01_ // ereignis

event.listen();

Höre ein einzelnes Ereignis vollständig. Benenne seine Funktion noch nicht.

>> observe(event)

02_ // folge

second_order();

Höre nicht nur, was ein Eingriff auslöst, sondern was das Ausgelöste anschließend verändert.

>> trace(consequence)

03_ // eigenzeit

keep_time();

Lass zwei gekoppelte Prozesse ihre unterschiedlichen Zeiten behalten.

>> preserve(times)

04_ // kopplung

connect(one);

Verbinde zwei eigenständige Prozesse zunächst nur an einer Stelle.

>> limit(coupling)

05_ // beziehung

relation_gain();

Verändere nicht den Klang, sondern die Stärke einer Beziehung.

>> modulate(link)

06_ // topologie

fade_link();

Lass eine bestehende Verbindung allmählich unwirksam werden, ohne sie zu entfernen.

>> topology(functional)

07_ // instabilität

dont_fix();

Wenn ein Zustand instabil wird, stabilisiere ihn nicht sofort.

>> observe(instability)

08_ // gewohnheit

break_prediction();

Sobald du deine nächste Handlung sicher voraussagen kannst, führe sie nicht aus.

>> interrupt(habit)

09_ // möglichkeit

let_go();

Lass eine Möglichkeit verschwinden, ohne sie sofort wiederherzustellen.

>> allow(loss)

10_ // beobachtung

name_sensor();

Benenne, welche Eigenschaft das System technisch beobachtet.

>> define(observation)

11_ // zwei hörer

dual_listen();

Höre dasselbe Signal als Mensch und als technische Analyse.

>> compare(listening)

12_ // form

future_space();

Ein Ereignis wird formal wichtig, wenn es verändert, was später möglich ist.

>> transform(possible)

13_ // beginn

before_name();

Höre ein Ereignis, bevor du entscheidest, welche Funktion es im System besitzt.

>> delay(category)

14_ // verlauf

follow_shape();

Verfolge Beginn, Wachstum, Umschlag und Ende eines Ereignisses. Greife erst danach ein.

>> wait(complete)

15_ // differenz

compare_return();

Warte auf ein ähnliches Ereignis. Höre nur die Differenz zwischen beiden.

>> listen(delta)

16_ // kleine ursache

change(min);

Verändere einen Wert möglichst wenig. Beobachte, ob daraus ein anderes Regime entsteht.

>> test(threshold)

17_ // ende

no_continue();

Lass ein Ereignis enden, ohne es zu bestätigen, zu wiederholen oder zu beantworten.

>> release(event)

18_ // späte wirkung

delay_effect();

Setze einen einzigen Eingriff. Warte, bis seine Wirkung nicht mehr eindeutig zuzuordnen ist.

>> trace(late)

19_ // nachwirkung

new_state();

Wenn ein Ereignis endet, behandle die veränderte Situation als neues Ausgangsmaterial.

>> state(next)

20_ // keine zäsur

dont_reset();

Beende oder verändere nicht alle Schichten gleichzeitig.

>> preserve(layer)

21_ // langsame ursache

slow_condition();

Nutze einen langsamen Verlauf als Bedingung schneller Ereignisse, nicht nur als hörbare Modulation.

>> condition(fast)

22_ // rollenwechsel

invert_role();

Lass eine Verbindung, die zuerst stabilisiert, später destabilisieren.

>> switch(function)

23_ // widerstand

stay_with();

Wenn das System Widerstand erzeugt, vereinfache deine Handlung nicht automatisch.

>> keep(resistance)

24_ // nicht eingreifen

withhold();

Höre einen Zustand so lange, bis das Nicht-Eingreifen selbst eine Entscheidung wird.

>> silence.active

25_ // nicht ausgleichen

no_compensate();

Wenn das System dichter wird, werde nicht automatisch leiser. Wenn es leiser wird, öffne nicht automatisch mehr Raum.

>> resist(balance)

26_ // möglichkeit

keep_condition();

Wiederhole keine konkrete Gestalt. Erhalte nur die Bedingung, unter der Ähnliches entstehen kann.

>> repeat(condition)

27_ // verengung

narrow();

Lass den Möglichkeitsraum für eine begrenzte Zeit kleiner werden. Öffne ihn nicht vorschnell wieder.

>> hold(limit)

28_ // erschöpfung

exhaust();

Verfolge eine Möglichkeit, bis sie ihre Wirkung verliert. Beende sie nicht nur aus Angst vor Wiederholung.

>> continue(until)

29_ // blinder fleck

blind_spot();

Benenne eine klanglich wichtige Eigenschaft, die das System technisch nicht beobachten kann.

>> mark(absence)

30_ // zeitskala

observe_slow();

Verändere nicht die Signalquelle, sondern nur die Geschwindigkeit ihrer Beobachtung.

>> scale(time)

31_ // unabhängigkeit

one_link();

Kopple zwei Prozesse an einer Stelle und höre, was sie unabhängig voneinander behalten.

>> listen(independent)

32_ // gegenregelung

invert_feedback();

Wenn ein Prozess zunimmt, schwäche seine Wirkung auf den anderen. Höre, was sich organisiert.

>> regulate(link)

33_ // freie möglichkeit

leave_open();

Wenn eine neue Möglichkeit entsteht, besetze sie nicht sofort.

>> space(possible)

34_ // störung

perturb();

Setze einen Impuls und beobachte seine längerfristigen Folgen, statt weiter zu steuern.

>> observe(after)

35_ // orientierung

learn_tendency();

Wiederhole eine Handlung nur, wenn ihre Folge nicht vollständig feststeht, aber unterscheidbar bleibt.

>> uncertainty(usable)

36_ // formfolge

change_future();

Beurteile einen Eingriff danach, welche späteren Möglichkeiten er öffnet, verengt oder verschwinden lässt.

>> evaluate(future)