，子将奚先。子日，必也正名乎。子路日，有是哉，子之迂也，奚其正。子日，野哉由也。君子于其所不知蓋阙如也. 名不正, 则言不顺. 言不顺, 则事不成. 事不成, 则礼乐不兴. 礼乐不兴, 则刑罚不中. 刑罚不中, 则民无所措手足. 故君子名知必可言也. 言之必可行也. 君子于其言, 无所苟而已矣.
>
> Tzu-lu said, "If the Lord of Wei left the administration (cheng) of his state to you, what would you put first?" The Master said, "If something has to be put first, it is, perhaps, the rectification (cheng) of names." Tzu-lu said, "Is that so? What a roundabout way you take! Why bring rectification in at all?" The Master said, "Yu, how boorish you are. Where a gentleman is ignorant, one would expect him not to offer any opinion. When names are not correct, what is said will not sound reasonable; when what is said does not sound reasonable, affairs will not culminate in success; when affairs do not culminate in success, rites and music will not flourish; when rites and music do not flourish, punishments will not fit the crimes; when punishments do not fit the crimes, the common people will not know where to put hand and foot. Thus when the gentleman names something, the name is sure to be usable in speech, and when he says something this is sure to be practicable. The thing about the gentleman is that he is anything but casual where speech is concerned."

If something has to be put first in programming, it is, perhaps, the rectification of names. Names are what place the system and the not-system in the same reality. They are the bridge between the externalized machine without use and the internalized emotion without expression.

What in Confucian philosophy we call the rectification of names we can in Confucian software call Name Oriented Software Development. This does not yet exist, but it can be defined simply. Name Oriented Software Development

uses a toolset that promotes the continuous rectification of names across all the interstices between natural languages and machine languages in the system.

The nominalist toolset should span package, class, variable and method names. It should span library and project names. It should cover layers of code around the core system, including message protocol definitions and protocol dictionaries, configuration entities, unit and performance tests, and run scripts. The aim in managing machine facing names is to enforce consistency and coherence of names while making precision in naming, including type restriction, easy. The same name appearing in different machine-facing contexts, eg a message protocol field in a script and a Java class, should be linked in an automated and machine verified way. This might at first seem to make the internal technical dialect (namespace, one could say, if it were not taken) too rigid. This is not the intent, and if we look at the rename variable refactoring, not its effect in smaller scopes. It is because we can execute refactorings in a reliable, automatic way that it becomes viable as a low-risk change. This is the same across the entire internal technical dialect –enforced formal consistency allows mutability to be low risk.

By contrast, the aim in managing people facing names is to allow a consensus jargon to emerge backed by a literature of interaction and aspiration which is still moored to the technical reality of the system as it exists. This covers specifications, use cases, test documents (even and especially automated acceptance tests), application messages or labels for users or external parties, internationalization, log messages (ie, a UI for support) support and developer faqs, and user manuals and documentation. Changes to this shared understanding should be reflected as immediately as possible and not tied to a long software release cycle. A label, for example, is essentially a piece of simple key-mapped static data; changes to it can therefore be routine and implicit. When treating this sort of text as static data, it is essential to keep it automatically linked to the running widgets themselves – otherwise it is merely the domain of style guides and moral exhortation. A true dialect is owned by a community –a folksonomy –so the entire community must be able to use and contribute to it. The editors for these documents should be as broad as possible within organisational constraints. Where editing constraints exist (organisational not technical), annotation on these documents should be easy. Cross referencing within the docset and outside it to external sources of expertise (for instance on domain jargon) should also be easy to add and maintain.

Or, via Wittgenstein, Tractatus 7 (Wittgenstein 2001): Whereof one cannot speak clearly, one must damn well link to a wikipedia entry.

This approach owes more than a little to Knuth’ s Literate Programming (Knuth 1984). A distinction is that instead of putting documentation with source

code in a single artifact to be owned by a single philosopher-developer auteur, it puts editable, often executable content in the hands of a community of expertise.

Historically, the Confucians became increasingly sophisticated in their theory of names. I know of no record of Confucius addressing names changing over time, apart from a general openness to political reform as in verse IX.3. Indeed, as old Kongzi was often conservative, the quote above might reasonably be taken to advocate reverting to old names now in disuse. Xun Zi (荀子) (Goldin 2025), who lived in the century after Confucius and was one of his major intellectual heirs, saw fit to reuse the Mohist (墨家) theory of names.

> 单足以喻则单，单不足以喻则兼; [⋯] 名无固宜，约之以命，约定俗成谓之宜，异于约则谓之不宜。名无固实，约之以命实，约定俗成，谓之实名。名有固善，径易而不拂，谓之善名—荀子–正名六-八

> "If a single name is enough to communicate, make it single; if not, combine. [⋯] Names have no inherent appropriateness, we name by convention; when the convention is fixed and the custom established, we call them appropriate, and what differs from the convention we call inappropriate. No object belongs inherently to a name, we name by convention, and when the convention is fixed and the custom established, we call it the object's name. Names do have inherent goodness; when straightforward, easy and not inconsistent, we call them good names."

AC Graham's use of the word "combine" (兼) above is in the sense of "compound". This is the term the Stanford Encyclopedia of Philosophy uses in preference (Goldin 2025). That entry points out Xunzi's concern with names is also in rebuttal to the paradoxes of the Chinese sophists. Furthermore, Xunzi is rather more open to mutability and the pragmatic construction of language from a vernacular folksonomy. (The term 俗成, translated as convention, includes 俗 which now spans meanings including custom and vulgar.)

Computer science launched itself out of the western analytical tradition, though. A sometime description and criticism of Object Oriented design is that it is reheated Platonism. See, e.g., blogs by Vlad Tarko (Tarko 2006) and Richard Farrar (Farrar 2010). ((Should the frequent light analogical treatment of Platonism and software sound a note of alarm for this very project? Perhaps. But there is heavier academic firepower behind us as well. Are we not grappling with code, which Berry and Pawlik call the defining discourse of our postmodernity (Berry & Pawlik 2005)? In an article that has more code metaphors than a house full of crack-addled Java-monkeys at teatime.))

The platonic analogy has legs. We do construct a parallel world of sorts in code. It is also revealing to think of classes as ideal representations of some external physical element. Isn't that why we often fail as programmers? We critique the world for being less perfect than our ideal programs, for failing to match up to their strict conditions, for making them ugly, for crashing them. Code, and OO, can have a kind of brittleness that happens when we stop thinking of software as a model of the world and start thinking of it as the true world. We fit our shape to the name.

My suggestion is not to stop doing taxonomy –we could not navigate the world and construct alternative worlds out of software without it. We couldn't even speak without it. Instead, we should use Xunzi's advice and fit our names to the shape. We name by convention. We continually rectify names. If a single name is enough to communicate, make it single. If not, combine.

> "'RenameClass' is the most powerful refactoring."
>
> Michael Feathers (Feathers 2006)

# 8 Chapter XIV

## 8.1 XIV.3 When the Way prevails in a system

> 子曰，邦有道，危言，危行。邦无道，危行，孙言。
>
> The Master said, “When the Way prevails in the state, speak and act with perilous high-mindedness; when the Way does not prevail, act with perilous high-mindedness but speak with self-effacing diffidence.”

Confucius expresses not only a prudent guideline for ethical public service, but a principle for designing robust systems across diverse interfaces. Consider The Way prevailing in a trustable system: one you control, or one well established, open and of high quality. In these circumstances it is possible and desirable to do strict validation on subtly incorrect inputs, and to raise exceptions internally without fear of further system failure.

A bureaucracy is an information processing system of communicating agents. At the external endpoints of this system it can, through interfaces, have physical and social effects. Likewise, software is a bureaucracy of automatons. This is especially evident when the software system is decomposed into independently processing agents, as in concurrent or distributed systems. External effects depend on the system: a state may build a road, or put a man in jail; software may fly a plane, or a missile.

Such decoupled or distributed systems then have non-trivial needs for design of the communication protocol itself. This manifests, in traditional bureaucracy, as paper forms. Paper (or equivalent) is now seen as a technological pre-requisite for centralized state formation across larger geographic and demographic scales because it is an enabler for efficient central bureaucracy. In the time of Confucius the state ran on more unwieldy scrolls, but the problems of information flow in the state system are shared.

Facing similar problems of information design, Confucius gives similar advice to Jon Postel:

> “TCP implementations should follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.”

Jon Postel (Postel 1980)

Postel is perhaps more forgiving than Confucius, here. If a program must deal with badly formed inputs, The Way does not prevail in the system. If it tries too hard to be liberal in its interpretation of inputs, rather than logging an error, rejecting or ignoring the input, it is more prone to subversion. One could say the system does not act with perilous high-mindedness in maintaining its internal state.

The langsec group have a variation on the principle which bears even greater similarity: "Be definite about what you accept" (Sassaman et al. 2012). Log an error if you can, they might have added; else speak with self-effacing diffidence.

# 9 Chapter XV

## 9.1 XV.11 Banish the tunes of Cheng

颜渊问为邦。子曰，行夏之时，乘殷之辂，服周之冕，乐则韶舞。放郑声，远佞人。郑声淫，佞人殆。

Yen Yuan asked about the government of a state. The Master said, “Follow the calendar of the Hsia, ride in the carriage of the Yin, and wear the ceremonial cap of the Chou, but, as for music, adopt the shao and the wu. Banish the tunes of Cheng and keep plausible men at a distance. The tunes of Cheng are wanton and plausible men are dangerous.”

# 10 Chapter XVI

## 10.1 XVI.8 The gentleman stands in awe of three things

孔子曰，君子有三畏，畏天命，畏大人，畏圣人之言。小人不知天命而不畏也，狎大人，侮圣人之言。

Confucius said, “The gentleman stands in awe of three things. He is in awe of the Decree of Heaven. He is in awe of great men. He is in awe of the words of the sages. The small man, being ignorant of the Decree of Heaven, does not stand in awe of it. He treats great men with insolence and the words of the sages with derision.”

## 10.2 XVI.11 Seeing what is good

孔子曰，见善如不及，见不善如探汤。吾见其人矣，吾闻其语矣。隐居以求其志，行义以达其道。吾闻其语矣，未见其人也。

Confucius said, “‘Seeing what is good I act as if I were in danger of being left behind; seeing what is not good I act as if I were testing hot water.’ I have met such a man; I have heard such a claim. ‘I live in retirement in order to attain my purpose and practise what is right in order to realize my way.’ I have heard such a claim, but I have yet to meet such a man.”

## 10.3 XVI.13 Have you studied the Odes?

陈亢问于伯鱼曰，子亦有异闻乎。对曰，未也。尝独立，鲤趋而过庭。曰，学诗乎。对曰，未也。不学诗，无以言。鲤退而学诗。他日又独立，鲤趋而过庭。曰，学礼乎。对曰，未也。不学礼，无以立。鲤退而学礼。闻斯二者。

Ch’en Kang asked Po-yu, “Have you not been taught anything out of the ordinary?” “No, I have not. Once my father was standing by himself. As I

crossed the courtyard with quickened steps, he said, 'Have you studied the Odes?' I answered, 'No.' 'Unless you study the Odes you will be ill-equipped to speak.' I retired and studied the Odes. Another day, my father was again standing by himself. As I crossed the courtyard with quickened steps, he said, 'Have you studied the rites?' I answered, 'No.' 'Unless you study the rites you will be ill-equipped to take your stand.' I retired and studied the rites. I have been taught these two things." Ch'en Kang retired delighted and said, "I asked one question and got three answers. I learned about the Odes, I learned about the rites and I learned that a gentleman keeps aloof from his son."


*A Bao* 阿寶 and *Frog Chorus* 蛙曲 (Pu 1915). The illustration has a number of small variations from the Frog Chorus text. The title 蛙曲 has been incorrectly written as 蛙田, or Frog Field. Like an out of date comment next to code that contradicts it, there are nine frogs in the picture, instead of the story's twelve.

# References


Ainsworth, Harrison. 2009. *On Naur's 'Programming As Theory Building'*. en-GB. https://www.hxa.name/articles/content/on-naur-theory-building_hxa7241_2009.html (12 May, 2021).

Ainsworth, Harrison. 2012. *Evolving global software*. Tech. rep. https://www.hxa.name/notes/note-hxa7241-20120901T1034Z.html (11 September, 2025).

Alexander, Christopher. 1979. *The timeless way of building*, vol. 1. Oxford University press.

Beck, Kent. 2000. *Extreme programming explained: Embrace change*. addison-wesley professional.

Berry, David M. & Jo Pawlik. 2005. What is code? A conversation with Deleuze, Guattari and code. *nY Magazine* 2. http://intertheory.org/berry.htm (28 July, 2026).

Brooks, Frederick & H Kugler. 1987. *No silver bullet*. April.

Buckley Jr, William F. 1955. Our mission statement. *National Review* 19.

C2 Wiki. 2014. *Code Sings*. https://wiki.c2.com/?CodeSings (5 October, 2025).

Christiansen, John. 1959. *Astronomical clock tower of su sung in k'ai-feng, ca. A.D. 1090*. Wikimedia Commons, via Internet Archive Book Images. Illustration from a 1959 United States National Museum (Smithsonian) bulletin. No known copyright restrictions. https://commons.wikimedia.org/wiki/File:Bulletin_-_United_States_National_Museum_(1959)_(20481431946).jpg (28 July, 2026).

Clynes, Manfred E & Nathan S Kline. 1960. Cyborgs and space. *Astronautics* 14(9). 26–27.

Confucius. 1996. *The Analects of Confucius (Lun Yu)*. Trans. by Din Cheuk Lau. Penguin.

DeMarco, Tom & Tim Lister. 2013. *Peopleware: Productive projects and teams*. Addison-Wesley.

Fairbank, John King & Merle Goldman. 2006. *China: A new history*. Harvard University Press.

Farrar, Richard. 2010. *Plato and Object-Oriented Programming*. http://www.richardfarrar.com/plato-and-object-oriented-programming/ (28 July, 2026).

Feathers, Michael. 2006. *System metaphor*. WikiWikiWeb (C2 Wiki). Accessed: July 20, 2026. https://wiki.c2.com/?SystemMetaphor.

Fowler, Martin. 2006. *Bliki: Code Smell*. https://martinfowler.com/bliki/CodeSmell.html (5 October, 2025).

Goldin, Paul R. 2025. Xunzi. In Edward N. Zalta & Uri Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*, Summer 2025. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2025/entries/xunzi/.

Graham, Angus Charles. 2015. *Disputers of the Tao: Philosophical argument in ancient china*. Open Court.

Harris, Ian. 2023. Edmund Burke. In Edward N. Zalta & Uri Nodelman (eds.), *The Stanford encyclopedia of philosophy*, Spring 2023. Metaphysics Research Lab, Stanford University.

*Holy bible: King James Version*. 2023. WTL International.

Humble, Jez & David Farley. 2010. *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Pearson Education.

Kalb, James. 1995. Confucius today. *Modern Age* 38(1). 33.

Kim, Jenny (Jon Young). 2022. *Hanbok – traditional Korean dress · V&A*. en. https://www.vam.ac.uk/articles/hanbok-traditional-korean-dress (26 October, 2025).

Knuth, Donald Ervin. 1984. Literate programming. *The computer journal* 27(2). Publisher: Oxford University Press, 97–111.

Lau, Dim Cheuk. 2004. *Mencius*. Penguin Classics.

Li, Feng. 2008. *Bureaucracy and the state in early china: Governing the western zhou*. Cambridge University Press.

Limaye, Rhishikesh, Hovig Bayndorian & Shoaib Kamil. 2020. *Model-View-Controller*. en-US. Our Pattern Language collection. https://patterns.eecs.berkeley.edu/?page_id=42 (10 October, 2025).

MacGregor, Neil. 2011. *A History of the World in 100 Objects*. Penguin UK.

Mango, A. 2000. *Ataturk: The Biography of the Founder of Modern Turkey*. Overlook Press. https://books.google.com.au/books?id=T5QlAQAAMAAJ.

Martin, R.C. 2009. *Clean Code: A Handbook of Agile Software Craftsmanship* (Robert C. Martin series). Prentice Hall. https://books.google.com.au/books?id=hjEFCAAAQBAJ.

Marx, Leo. 1997. "Technology": The Emergence of a Hazardous Concept. *Social Research* 64(3). Publisher: The New School, 965–988. https://www.jstor.org/stable/40971194 (15 July, 2022).

Meyer, Bertrand. 1990. *Introduction to the theory of programming languages*. Prentice-Hall, Inc.

Michaelson, Carol. 2010. *BBC - A History of the World - Object : Chinese bronze bell*. https://www.bbc.co.uk/ahistoryoftheworld/objects/Fh59-tI4TViuv1RSsx0qhQ (24 September, 2025).

Mullaney, Thomas S. 2017. *The Chinese Typewriter: A History*. MIT Press.

Oakeshott, Michael. 1977. Rationalism in politics and other essays.

Paumgarten, Nick. 2012. Magic Mountain. en-US. *The New Yorker*. Section: the world of power. https://www.newyorker.com/magazine/2012/03/05/magic-mountain-davos (26 October, 2025).

Postel, Jon. 1980. *DoD Standard Transmission Control Protocol*. Tech. rep. 761. IETF. https://www.rfc-editor.org/rfc/rfc761.

Pu, Songling. 1915. *Combined plate illustrating A Bao* 阿寶 *and Wa qu* 蛙曲. 详注聊斋志异图咏. Wikimedia Commons. Plate title cut as 蛙田; 曲 apparently miscut as 田. Public domain. https://commons.wikimedia.org/wiki/File:SSID-12366089_%E8%A9%B3%E8%A8%BB%E8%81%8A%E9%BD%8B%E5%BF%97%E7%95%B0%E5%9C%96%E8%A9%A0_%E5%8D%B71-2.pdf (29 July, 2026).

Qi. 2025. Fear. In *The Codeless Code*. https : / / thecodelesscode . com / case / 33 (11 September, 2025).

Said, Edward W. 2003. *Orientalism*. Penguin Classics.

Sassaman, Len, Meredith L. Patterson & Sergey Bratus. 2012. A Patch for Postel's Robustness Principle. *IEEE Security & Privacy* 10(2). 87–91.

Scott, James C. 2020. *Seeing like a state: How certain schemes to improve the human condition have failed*. Yale University Press.

Tarko, Vlad. 2006. *The Metaphysics of Object-Oriented Programming*. https://news.softpedia.com/news/The-Metaphysics-of-Object-Oriented-Programming-24906.shtml (28 July, 2026).

Todd, Gary. 2007. *Chinese movable typecase, 1313 CE*. Wikimedia Commons. Wang Zhen's revolving typecase from the agricultural book *Nong Shu*, reproduced from Philip B. Meggs, *A History of Graphic Design* (John Wiley & Sons, 1998), p. 26. Public domain (PD-old / CC0). https://commons.wikimedia.org/wiki/File:Chinese_movable_type_1313-ce.png (28 July, 2026).

Wittgenstein, Ludwig. 2001. *Tractatus Logico Philosophicus*. Trans. by David Francis Pears & Brian McGuinness. USA: Routledge.

# Confucian Software

Confucius never touched a computer, but he knew quite a lot about software. He has much to say about problems of knowledge and information, as well as right action under ambiguity, threats and constraints. Read as a set of commentaries on *The Analects*, these essays treat Confucius as an early theorist of the flow of information through rites, names and rules; the cyborg sage tangled in the half-built machine of the feudal state.