HACKING THE ESSENCE OF SOFTWARE

ENCODING AND DECODING TURING MACHINES AND META-MACHINES

Kent D. Palmer, Ph.D.

Orange CA 92856 USA
714-633-9508
kent@palmer.name
http://kdp.me

Started 6/9/2012; Version 0.11; 8/12/2012; th01a11.doc
Cut down for CSER2013 Conference 2012.09.29
Corrected for CSER2013 2012.10.01 as hes01a04.doc, edited 2013.06.27
http://orcid.org/0000-0002-5298-4422

Keywords: state machines, Turing machines, universal Turing machines, software essence, agility, lean, hacking, C.S. Peirce, B. Fuller.

Abstract

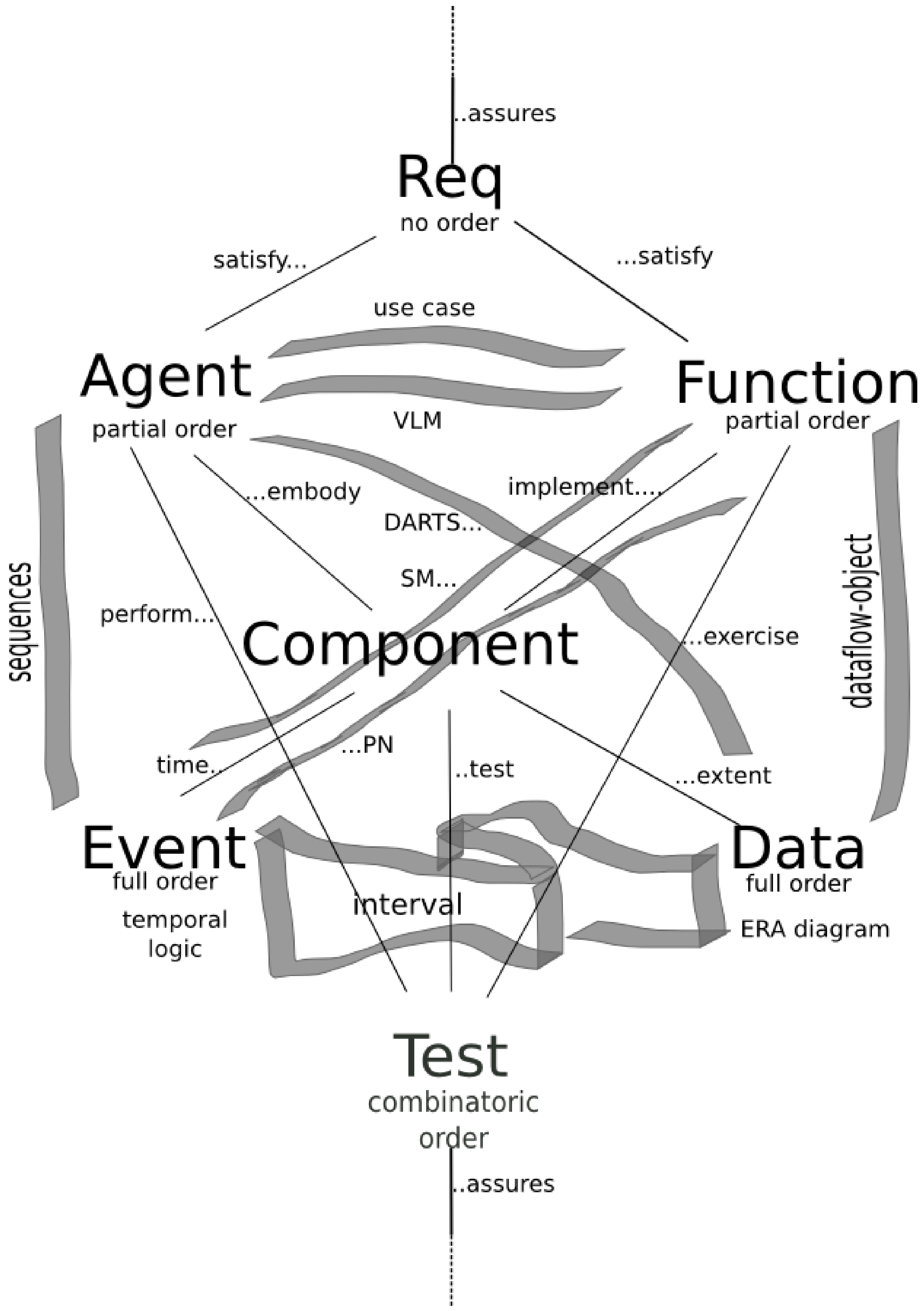
This paper is a condensation of second part of a monograph called "Tangled Hierarchies" which has been cut down a briefer version for publication. The first part of the paper concerned the way in which tangled hierarchies might be used to model the design of systems, and perhaps give us a way to show the consistency of domain specific design languages like the Integral Software Engineering Methodology (ISEM) first described in Wild Software Meta-systems[1]. In a subsequent paper "Reworking the Integral System Engineering Method Domain Specific Languages" at CSER11 the original language was expanded from 750 to 1700 some statement templates based on research into General Schemas Theory in the dissertation of the author Emergent Design[2]. This further part of the monograph looks again at the core of the realtime minimal methods which is the State Machine along with its dual the Petrinet and attempts to look at them in a new way based on the ideas of C.S. Peirce which are used to re-understand the notion of the Turing Machine. In order to understand Software in its essence we must return to well-worn concepts again and again and attempt to comprehend them more deeply. There are perhaps more secrets for them to give up and sometimes what seems so familiar and commonplace have aspects that are not recognized due to the assumptions we make about them that are unwarranted. C.S. Peirce was the greatest American Philosopher, but is hardly known in many circles where is ideas on Pragmaticism have not been fully appreciated. It could be that his work could give us a new perspective on the Turing Machine and within it the state machine if we applied the principles that he developed in his philosophy and semiotics to the Turing Machine to comprehend it in a new way.

[1] See http://works.bepress.com/kent_palmer
[2] See http://about.me/emergentdesign

Hacking The Essence of Software

It is clear that the core of software from a theoretical perspective is the Turing Machine. And since this is the Turing centenary we should focus to some extent on this core and attempt to understand it with respect to the problem we are attempting to solve which is how to represent the design of software, and how to show that Design DSLs are para-consistent and para-complete and para-clear. What has been suggested so far is that there are hierarchies of entities in the related to requirements, agents, functions, events, data, and tests and that these become tangled in a component hierarchy, and it is the relations and actions in this tangled hierarchy which is the means of establishing the para-characteristics (clear, complete, consistent, verifiable, valid, and coherent) related to the aspects (real, true, identical, present) of Being. We note that the component hierarchy appears in the hole in the lattice of the methodological distinctions and between the dual measures of linear order without distance and Partial Order with distance. These two types of order stand as a way of talking about delocalization and decoherence as well as the basis for the structure of the dualities of the minimal methods. So for instance linear order without distance is a description of the Code we create when programming. Each statement in the code is in a linear order, but the performance distance between each statement is unknown. We have to make external measures of performance to determine that distance, and because there can be context switching between statements it is not sure that a given statement will actually be executed immediately after the next when considered from the point of view of actual timing, we just know that if it executes at all it will be sometime later. The other type of order, partial order with distance says that programming constructs that are being executed are not necessarily ordered in a strictly linear way, and may if say they are being executed in different tasks, or by different hardware infrastructures be such that they will actually execute in different orders during different runs but the effects may be measureable on some background variable such as time or memory or population (which are the sources of the hierarchies that represent the viewpoints on realtime systems in Klir).

Both of these ordering types can have many different relations to each other and the result of that is decoherence and delocalization that occurs in actual programmed, so called hacked, code. The use of the term hacking highlights the very pragmatic nature of all our attempts to get things to work in spite of decoherence and delocalization. Delocalization reminds us that references to design elements may be spread out or smeared out in the code as a linear static organization. Decoherence reminds us that just because something is near to something else in the code it does not mean it will be executed in a way that we expect, discontinuities can occur between statements when they are reduced to assembly code and executed, and the fact that there are multiple hardware processing elements, or multiple tasks may mean that the order of actual execution is not set, even though it can be measured so that we are dealing with probabilities and not determinate results when we are talking about executing masses of binary executables. A lot happens when code is compiled and what that compiled code does in actuality may be different from what we expected it to do when we encoded it. In a sense the actual result of running code must be used to decode what we encoded. So there is a continual interpretative process that occurs as we write, execute and debug the code we are hacking. To say that code is decoherent and delocalized is merely to say that it represents an example of what Merleau-Ponty called Hyper Being, or what Derrida called DifferAnce, or what Heidegger called ~~Being~~ crossed out. The pointers and accumulators in the hardware represent what Heidegger calls Present-At-Hand (pointing) and Ready-To-Hand (grasping) in Being and Time and which are interpreted psychologically in Phenomenology of Perception. Software is the only cultural artifact that has as its essence Hyper Being or differAnce. DifferAnce means differing and deferring all the time, which Paul Simon calls “slip-sliding away” and which Plato in the Timaeus called the Third Kind of Being. Software is something written that executes and as it does so it can rewrite itself. And in fact it is precisely a machine that can rewrite its own tape, and ultimately its own program, that we call a Turing machine. And there are two types of Turing machines, the normal ones and Universal ones that run other Turing machines which we now call operating systems. But we should really call them operating [meta-]systems, because they go beyond the system of the Turing machine and are actually models of its environment. The dual of differing and deferring of software is effectiveness (agile) and efficiency (lean) which together give us efficacy. Software can differ from itself as it institutes differences and its execution can be measured in terms of its effectiveness and efficiency. And as humans that produce software as an allopoietic product, we can be lean and agile in that production process, which together make up what Reinertsen calls Flow. What we are saying is that there is some mirroring between the software product and the software development process that we need to take into account when we consider it as a pragmatic human activity. These computing infrastructures we build are far from autonomic, and so we have to build them piece by piece and then we have to maintain them for them to continue to work, and they are very fragile, and that is why the testing needs to explore as much as it can the combinatoric order of testing possibilities in order to assure robustness to the extent we can. But combinatorics are so vast that we need special ingenuity to make sure that systems are well tested, because it is many times impossible to test all paths beforehand. So that means we really need to understand the nature of software in order to produce good software. Part of that is understanding each of the minimal methods that can be generated out of the duality between the two orders as they are projected as bridges between viewpoints on the realtime system.


..assures
Req
no order
satisfy...
...satisfy
use case
Agent
partial order
VLM
Function
partial order
sequences
...embody
implement....
DARTS...
SM...
perform...
Component
...exercise
dataflow-object
time..
...PN
..test
...extent
Event
full order
temporal
logic
interval
Data
full order
ERA diagram
Test
combinatoric
order
..assures

So now in order to try to extend our understanding of the Turing machines and State machines that are the core of Software, i.e. the place where Design meets Programming, which in its pragmatic aspect can be called hacking which seeks ultra-efficacy in the development of working software as seen in the Agile paradigm that emphasizes hyper-effectivity and the lean extension that emphasizes hyper-efficiency. A key question in this regard comes up when we try to understand the nature of Peirce's Firsts (isolate, points), Seconds (relata, lines) and Thirds (continua, surfaces) are in relation to the Turing machine and its state machine. A state machine plus a list or queue or tape is a minimal Turing machine. When we look at a state machine we see that it is normally a set of vectors composed of input, entry state, output, exit state. These can be expressed as rules ***if*** *input and current state* ***then*** *function producing output and new state*. Gurevich showed that we can use rules such as these to describe any system at any level of abstraction and it would be Turing equivalent. Thus it is not necessary to reduce something to a Turing machine to show it is computable. And Computability reduces to knowing the causality running from outputs back to inputs through the system and knowing that all those threads of functionality are complete and consistent and are well-formed. So that means we can abstract from low level Turing machines and just use the Gurevich Abstract State Machine method as our representation of the process of computing. Executing software ultimately reduces in its essence to one syntactic construct which is the if…then… statement. Execution of software means to execute rules. All software can be represented as a stepwise refinement of rulesets from any level of abstraction down to the level at which the rule can be represented in a general purpose programming language. And we can use the Pieter Wisse's Metapattern method to understand how to derive the objects that the rules are referring to at the various levels of abstraction. Guervich ASM and Wisse Metapattern methods are duals of each other in this regard, one giving the causal structure and the other the contextual basis for the identification of objects based on their different behaviors in different situations, which amounts to the identification of discontinuities in the identity and the behavioral response of objects.

Now what we notice is that actually no matter how many inputs we have, and no matter how many outputs we have, there is a three way relation between the inputs, the outputs and the states, and that the two mentions of the state, i.e. the self-reference of the state machine providing a pivot of identity still makes only a third element. And this is related to what Peirce calls the structure of the sign. A semiotic relation is a Third, or continua that is an object, and interpretant and the sign itself. In this case we have the object as seen in the input, and we have the interpretation as seen in the output, and we have the sign in the state which is transitioning within the state machine that produces an algorithm that converts from input to output based on state. The transformation between input and output is performed by a function. Agency is represented by the infrastructure that is performing that computation. For instance we might have the same statement performed in different tasks or on different hardware platforms, and thus they can be performed in parallel. It turns out that if you represent a simulation of a system with a refinement of the Gurevich ASMs all the way down to the code what results is very inefficient, even though it may be functionally effective. So, performance improvement comes from introducing architecture which usually means distributing the functionality among various agents, i.e. into tasks or among processors in a distributed system.

The state machine is in fact made up of a three way relation between inputs, outputs and states. There is a triangular surface that connects these three elements and we call that a Third or a continua. It is what Steven Wallis calls a robust theory[3]. One way to see a state machine is to think of it as having anchors of functions between input and output, but that it changes the

[3] From Reductive to Robust: Seeking the Core of Complex Adaptive Systems Theory Steven E. Wallis 2008 DOI: 10.4018/978-1-59904-717-1.ch001 http://www.igi-global.com/chapter/intelligent-complex-adaptive-systems/24182

functionality, based on its state and thus providing a different layering surface to the state machine triangles. Data from input to output will flow a certain way until there is a state change, in which case it will flow differently in dataflow systems. States change transformations from inputs to outputs, but this can be seen as a three way semiotic relation with different computational surfaces being actualized giving the state machine an identity as a single machine as it executes on various input data transforming it into different output data based on the state of the system. Now since this surface can be represented as a rule we will call the surface itself the Rule. The arrow of functional transformation of inputs to outputs is complemented by an arrow from input to state, and from state to output as the state machine determines its own state for the next input session. The rule is a surface, and its boundaries are the functional transformation, the if part (left hand side) queries the state to determine the function, and the then part (right hand side) that sets its own state for the next round of inputs. We can then see that the data of input, output and state are the discrete isolate of the First, and that the function, and the self-querying of the *If S* and the response of the *Then S'* are the seconds or the relata that bound the surface of the rule R. Rules are surfaces or continua. I think this is a new way to look at them in terms of Peircian principles that I have not seen in the literature yet. It also shows why states are signs, and that state machines are semiotic machines. They take in objects (inputs) and they use signs (states) to interpret them giving outputs via functions. If we understand that state machines are semiotic machines then I think it clarifies why we call their production encoding and the interpretation of their execution results decoding. And this way of looking at it probably came directly from Turing's working on codes during the Second World War. When we are coding we are setting up a sign system and that involves taking in information and transforming it and putting out our interpretations along with the product of the computations. The internal state of the state machine is what gives it an identity. It is the identity that is preserved by the various rules that make up the machine as its sensing-action vectors. It is sensing what is present, i.e. the inputs it is given. When we combine the aspect of presence and identity with truth we get a formal system. A formal system has the properties of completeness, consistency, and clarity (wellformedness). The rule set of the state machine (its vectors as a set) need to be consistent and complete for the state machine to function properly. Wellformedness comes from the fact that all the vectors are expressed in rules. The lowest level of Truth with respect to Pure Being is verifiability. That means that we can compare the reality of the results of execution to the statements themselves and show that the statements do in fact express what the machine does. So the truth of the state machine has to do with the gist of the statements and their mutual interoperability and the wholeness of their organization indicating a singular unified totality, i.e. a synthesis, which is complete (Truth related to Presence) and consistent (Truth related to Identity) and clarity (Presence related to Identity).

Now we know in the Turing Machine that the state machine is related to a tape, and that the tape is a series of places with symbols in them. The Turing machine takes in the symbols and produces other symbols. There is a pointer that indicates what place with a symbol that we are talking about at any given time. This is called the tape pointer. Tapes are finite on one end and infinite on the other end in the original conception of the Turing machine so that it can handle infinite computation. The tape is an extent and this is the representation of space. The pointed to symbol is a gestalt on the background of all the other symbols on the tape. Now the input and output for the state machine comes from the tape. So the symbol on the tape is a fourth entity producing a minimal system with the input, output and state. So what we need to explore is what this fourth entity gives us beyond the state machine. Since the other entities make up the formal system of the state machine, then we would expect the symbols on the tape to stand in for reality. Reality is related to the other aspects of Being by giving us verifiability, validity, and coherence characteristics. The state machine can read and write symbols to and from the tape. So there is a directional line from the tape to the inputs and from the outputs to the tape related to read and write operations. This creates another triangle which is composed of read, function, write operations. The focus of the read and write is where the tape pointer is pointing at any given time, and this is the point in the extent where timing occurs. In other words the pointed to cell becomes a spacetime nexus within the worldline of some agent. The surface that is defined by the triangle of input, output and symbol in place on tape (gestalt) with the read, function write directional relata (arrows) is an interactive flow.

Once we have defined another surface which relates the state machine to its tape which also relates the formal system to reality and thus generates significance. We can verify the statements of the state machine against the tape by watching what is written to the tape. And we can validate the state machine by looking at the results of the execution of the state machine through the results on the tape. By relating the state machine to the tape we also get coherence because the state machine state is an identity and that identity gets reflected back onto the tape though the outputs of the state machine operation which can be seen as coherent if it does what was intended and so we start to see agency in the coherence of the operation of the state machine as reflected on the tape. There are two other surfaces that related to this effect. The first is the surface related to reading input. Associated with this input is the state we see in the left had part of the rule and that is completed by an interpretation of the symbol on the tape that is the figure of the gestalt. This surface is hermeneutical. On the other hand there is a surface related to writing output which is associated with the right hand side of the rule and signifies intent. So interpretation takes the symbol as a sign of some significance and the intent gives a sign of some significance. Both of these semiotic characteristics are signs of agency, which is the dual of functionality. But the interesting thing is that there is a duality between ‘interpret’ and ‘intent’, while functionality is unified. The surface related to writing outputs and intent is causal. Now we have four surfaces rule, interactive flow, hermeneutics and causal intent (or affect) that are all what Peirce would call a third or a continua. Interactive Flows relates the state machine to the tape and thus relates it to spacetime creating a worldline of an agent through the controlled and organized operation of the functionality of the state machine as it relates to the contents of the tape which can contain either encoded data or algorithms. The organization of the state machine is seen in the relations of its rules to each other that reasserts its identity. So the State is related to identity and the Tape is related to Reality. Presence is related to inputs and truth is related to outputs. Inputs are what is present in input variables. Outputs are what show the organization of the system via the state machine that is true, where true means going in a straight line based on criteria that are used to determine that it is straight. So for instance any linear system is true. i.e. it is producing straight line output. All non-linear output is judged on the basis of the true coordinates, i.e. orthogonal straight lines.

This association of the isolate with the aspects of Being (tape=reality, state=identity, input=presence, output=truth) comes from the fact that the various surfaces (interaction, rule, hermeneutic, causal intent) intersect by threes.

Surfaces: Interaction, rule, hermeneutic = input isolata -> Presence aspect

Surfaces: Interaction, rule, causal intent = output isolata -> Truth aspect

Surfaces: Interaction, causal intent, hermeneutic = symbol on tape isolate -> Real aspect

Surfaces: Hermeneutic, causal intent, rule = state isolata -> Identity aspect

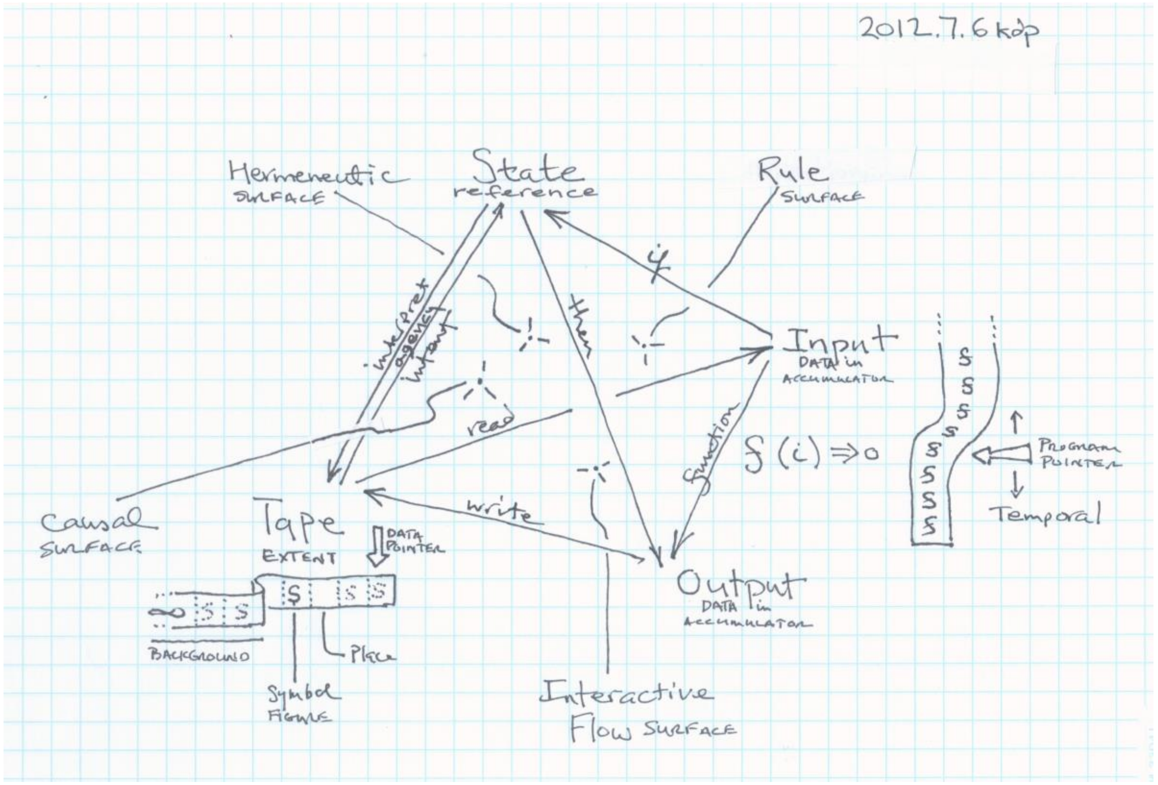
Similar things can be done by looking at relata:

Read, If clause, function = Input

Write, Then clause, function = Output

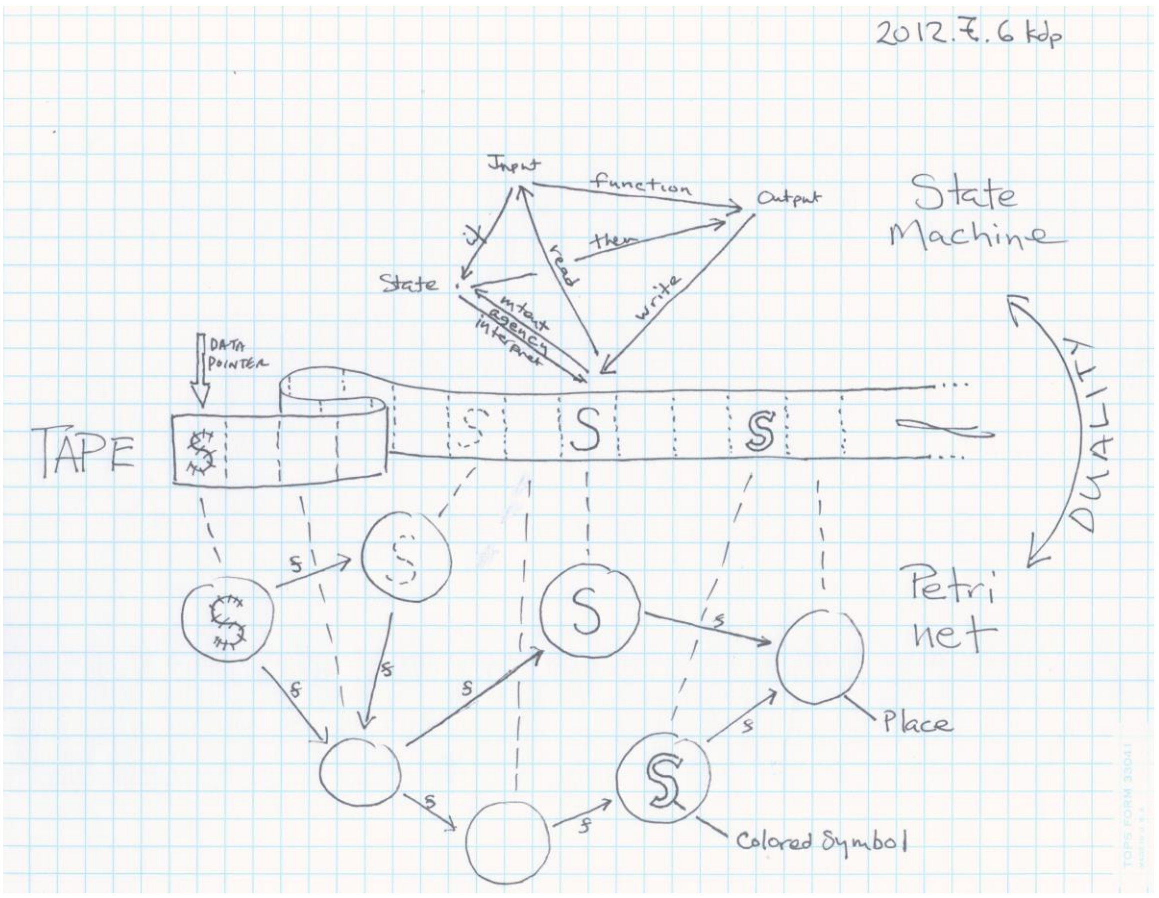
If clause, Then clause, semiotic = State

Read, Write, semiotic = Tape symbol

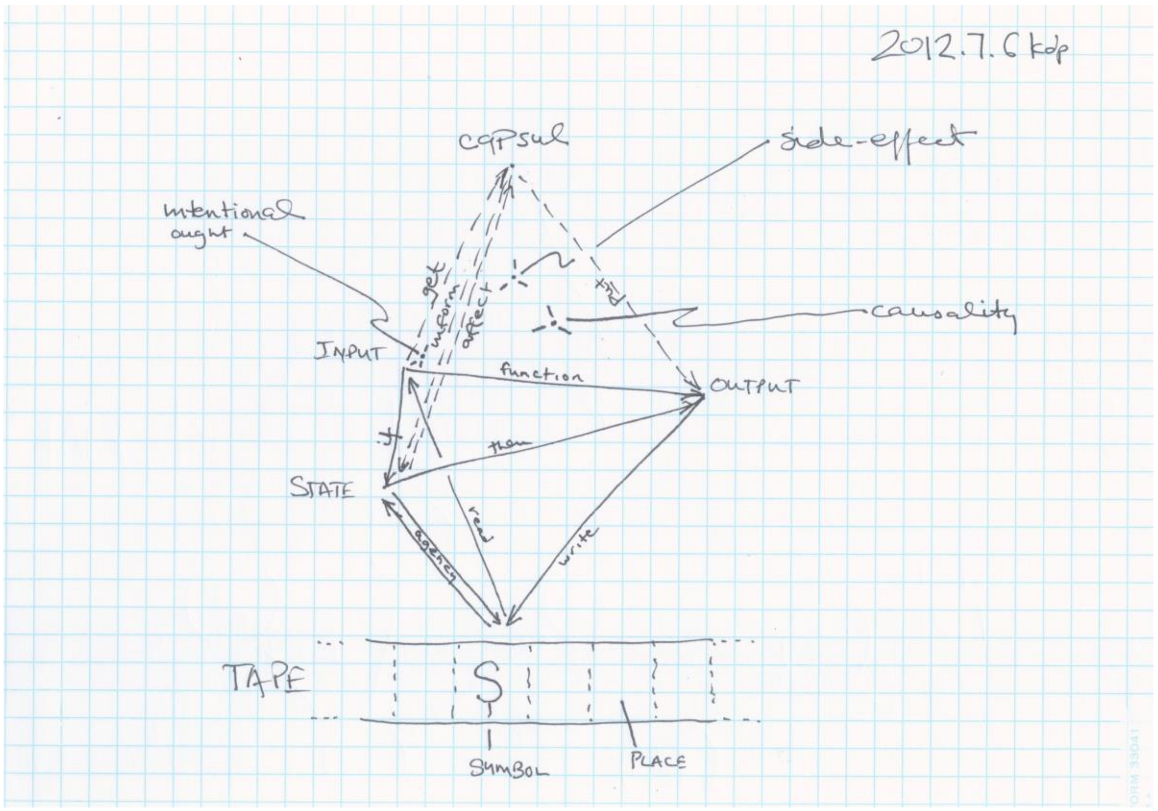


This is a minimal system as defined by B. Fuller. All the elements are informed by all the others diacritically. It is a tetrahedron of concepts composed of four isolates (symbol, state, input, output), six relata (function, read, write, if clause, then clause, semiotic), and continua (rule, hermeneutic, causal intent, interactive flow). Now what is surprising about this extension of the concept of the Turing Machine is that it is semiotic and thus connects directly with Peirce's idea of semiotics as a threefold relation. In it Rules as a surface mediates between the hermeneutic surface and the causal surface. Both of these surfaces are based on and define the surface of interactive flow, which is the basis for positing the gestalt of the symbol on the ground of the whole tape. It also produces a double bridged line of agency existing in a tension between interpreting (taking for a sign) and causal intent (giving a sign). This double action of the agency is the dual of the orthogonal line of function, which is also a method for objects. It is interesting that the agent line is composed of two oneway bridges while the function is a single oneway bridge. There are various compositions of directional arrows bounding each surface. All the isolata are variables of different kinds. All of the lines are directional. Two of the surfaces form circuits around their parameters. The oneway arrows of the function and the clauses of the rules forms a circuit with the tape. It is the dynamism of the tape that allows the machine to work. The state machine itself is reactive. The dual of the state machine is the petri net which is proactive.

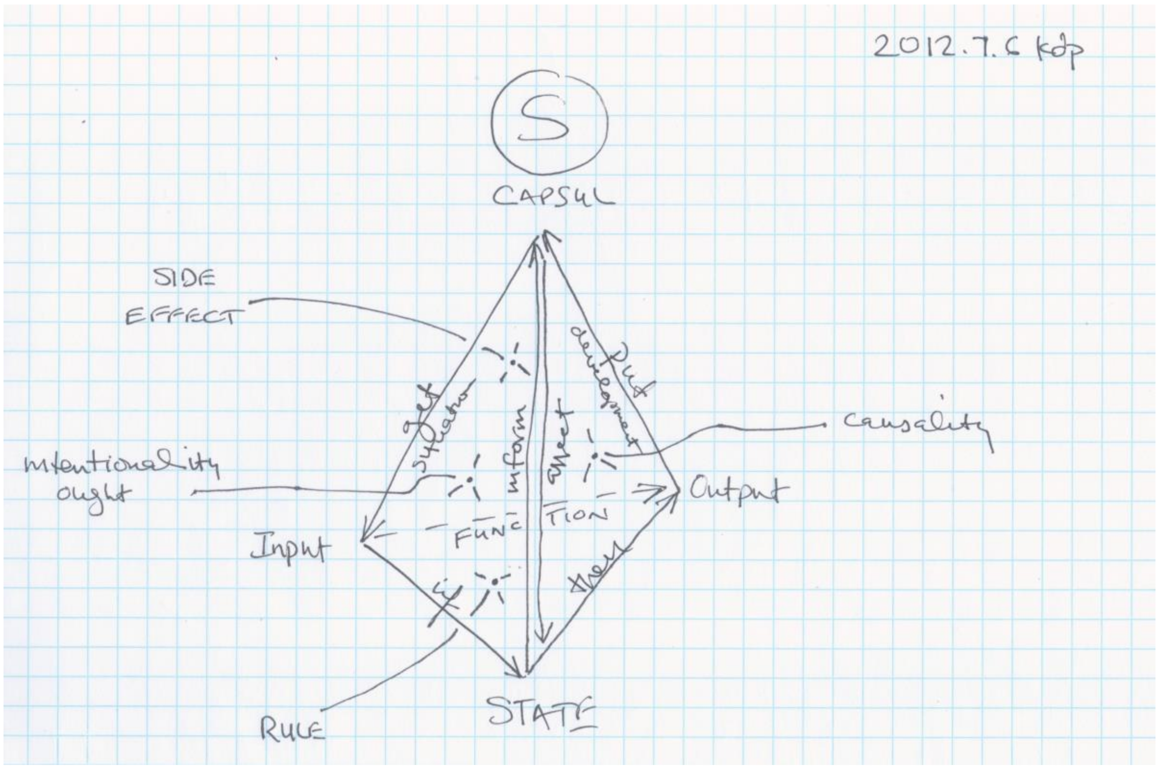
But also there are multiple petri net representations for a given state machine kernel. Petrinets are more proactive but also more superficial. State machines are condensed representations that are most efficient and effective. You can get this kind of proactive structure from two state machines that are interlocked each feeding the other. Colored Petrinets are better at exhibiting control structures that are self-starting. The colored Petrinets operate more like cellular automata using markers in places to activate transitions. Petri Nets look at function from the point of view of event, while State Machines look at events from the point of view of function. The event is a triggering of the transition when the marker is in the place and the function occurs in the transition. The colors of the markers are the inputs, and the colors of markers are the outputs too. State machines on the other hand transform what functions are called given monadic state identity operations whose differences can be used as a controller. In state machines function is central and in petri nets it is event that is central created by the marker being in a place, like the symbol is in the place on the tape. When we put together the petri net and state machine then we have the tape as active, and the symbols being triggering mechanisms to change the colors of the symbols. So the two dual mechanisms can both work together without interfering. The petri net merely colors the symbol. This is an autopoietic symbiotic relation between the two archetypes of computation.



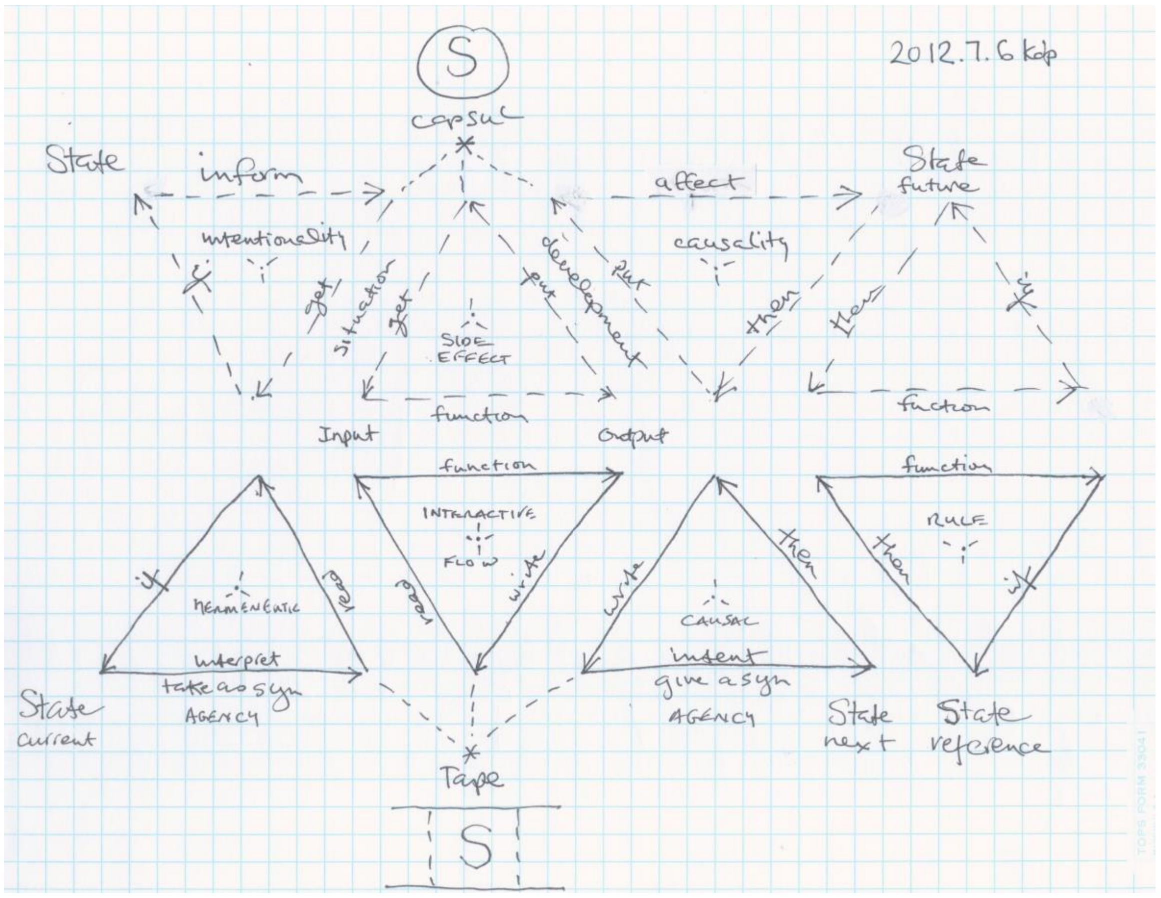
Once we have realized that the two archetypes of computing can coexist together manipulating and using the same tape in an autopoietic symbiotic relationship where one manipulates the symbol and the other manipulates the color of the symbol, one treats the symbol as an existent maker with color which causes transitions to fire, a firing transition is just a function that takes colored makers as input does a transformation with side effects and then places makers in their output places. On the other hand the State Machine treats the symbol as a character and reacts to its characteristics as a symbol which informs how the function will treat it as an input symbol which is read from the tape and then an output symbol is written back to the tape, perhaps after moving the data pointer backwards or forwards. The state machine has a direct relation to the place on the tape that is pointed to and it reads and writes symbols based on where the data pointer is pointing. The petri net on the other hand has an indirect relation to the tape where certain cells are treated as places into which existent markers are placed and these places form a network that is activated by the existence of a symbol in a place of a given color. So there is a superimposition of color on symbol such that the two computations can be separate yet indirectly interact. What would happen if you had such a computational setup is unknown as each assumes the stability of the tape but the petri net would be shuffling the symbols and tuning them different colors behind the scenes from the point of view of the state machine, and from the point of view of the petri net markers would be coming into and going out of existence suddenly. From the point of view of the state machine symbols would be appearing and disappearing. What is interesting about this is that Petri Nets are active and State Machines are passive and so they have completely different characters, and the Petri Net could act as the controller for the State Machine jump starting and boot strapping action by the State Machine. Also the Petri Net is better in modeling protocols than the state machine. So it could be that the petri net could act as the protocol between two state machines within the universal Turing machines that run separate Turing machines. This thought of the Universal Turing machine (meta-machines) takes us into the modeling of the meta-system by the addition of a capsule to the state machine minimal system to form its dual.



We will think of the capsule as the encapsulated data of an object, but we can also think of it as a functional programming monad. We do a get operation in order to take the contents from the capsule and we do a put operation in order to place new contents in the capsule. This is a side effect that is placed in the capsule or monad. The surface from surrounded by put, get and function should be thought of as the side-effect surface which is different from the rule surface or the interaction surface. Once we realize that there is another surface related to capsule side-effects then we must ask what the other two surfaces represent.



Interestingly the other two surfaces impinge on a line between State and Capsule similar to the line between State and Tape. This line is a two way bridge, so that there is one surface that is Get, If, and Inform, and another surface which is Then Put, and Affect. Let us call the Get, If, Inform surface Intentionality for the time being. Let us call the Then, Put, Affect surface Causality tentatively. We note that since the capsule can either be inside the object or outside the system as a monad it can be interpreted as either inward or outward, so we can think of the two way bridge of inform and affect as either Subjective or Objective depending on whether the capsule is inside or outside the system. So it is hard to interpret exactly what is the next higher thing from agency that is being revealed here but let us call it Dasein following Heidegger who was following Hegel. Dasein is the projective capability posited by Kant. Dasein informs and then affects, just like agency interprets and then intends. The informing cycle is related to the intentionality surface and the affective cycle is related to the causality surface. Intentionality and Causality stand over and against the side-effect surface.



These three new surfaces are meta-systemic, whether that meta-system is seen as within or outside the state machine system region. Systems nest and Meta-systems nest. But they also interleave in their nesting like Russian dolls where the dolls are the super-system, system, subsystem and the interspaces between the dolls are the meta-systems. Meta-systems are operating systems for applications and they are modeled as Universal Turing machines. A given meta-system can run multiple applications. Those applications are all state machines, that communicate with each other via protocols represented by the petri nets. The System as a bubble between higher and lower Meta-systems can see the meta-system as within or on the outside, and thus the capsule can be on either side, either within or on the outside. If it is on the outside then it is a monad. If it is on the inside then it is an object. So, monads and objects are duals.

The point of these musings is that I have long wondered how to apply Peirce's insights regarding continua or thirds to fundamental structures, and there is no more fundamental structure than the Turing Machine for Computer Science and Software Engineering. Gurevich generalized it so that we can take arbitrary levels of abstraction and see whether they are computationally and thus causally complete by expressing them as Rules. Here we see why this works which is because the rule is the surface circumscribed by If, Then and Function. Notice that the If and Then arrows on this surface are both go the same way as the function edge. This is an asymmetry within the structure. The other three surfaces are bounded by circuits of arrows, and the line that is opposite the function that stands for agency is a double bridge in order to allow these circuits to exist within the structure of the tetrahedron. I was thinking about Steven Wallis' idea of robustness which counts Newton's law and Ohm's law as robust theories, and I realized that state appears twice and that really the relation between input, output and state as a robust relation if we thought about state in terms of identity. And then I realized that all we needed was the Tape to have a Turing machine and that meant there was a minimal system. Between the tape and the input and output variables the interactive flows were defined between the state machine and the tape. The next step was to figure out the nature of the other two surfaces. Hermeneutical and Causal is what came to mind. One surface is involved in interpreting the tape, and the other surface is involved in reacting based on that interpretation. But it was surprising that agency was reflected in a dual bridge of interpret and causal intent (or affect). It is even more surprising that if we extend this to the meta-system beyond the Turing machine (the meta-machine) then we get something like Dasein and there are surfaces for intentionality and causality, which are opposite the side-effect surface. And this interpretation is forced on us by the fact that the capsule can either be seen as inside or outside the system, because meta-systems can be nested within or as environments outside the system. The interesting thing is how when we flesh out this robust structure we see higher concepts come into play like agency and Dasein where we do not expect them. We also see how double bridges arise as a result of asymmetries in the way that arrows are configured along the edges of the tetrahedral diamond. It is only the octahedron that has perfect flow along its arrowed edges, so there has to be asymmetries in the tetrahedral system.

The tetrahedrons we have uncovered are the three dimensional and thus related to a philosophical principle beyond those that Peirce adhered to which are fourths which signify synergy and fifths which signify integrity developed by B. Fuller in Synergetics. Synergy is the reuse of parts within a whole. We see that in the reuses of state to emphasize identity across the changes of state. Integrity is tensegrity which is flexible and inflexibility mixed to give resilience. We see then that the Turing machine and the Universal Turing meta-machine tetrahedral have synergy by the reuse of the rule surface, we also get reuse of the state variable within the rule and reuse of the agent and Dasein asymmetric paths by their doubling. The capsule gets reused because it can appear as an object inside or a monad outside the system. So there are many aspects of reuse showing synergy in the diamond of the Turing machine with capsule configuration that unites system with meta-system. Integrity specifically appears as the combination of replicated and non-replicated elements in the Turing meta-machine representation. Via repetition some give or dynamism is allowed in the structure that can allow it to be dynamic and thus give software the adaptability or resilience we find in its essence. This diamond is a picture of the essence of software and is founded on the ability of software to rewrite itself and thus on the differAnce of Derrida. By using Peirce to understand the essence of software anew we are in effect hacking the essence of software itself by changing our concept of it and reaching more deeply into what it means by using the principles of Peirce and Fuller to understand this unique cultural artifact that embodies Hyper Being and that is changing our world profoundly by its incorporation into all manner of devices that are in turn change the available affordances and thus transform our world.

Frederick P. Brooks, Jr. in his famous article on the Essence and Accidents in Software Engineering called "No Silver Bullet" identifies what he believes is the fundamental and essential characteristics of software which are Complexity, Conformity, Changeability, and Invisibility. Our new view of the diamond of the System and Meta-system interface between the tetrahedral of the Turing machine and the capsule that share the rule surface does not change any of these characteristics. But what it does is explain the structure of the building blocks that when put together in ingenious ways result in complexity, and have the ability to conform, and control changeability, and inform the invisibility of the conceptual and theoretical structure of software as well as the praxis producing source code that embodies that structure effectively and efficiently. Software only seems werewolf like because it appears alien to our conceptual apparatus. But Kant placed rules at the center of reason in his first and second Critiques. But he maintained in the third critique that there are no rules for formulating rules. And when we can put this together with the idea of Wilden that The Rules are No Game. Then we see at least three levels, that of the game, i.e. the rule governed activities, the rules themselves and that which produces the rules which escape representation by them. The essence of software points to the non-representability of software design a subject that I cover in my dissertation on Emergent Design[4]. The characteristics of software come from the relation of the theory of design to the delocalization and decoherence of the code as we attempt to play the game by the rules we make up as we attempt to continue to indicate the non-representables. The point made in Scrum is that we can always change the rules and thus get an emergent event that transforms the nature of the work we are doing and the means of achieving our goal. This is the pragmatic aspect of our play of the game in practice where we seek hyper efficiencies and effectivities and thus ultra-efficacy. Understanding the essence of software synthetically rather than analytically via the philosophical principles of Peirce and Fuller give us a better appreciation of how the various characteristics of the software essence interact to produce its intrinsic difficulty for which there is no Silver Bullet. Now we can think not just about variables within our source code and how they are algorithmically connected to each other, but we can think in terms of lines of flow, surfaces that are bounded by these flows, and the solids that bring together these surfaces into System and Meta-system spanning models. Thinking about the essence of software in this more elevated way should help us deal with the problems of decoherence and delocalization that make the essential characteristics of software intractable.

References

Börger, E, and Robert F. Stärk. Abstract State Machines: A Method for High-Level System Design and Analysis. Berlin: Springer, 2003

Brent, Joseph. Charles Sanders Peirce : A Life. Bloomington: Indiana University Press, 1993.

Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. Reading, Mass: Addison-Wesley Pub. Co, 1995.

Copeland, B. Jack. The Essential Turing. Oxford: Oxford University Press, 2002.

Derrida, Jacques. Of Grammatology. Baltimore: Johns Hopkins University Press, 1976.

Derrida, Jacques. Speech and Phenomena: And Other Essays on Husserl's Theory of Signs. Evanston: Northwestern University Press, 1973.

Derrida, Jacques. Writing and Difference. Chicago: University of Chicago Press, 1978.

Eco, Umberto, and Thomas A. Sebeok. The Sign of Three : Dupin, Holmes, Peirce. Advances in Semiotics. Bloomington: Indiana University Press, 1983.

Fowler, Martin. Domain-specific Languages. Upper Saddle River, NJ: Addison-Wesley, 2011.

Fuller, R B, and E J. Applewhite. Synergetics; Explorations in the Geometry of Thinking. New York: Macmillan, 1975

Gammon, Shauna C. A. Notions of Category Theory in Functional Programming. Vancouver: University of British Columbia, 2007.

Heidegger, Martin. Being and Time. New York: Harper, 1962.

Herken, Rolf. The Universal Turing Machine: A Half-Century Survey. Oxford: Oxford University Press, 1988.

Jensen, K, and Lars M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Dordrecht: Springer, 2009.

Klir, George J. Architecture of Systems Problem Solving. New York: Plenum Press, 1985

Kockelmans, Joseph J. Heidegger's "being and Time": The Analytic of Dasein As Fundamental Ontology. Washington, D.C: Center for Advanced Research in Phenomenology, 1989.

Langer, Monika M., and Maurice Merleau-Ponty. Merleau-Ponty's Phenomenology of Perception : A Guide and Commentary. Tallahassee; Gainesville, FL: Florida State University Press ; University Presses of Florida [distributor], 1989.

Laplante, Phillip A. Great Papers in Computer Science. New York: IEEE Press, 1996.

[4] http://about.me/emergentdesign

Leavitt, David. The Man Who Knew Too Much: Alan Turing and the Invention of the Computer. New York: W.W. Norton, 2006.

Leffingwell, Dean. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Upper Saddle River, NJ: Addison-Wesley, 2011.

Merleau-Ponty, Maurice. Phenomenology of Perception. New York: Humanities Press, 1962

Minsky, Marvin L. Computation: Finite and Infinite Machines. Englewood Cliffs, N.J: Prentice-Hall, 1967.

Motro, Ren. Tensegrity. London: Hermes Penton Science, 2003.

Peirce, Charles S., et al. Writings of Charles S. Peirce : A Chronological Edition. Bloomington: Indiana University Press, 1982.

Petzold, Charles. The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine. Indianapolis, IN: Wiley Pub, 2008.

Priest, Graham, Richard Sylvan, Jean Norman, and A I. Arruda. Paraconsistent Logic: Essays on the Inconsistent. München: Philosophia, 1989.

Pugh, Anthony. An Introduction to Tensegrity. Berkeley: University of California Press, 1976.

Reinertsen, Donald G. The Principles of Product Development Flow: Second Generation Lean Product Development. Redondo Beach, Calif: Celeritas, 2009.

Sallis, John. Chorology: On Beginning in Plato's Timaeus. Bloomington: Indiana University Press, 1999.

Swierstra, Wouter. A Functional Specification of Effects. University of Nottingham, 2009

The Question of Being, trans. William Kluback and Jean T. Wilde (New York: Twayne, 1958) and “On the Question of Being,” trans. William McNeill, in Martin Heidegger, Pathmarks, ed. William McNeil (Cambridge: Cambridge University Press, 1998). See http://www.counter-currents.com/2010/07/junger-heidegger-nihilism/

Turing, Alan M, and B J. Copeland. The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life, Plus the Secrets of Enigma. Oxford: Clarendon Press, 2004.

Wallis, Steven E. “Toward a Science of Metatheory” Integral Review, volume 6, #3, 73-120 http://www.doaj.org/doaj?func=abstract&id=590329

Wisse, Pieter. Metapattern: Context and Time in Information Models. Boston: Addison-Wesley, 2001