An Analogy: Software AI and Natural Language

by Steve Bucuvalas, CEO, Phase Change Software
January 2017

While recently rereading Ray Kurzweil's "How to Create a Mind," I realized that his discussion of Natural Language Processing technology (NLP) and its consequent economic impact, are close analogies to the technology and impact of Phase Change in the software domain.

My intention here is to briefly describe Phase Change's technology and its impact, using the concepts that Kurzweil eloquently develops in his book. His larger theme of creating a general artificial intelligence (AI) is interesting and perhaps relevant, but I will defer that to another time.

Software and Information Technology

Software is the critical ingredient in the economic explosion in the use and value of information technology. Even so, prior to Phase Change, software development and engineering have not themselves become an information technology. This subtle distinction may need some explanation to see the rather glaring incongruity. It can be clarified by looking at the recent history of Natural Language Processing (NLP).

Over the last 30 years, through the efforts of Ray Kurzweil and many others, we as an industry have brought human language:

- *from a state in which only humans could interpret and use the meaning of text and speech*
- *to a state in which AIs are interpreting and using the meaning of text and speech*

The impact is enormous, with some stunning examples:

- *we can talk to our smart phones, directing their actions, asking arbitrary questions, expecting definite answers*
- *a technology company has created a Jeopardy® playing AI, which read and understood the knowledge in Wikipedia, and beat the best human Jeopardy® players*

My core point here is this: AI interpretation of the meaning of text and speech is what made human language an information technology. Kurzweil's notion of the Law of Accelerating Returns (LOAR) certainly applies to NLP. Chatbots and other applications are exploding in all industries.

This has been achieved even though human language is fraught with ambiguity, hidden contextual references and social meaning – creating inaccuracies in interpretation in the current state of the art of NLP, which somewhat limits its utility.

Phase Change is doing the same with software – but with complete logical accuracy – since the handicaps inherent in the interpretation of human language by NLP are not present in the interpretation of programming languages. Prior to Phase Change, software could be interpreted only by humans. People needed to read software text composed of a formal programming language, and, with careful reading and thought, derive its meaning. Now Phase Change has created the capability for AIs to interpret the meaning of software.

And so, the LOAR can now be applied to software development and engineering. This makes our prediction of two orders of magnitude improvement in productivity look modest.

It is.

Software's Practical Problem

To solve a problem as hard as "interpreting the meaning of software," you must start with understanding the nature and essence of the problem. This is essential to the next step, which is applying the right formal and scientific tools.

Software's problem is one of practical development and engineering. The productivity and quality for the development process does not scale. To borrow the visual metaphor from Fred Brooks "The Mythical Man Month," it is stuck in a tar pit.

The break-through insight in the problem's essence is that we're developing practical physical computational systems, not abstract mental models of computation. This observation changes the broadly held notions of limits and constraints on software as an information medium. Physical systems are always finite, although it is sometimes useful to think of them as being infinite. In fact, and in substance, no physical computational system is infinite.

The implications of this observation are not obvious. Again, an example may help.

Consider the Turing Halting Problem: the analytical limits derived by the halting problem results cannot be validly applied to a physical software system. I want to emphasize: the previous sentence asserted Turing's limits "cannot be applied." Turing constructs a counter-example for which halting cannot be proven without paradox, but the "source code" of the counter-example is necessarily infinite in size. The counter-example literally cannot exist in a physical world. By the way, this is not a criticism of the legendary Alan Turing, his proof is totally correct in the world of abstract computation, and the results apply to abstract computation. It simply doesn't apply to solving software's practical problem. This is also true of the Halting Problem's intellectual derivative, Rice's Theorem. Rice's results similarly to not apply to physical computational systems.

In physical computation, the concept of decidability must be adjusted for a physical computational system. We are forced to use a different type of mathematics. With this understanding, that applied software and computation are physical systems, the choices in mathematics and science follow.

Formal Data Type

Phase Change turns the text of programs into a formal data type. Much as with "floating-point" or "int" we have formal operations that we use to manipulate the data type in algorithms. We call this data type a "formal software concept." One can think of these formal concepts as something akin to a design pattern, only operating on multiple levels of abstraction from "ground" program units to abstract design patterns and OO class concepts.

This formal software concept is not the text of the programming language, although there is an essential correspondence between source code and meaning. It is a physical symbolic structure that is the fundamental unit of software meaning. Phase Change extracts this normalized representation using compiler and program analysis technology.

Formal operations on software concepts use intuitionistic set theory, an application of abstract algebra, and probabilistic reasoning. Much like numeric operations on numeric data types, the concept operations are guaranteed to work, regardless of scale.

Again, we have an analogy to NLP. In NLP, the natural language text or speech corresponds to the meaning, but AI's capacity to understand is based on formal concepts, a separate and very non-obvious representation.

The Phase Change extraction techniques work for any programming language, and across different programming languages. And again, in parallel with NLP, there are a great many different human languages. But largely our minds internally understand language concepts "the same" regardless of the language spoken. Whether I'm speaking my native tongue or a foreign one, both languages map to common concepts in my mind.

Hierarchical Recursive Concepts

Much like the work in NLP processing, Phase Change organizes the formal concepts into hierarchical structures, which are probabilistically linked: horizontally and vertically. Much like the human neocortex, the concepts become more abstract as one moves up the hierarchy.

Thus, at the ground-level one can ask (using a NLP chatbot) a question about the concrete use-cases in a software application. But one can also ask questions at a more abstract level, about a class of software applications without regard to implementation, for example, "what is a banking application?"

The probabilistic reasoning generally uses the Bayesian approach, in which the network topology is specific to the applied software development domain. We believe we've optimized Ray Kurzweil's concept of "modeling at the right level."

Not to belabor the analogy, but NLP's success stems from this same ability to create hierarchical representation of patterns between phonemes, words, and concepts.

Big Understanding, Big Data

The AI Jeopardy® champion became dominant because its creators directed it to absorb the machine-readable natural language text in Wikipedia and other sources. Its stunning competence stemmed from its ability to transform that text into a massive conceptual structure. Not surprisingly, it is a conceptual structure of hierarchical recursive concepts.

Similarly, Phase Change technology's approach increases in competence and value as the scale of its knowledge increases. It works on individual programs, but it's value will increase when it can absorb applications and portfolios on a company and industry scale.

Obviously – and fortunately – this body of source code is already machine readable, and is much larger and more economically significant than Wikipedia.

Equally obvious, much like human language is partitioned into many individual languages, so is software partitioned into many programming languages. Fortunately, a few dominant programming languages account for the vast majority of economically important software.

Creating the formal software concepts for software applications is the threshold to Kurzweil‘s Law of Accelerating Returns, and our prediction of at least two orders of magnitude productivity improvement. The effect of recognizing the inherent concepts and abstractions in a single commercial application (whether a banking system or a video

game) will change the game for the corresponding software teams' engineers, and catalyze these startling productivity improvements.

Contact us for more information.
Phase Change Software LLC
http://phasechange.ai
info@phasechange.ai
(303) 586-8900