The 2014 International Conference on Systematic Innovation

July 16-18, 2014, San Jose, the USA

Systematic Innovation for Computing

David W. Conley

Innomation Corp - President, PQR Group - Managing Partner

E-mails:

David@Innomationcorp.com David@TRIZPQRGroup.com

Abstract

There have been a variety of applications of TRIZ and Systematic Innovation (SI) techniques to the disciplines of computing and software development. When these computing system analyses are directed specifically towards a system's computing hardware it is a somewhat straight forward endeavor of standard TRIZ and systematic innovation employed for electro-mechanical/thermal systems. However, the application of TRIZ and systematic innovation to only the software sub-component of a computing system is not only problematic but also fundamentally misguided in its rationale. As stated, software is simply a single component, or multiple components, in a larger system which most likely serves as a control system for some even greater physical system. Granted the software component is unique in that it has virtual characteristics, but the software/coding is a component none the less. One of the fundamentals of TRIZ is that it helps to identify and resolve contradictory requirements between system components. TRIZ is not effective when applied to individual components or when visibility to the larger system is not available to shed light on the functional requirements of the components.

Unfortunately many of the attempts to apply TRIZ and systematic innovation to software have been met with less than desirable results. The lackluster outcomes are a consequence of two misunderstandings: 1.) that the non-tangible software is a functional tool in and of itself and 2.) that it is sensible to improve the design/functionality of a single component operating within a larger system by analyzing that component as if it were a standalone entity. Attempting to improve a piece of coding by analyzing it in and of itself is analogous to attempting to improve a musical score without any thought or attention to the musical instruments (at the micro level), or the orchestra (at the macro level), that the music has been written for. While syntax changes may very well result in some improvements in the software/code operation or effectiveness (an optimization attempt), the gains over the functionality of the basic code (assumed to be already operational) will be limited. In fact, the only way to effectively improve a piece of coding, as far as its ability to substantially improve its affect as to the operation of the system to which it is assigned, is to understand the interface and interaction of the coding within the overall system. Only then can meaningful changes be made to the coding based on understanding the needs of the larger system. This analysis can never be accomplished based solely on the examination of the software itself.

The paper will describes a systems engineering approach to analyzing and improving computing systems which sometimes requires coding changes and which sometime can be improved by changes to other portions of the overall control system.

Keywords: Software, Computing, Innovation, Systems Engineering, Control Systems

.

1. Introduction

According to the Gartner Worldwide IT Spending Forecast "...worldwide dollar-valued IT spending will grow 3.2% in 2014..., reaching $3.8 trillion as the world economy gradually recovers."[1] This level of spending signifies that IT is seen as a crucial component in most every commercial and government operation worldwide. It is no wonder that businesses are looking for ways to innovate in the software development space. As a result of the pressure to improve coding development and effectiveness many have attempted to apply TRIZ and Systematic Innovation techniques to the discipline of programming, usually with lackluster results. The primary reason for the less than desirable affect of TRIZ/systematic innovation on the world of syntax is the shortsighted focus of looking only at the software. In fact, most everyone who has ever asked me about the application of TRIZ to software only wants to analyze, and therefore "fix," the coding itself. However, a piece of coding is without exception but a single component, or several components, within a much larger and more complex engineering system. Further, the computing system the coding operates within most often fundamentally serves the purpose of being a control system within an again larger engineering system. Why is it then that so many have concentrated only on the coding when trying to improve their programming? Isn't that very much like an automotive engineer, who wants to improve the operation of the cam shaft, only analyzing the cam shaft (one of the devices that controls and orchestrates the engine) and ignoring all other system components while doing so? This appears to be a technique limited to marginal affect.

Because of the poor results of previous software innovation attempts a more fruitful method for software innovation has been developed. The technique is the best way to gain innovative insight when working to improve an engineering system that includes a computer based control sub-system. The methodology, also known as SI for Computing, combines modifications of several traditional TRIZ and systematic innovation techniques in a process that provides enhanced insight towards the goal of innovating computer based engineering system. The following materials will discuss SI for Computing's: Capabilities Maturity Model Integration (CMMI) process inspiration, utilization of systems engineering analysis as the basis for the method and finally additional features that allow for its effective usage in a variety of scenarios.

It should be noted that a glossary for "words of art" utilized within this paper is included just prior to the references.

2. Capabilities Maturity Model Integration (CMMI)

The Capabilities Maturity Model Integration (CMMI) process improvement methodology demonstrated that poor software is not a development issue but rather a system analysis and integration issue. "CMMI was developed by a group of experts from industry, government, and the Software Engineering Institute at Carnegie Mellon University. CMMI models provide guidance for developing or improving processes that meet the business goals of an organization. A CMMI model may also be used as a framework for appraising the process maturity of the organization."[2] CMMI was originally designed for the support of software engineering processes but has since been expanded for general product and service development, among other applications. According to the Software Engineering Institute (SEI, 2008), CMMI helps "integrate traditionally separate organizational functions, set process improvement goals and priorities, provide guidance for quality processes, and provide a point of reference for appraising current processes."[3] As can be seen in "Figure 1 - Characteristics of the CMMI Levels" organization's processes are rated on five levels of maturity. The higher maturity ranking an organization's processes receive the more effective and robust software those processes are capable of developing.

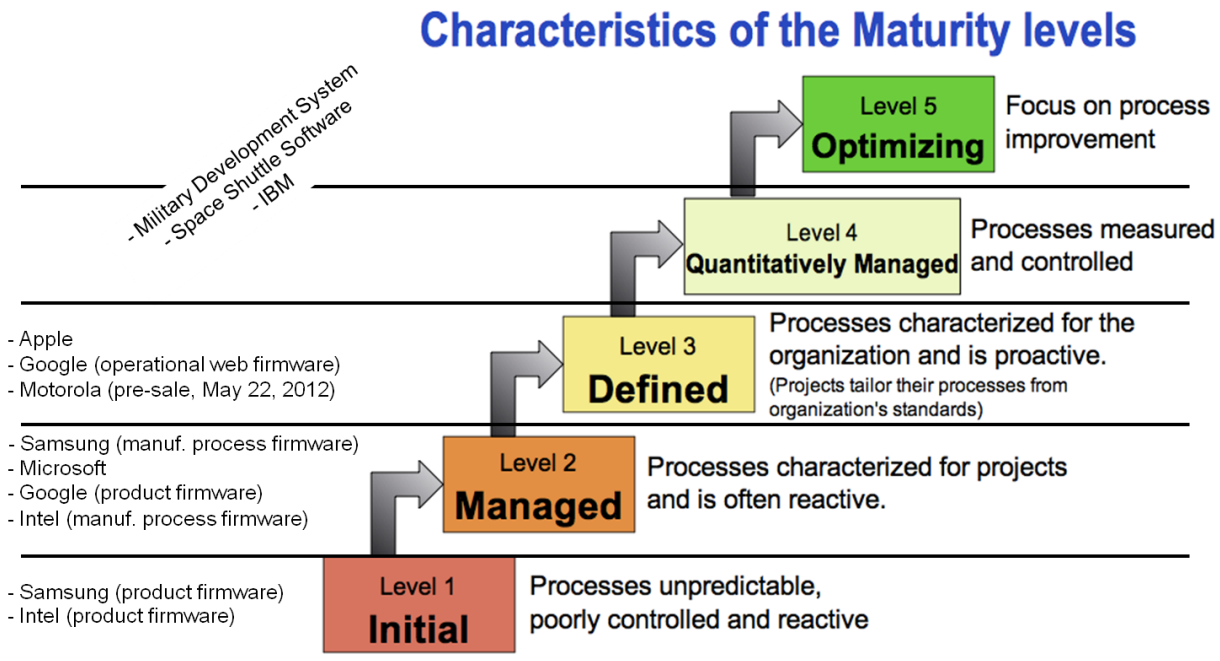

Figure 1 - Characteristics of the CMMI Levels [4]

Organizations whose processes are ranked at the lower levels (one and two) mistakenly believe that their software developers need more skills and that they will then write better and more effective coding. However, in fact there is probably little to no difference in the ability of the programmers within those CMMI level one and two organizations than those employed by the organizations in the upper echelon levels of the CMMI. So then why do the programmers working within organizations

with processes ranked at CMMI levels three, four and five consistently produce high quality software? The answer is surprisingly simple. Among other attributes, higher level CMMI organizations utilize elevated degrees of organizational coordination, customer input, feedback and understand, and control their projects from a full systems level approach (in so far as both the software development process and the engineering system the software is being developed for). It sounds obvious but the extent to which programmers interface with the end customer, and better understand the entire system they are coding for, the more robust and effective coding they write. However, organizations with processes ranked at CMMI levels one and two most often throw programming chores to their software developers treating their task as a necessary evil in finishing a larger project. Referring again to "Figure 1" it can be seen that both Intel and Samsung Electronics are believed to be ranked at CMMI levels one and two. When you step back and consider that both Intel and Samsung Electronics' primary business is hardware design and manufacturing it seems more likley that they would treat firmware for those products as a task rather than a product in and of itself. Contrary to that approach, organizations whose processes are ranked at the upper CMMI levels treat the software they development as a products and expect it to be orchestrated with their, or their customer's, full engineering systems. It is this last attribute of understanding the software coding requirements form a full systems engineering analysis standpoint that prompted my initial jump into developing the systematic innovation for computing process (SI for Computing).

3. Systems Engineering Analysis

Systems engineering analysis is a process that drives to insure that all pertinent aspects of a system are considered and integrated into a single study. There are many models that can be effectively used within a systems engineering approach but the method that best integrates with TRIZ, and other systematic innovation methodologies, is functional modeling. Functional modeling is a system analysis method which considers functional relationships between a system's components. A graphical representation of a functional model depicts the system's components and connects those components with arrows showing the functional relationships between those components. In other words, it depicts the affect the components have on each other. "Figure 2 - Generic Computer Based Engineering System Functional Model" portrays a simple functional model of a generalized computer based engineering system (*note - no insufficient, excessive or harmful functions are annotated outside of the air and heat components*). The beauty of a graphical functional model is that it focuses the analyst on what is import about an engineering system, namely the functional output of the system and the functions occurring between the system's components that create that functional output. It is through this function mapping that the requirements of an engineering system's computer control systems software can easily be established with little uncertainty.

The goal when functionally modeling a computer based engineering system should not be myopically focused on improving the software but rather on improving the system in general.

Sometimes the system improvement pursuit does point towards the need for a modification of the control software/firmware. However, in other cases the system issues can be addressed at other locations within the engineering system. Just as with an engineering analysis of an electro-mechanical/thermal system the problem solver should never begin an analysis with a preconceived notion of which system component, or interaction, needs to be improved.

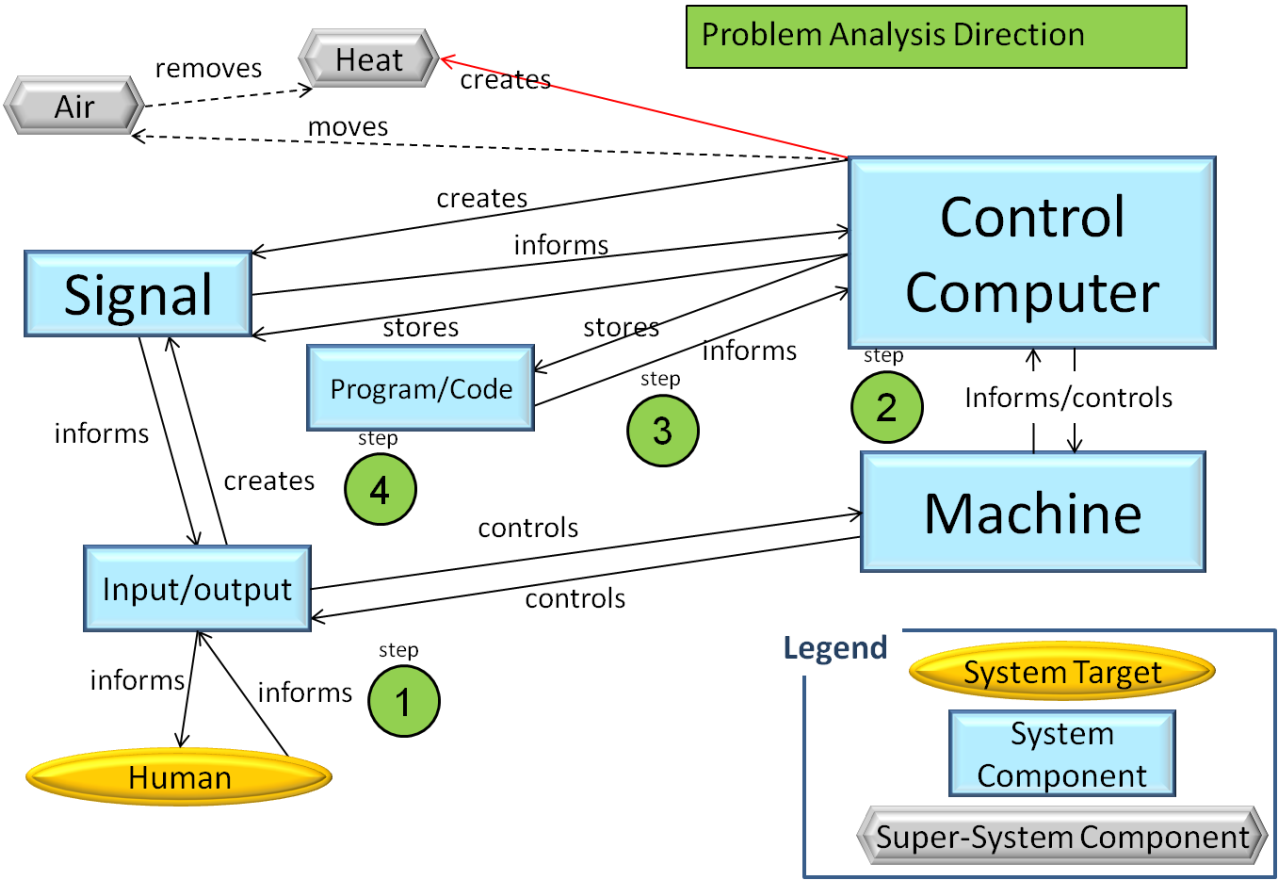

Figure 2 - Generic Computer Based Engineer System Functional Model

Pre-determining an improvement focus usually results in significantly limiting the improvement opportunities and therefore the overall results. In fact the first place an analyst should focus on is the basic function of the system. What is the basic function? The basic function of a system is a "useful function that acts directly upon the product of the functional model, also very likely to be the main function of the engineering system."[5] The reason it is best to start the analysis with this interaction is that the basic (or main) function is the most important function of the engineering system as this is where the systems functionality (or affect) is delivered. For example, a programmable thermostat has the job of measuring air temperature, deciding if the air temperature is within a pre-set range and then sending an on or off signal to the furnace. It is the function of signal *informs* furnace that is most important to the successful operation of the engineering system called programmable thermostat. If the basic function of signal *informs* furnace is judged to be insufficient then it is necessary to understand why. It should not automatically be assumed that the

programmable thermostat's firmware is the problem. What if the signal wire from the thermostat to the furnace is of a smaller gauge than necessary, resulting in a high electrical resistance, in-turn not allowing the signal to reach the furnace consistently? In this scenario a simple signal wire upgrade might very well fix the problem. However, assuming wire gauge was not the problem then it would be necessary to continue working the analysis back through the system eventually arriving at the thermostats firmware. If the issue is found prior to reaching the firmware then the problem could be addressed elsewhere. For instance, maybe the system controller is not effectively exchanging data with the system's memory. In this case a hardware fix may be in order. The point is that while software changes may indeed fix issues occurring at other points in the system the software/firmware should never be focused on solely as the target of the analysis. With this in mind any computer based engineering system analysis should always start from the basic function and work its way back towards the software while asking the question - where is the principal issue in the system and where is the best location within the system to address that issue? More specifically:

1.) What is the relationship between the machine and the super-system (shown as a human in "Figure 2") and where can it be improved?
2.) What is the relationship between the machine and the control computer and where can it be improved?
3.) What is the relationship between the control computer and the software/firmware and where can it be improved?
4.) What is the relationship within the software/firmware itself and can it be improved?

"Figure 3 - Computer Based Engineering System Analysis Flow" illustrates the above with an analysis direction and focus that could be utilized when analyzing a computer based engineering system and refers directly to the numbered steps in "Figure 2."

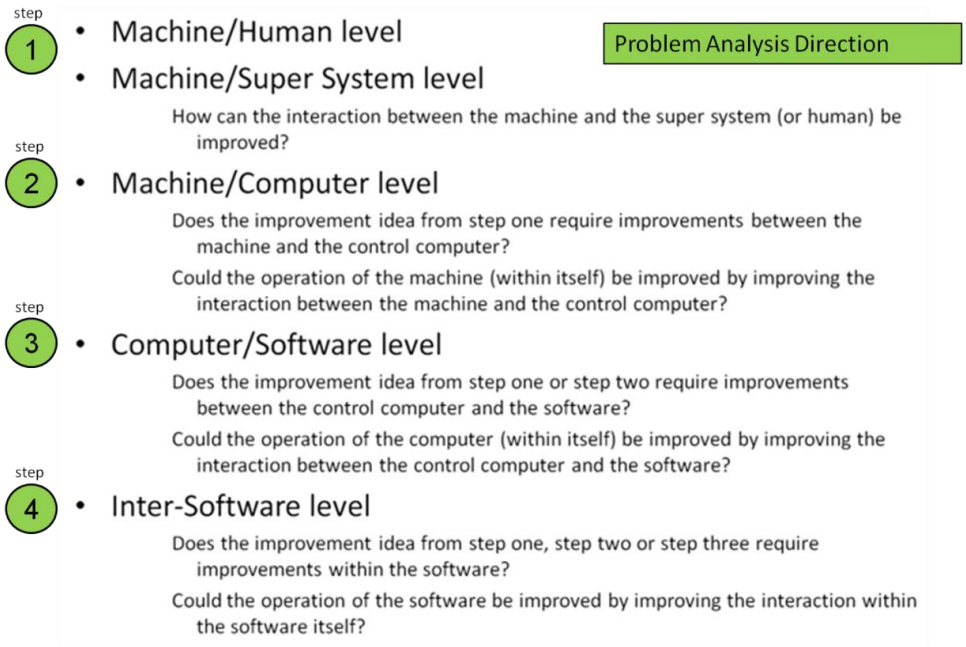

Figure 3 - Computer Based Engineering System Analysis Flow

Now that we have discussed the basics of functionally modeling a computer based engineering system there are a few more details that are important to the effective application of the functional model to this type of analysis. First is the delineation between the control system, the embedded/computing system and the coding. Second is the comprehension of the importance of identifying and understanding the interface and interactions between the physical world components and the control system in so far as establishing the software/firmware operational requirements. And finally, the modeling of the coding itself.

It is important to understand the boundaries, interactions and relationships between three hierarchically concentric, yet distinct, system levels within the computer based engineering system. Referring to "Figure 4 - Computer Based Engineering System Hierarchy" and starting from the inside and working our way out these sub-systems are: software/firmware, embedded/computing system and the control system. The software/firmware, embedded/computing system and the input/output devices constitute the control system. In relation to the management and operation of the greater engineering system, the most important sub-system (functionally) is the control system. The control system is made up of all the sub-systems that work together to create the function(s) that connect the virtual/digital computing components and sub-systems with the physical/analog "real world" sub-systems (e.g., display, hydraulic valve, relay, etc.). If the control system is designated as the engineering system then its output is of course a basic function and serves to ultimately control the super-system based target. For example, considering a motion activated security light, the control system would include the motion sensor (input device), computing/logic system and the relay (output device) that stops electrical current from reaching the light emitting device (see "Figure 4"). Moving deeper into the system the next functionally important sub-system is the embedded/computing system. This sub-system carries all of the components necessary to analyze input signals, make decisions and send output signals but it does not have the ability to connect to our outside "real" world. It can be seen that the embedded/computing system referred to in "Figure 4" is generically made up of a logic circuit, processor core, input/output device, analog/digital converter, signal, "logic" current, memory and code. The power supply may be considered as part of the embedded/computing system depending on whether or not that designation suits the analysis. Finally, moving even deeper into the system the least functionally important sub-system is the software/firmware. The software/firmware sub-system is actually a combination of the coding itself (a virtual component) and a memory module which serves as the physical location for the virtual software to reside and allows for the coding's interface and interaction with the other physical computing component. The inclusion of both the code and its residence (memory module) within the sub-system called software/firmware is important because the inclusion of the memory module allows for the modeling of the interactions of the coding with other components and the inclusion of the coding allows for the modeling of the system focused affects of the software/firmware. This view point produces sometimes small but often significant realizations about the functioning of the code within the embedded/computing system.

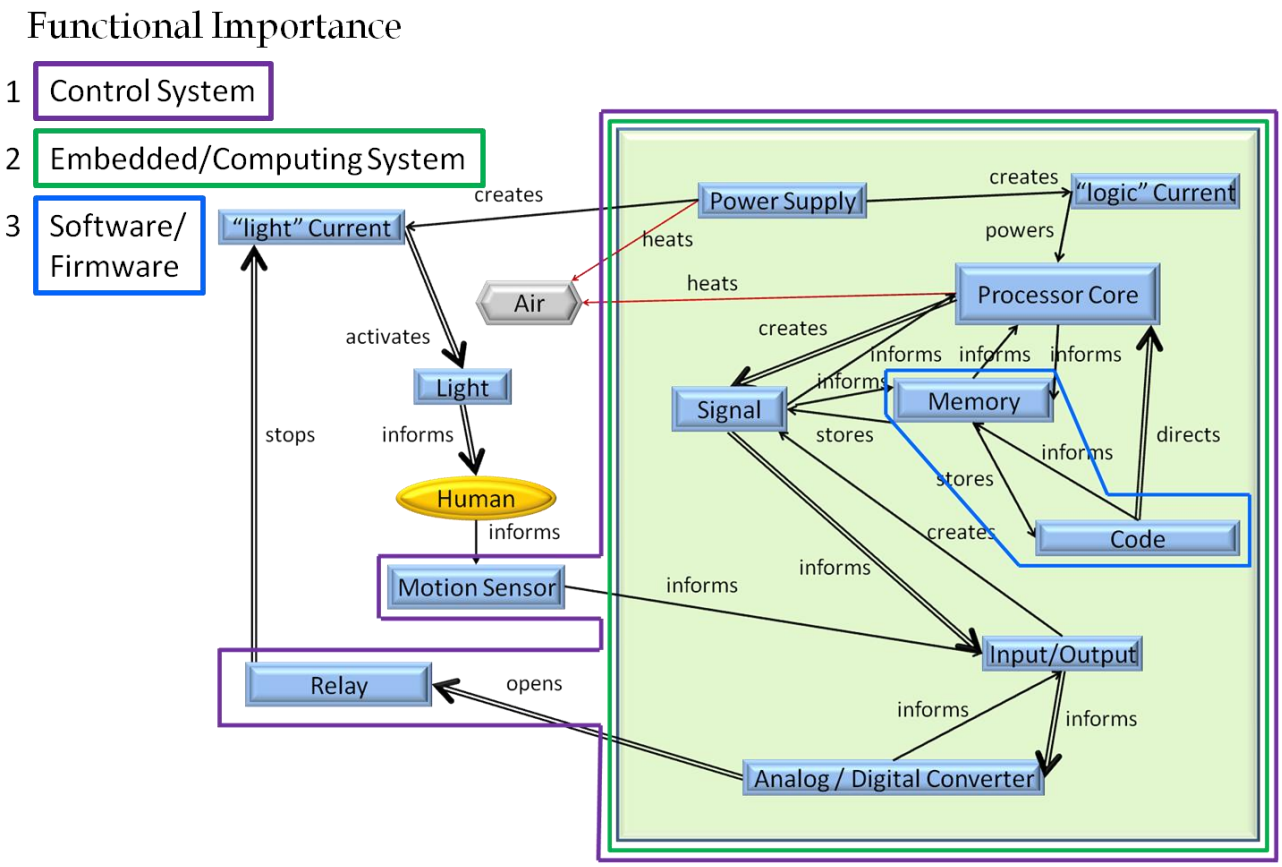

Figure 4 - Computer Based Engineering System Hierarchy

Now that we have defined the three concentrically arranged major sub-systems of the control system it is important to understand their inter-relationships. As we have learned the most important function of any engineering system is its basic function. In the example of the motion activated security light the basic function would be light *informs* human. Now stepping back to the outer most concentric control sub-system the most important function of that control system is the function that directly contributes to the delivery of the basic function. Again referring to the "Figure 4" the function created by the control system and affecting "outside" components is the function of relay *stops* "light" current. Therefore, the function of *stops* "light" current is the basic function of the control system and therefore its most important function of the control system. Furthermore, the auxiliary function of human *informs* sensor is also very important to the effective operation of the motion activated security light because it provides the input signal which triggers the control system. We now know that it is the input to, and output from, the physical/computing intermediary components (e.g., sensors, buttons, switches, relays, monitors, motors, etc.) that ultimately define the operational and performance requirements of the control system. In so far as effectively supporting the basic function of the entire engineering system it is these requirements,

and these requirements alone, that the system's software architecture, design and syntax should be designed around.

Assuming that a systems engineering functional analysis indicates that software/firmware changes are in order then it may be necessary to model and examine the existing coding. Due to the immense complexity and sheer quantity of coding existing in most computer based engineering systems this pursuit has proven extremely difficult to most. The difficulties are exacerbated by the practice of attempting to model coding at the command line and syntax level. While it is sometimes possible to gain some limited insight around a focused analysis between a few software commands, the results often provide limited impact and can produce unforeseen complications elsewhere in the coding or greater engineering system. Similar to an example given in the introduction, if we are to improve the functional output of an internal combustion engine we would probably not attempt to do so by modeling the system at the lattice structure level of the engine materials. Analogously, trying to model coding at the command line or syntax level is equally futile. So then how do we model coding? According to Kevin Brune of Google Corp., "software is best functionally modeled at the module level." This is true because it is at the modular level that discrete functional inputs and outputs can be delineated and their interactions with other modules, and of equal importance with other embedded/computing system components, can be analyzed. Once again referring to "Figure 4" it can be seen that the "coding" is represented as a single component within the embedded/computing system. Let us now look at an example of how coding can be functionally represented at the modular level by examining "Figure 5 - Code Module Functional Model." The graphic in "Figure 5" represents a generic anti-lock braking controller with appropriate code modules depicted for such a device. Notice that we now have the ability to see how the various modules interact with each other, and with other components, within the embedded/computing system. Further, we can see how the memory module, part of the "coding" sub-system, interacts with other embedded/computing system components. In other words, it is now possible to analyze the code modules at the same hierarchal level within which the computer/controller components exist. It is now fairly straight forward for a systematic innovation analyst to create technical or physical contradiction models around any problematic relationships identified in the modular level functional model. As modeling at the modular level provides a clear understanding as to which other system components each module interacts with it will then be apparent as to where coding modifications might best be made to support the desired system improvements addressing the individual contradictions. Therefore, this modular level analysis not only points the analyst towards specific sections of code for improvement but also allows for the understanding of how any code changes will affect the greater system. This allows for a much easier evaluation as to how those changes will ultimately affect the overall operational system. This is the desired output of such an analysis because it is system operational improvements we are interested in. The improvements might be driven by coding changes but those coding changes, and the analysis goal, should be focused on improvements to the system, not the coding itself. For the more advanced systematic innovation practitioners it is suggested to apply substance-field (su-field)

modeling to a coding module and embedded/computing system component analysis. This recommendation is made because su-fields allows the simultaneous modeling of the interrelationships between all components under analysis while the associated solution modeling Standard Inventive Solutions allow for more advanced solutions concepts to be applied.

4. Two Additional Systematic Innovation for Computing Methodologies

Finally, there is a bit more detail that additionally enhance the ability to analyze computer based engineering systems. Specifically, the further resolution of the action "informs" and a resources analysis framework that is very helpful for innovating within computing systems will be discussed. The advanced resolution of the action "informs" and the computing focused analogy of system resource analysis will improve the insight gained from functionally modeling a computer based engineering system and help with the subsequent solution generation.

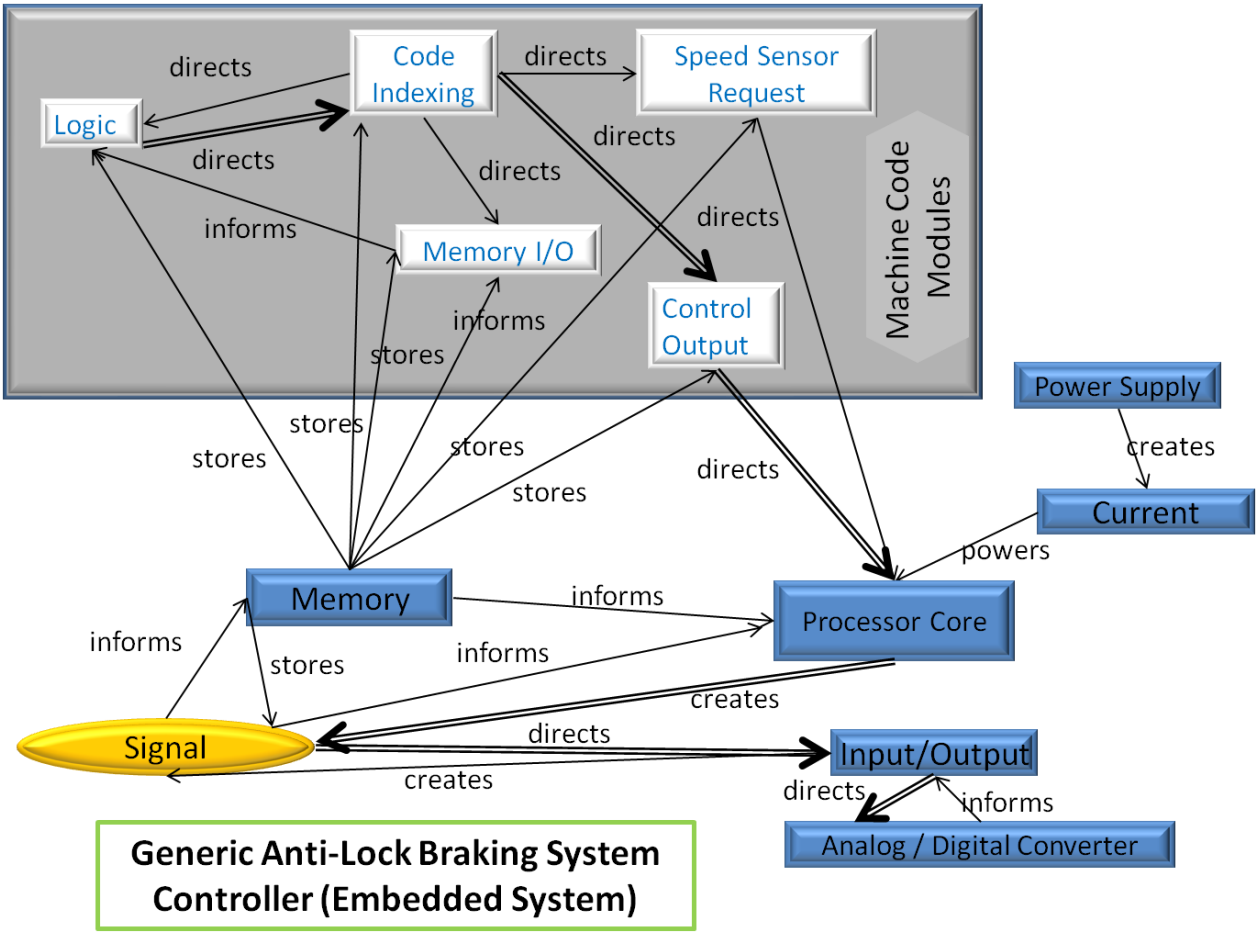

Figure 5 - Code Module Functional Model

The action "informs" is used in electro-mechanical/thermal TRIZ analysis to cover what is actually a fairly wide range of interactions and affects. For the majority of those electro-mechanical/thermal system analyses the action "informs" is sufficiently descriptive to allow

for an effective analysis. However, when analyzing computing based systems the action "informs" is much too broad in its meaning and does not sufficiently portray the relationship between the components of interest. Fundamentally, there is a big difference between the action "informs" when it is meant to represent that information is simply available and when it is meant to represent that the function results in a direct control of an effected component. In response the progressive sequence of "informs" functions/actions, which has the goal of providing better modeling tools resulting in improved analysis and insight, has been developed. The progressive sequence pertains to how much control the action "informs" represents. Referring to "Figure 6 - Hierarchy of the Action Informs,"[6] there are four levels of the action, each increasing the level of system control and therefore their importance to the associated engineering system's basic functionality.

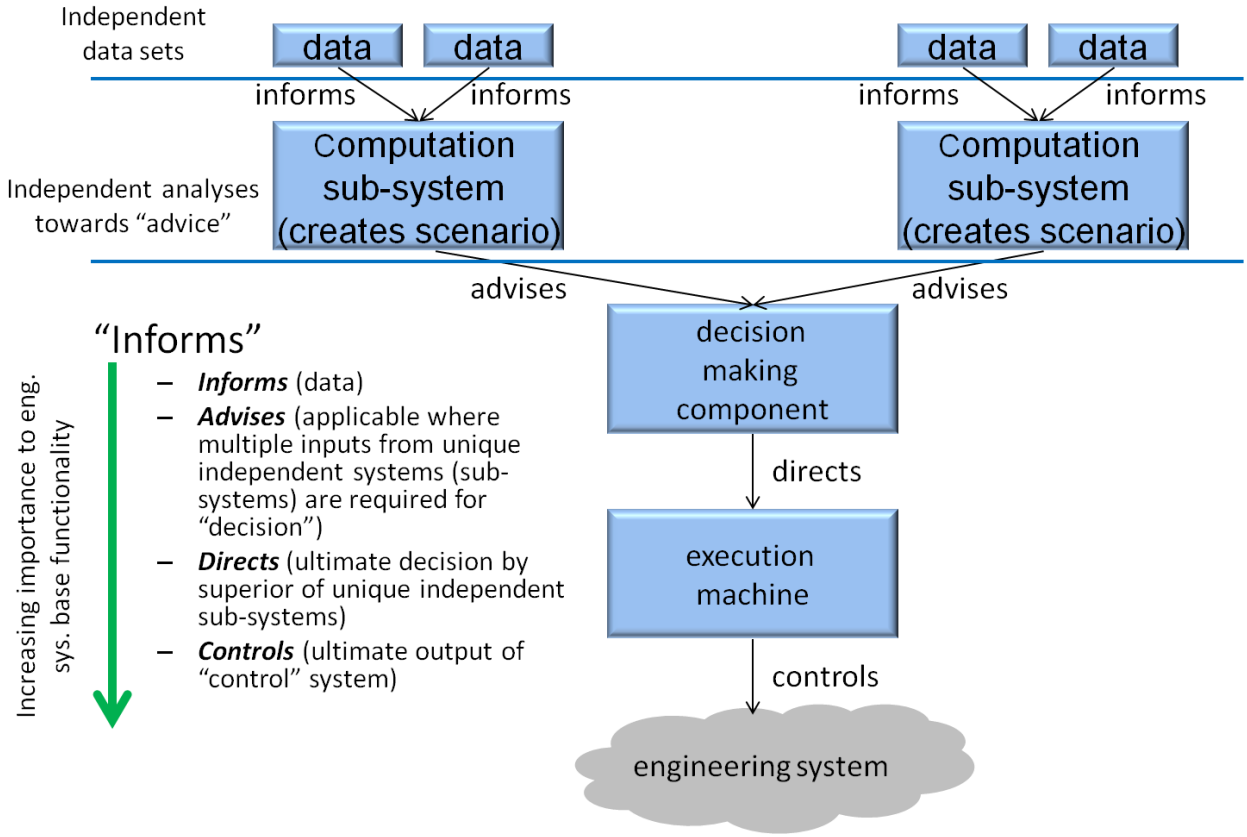

Figure 6 - Hierarchy of the Action Informs

Within this hierarchy, the first, and least important as to the delivery of the basic function, is that of "informs." The action refers to independent data sets and specifically providing those data sets to a component or sub-system that uses that data to create a intermediate step scenario for further analysis. An example of the action "informs" could be the provision of atmospheric environmental data. The next function along the path is that of "advises." "Advises" is the output of a scenario generating sub-system or component that takes multiple independent data sets and in turn combines them into a scenario that will be used in a decision making process by way of a subsequent component or sub-system. An example of the action "advises" could be the output of a scenario

generator that takes atmospheric environmental data and makes a tornado advisory recommendation to a subsequent system. Then follows the function of "directs." "Directs" is a function that specifically creates a triggering of a specific action with no additional logic required. This function affects the last component within the embedded/computing system and is therefore still in the data state. An example of the action "directs" is the output of a system which takes multiple "advice" inputs, say from an tornado advisory generator and from a radar analysis, and makes a decision to activate a severe weather alarm. Finally, the last function, and the most important to the basic function of the embedded/computing system, is that of "controls." As implied "controls" is the direct activation of a component, or sub-system, and specifically reaches across the boundary between the embedded/computing system and control system. Therefore the action "controls" is the activation of a devices that directly affects the electro-mechanical/thermal portions of the engineering systems. An example of the action "controls" would be a component that takes the input function of "directs" severe weather alarm and creates a signal that physically activates the alarm system.

Now let's discuss solution generation resource analysis in conjunction with computing systems. A popular method for analyzing resources for an electro-mechanical/thermal system is that represent by the mnemonic MATChEM. MATChEM stands for: mechanical, acoustic, thermal, chemical and electro-magnetic. In other words, when searching for ways to influence an electro-mechanical/thermal system the analyst should consider all of the resources associated with these categories. Therefore an analogous method for computing systems has been developed. Gregory Frenklach, provided some insight regarding the MATChEM resources. He described how each resource in MATChEM represents an interaction at a different level. A slight modification of Gregory's description is as follows:

Mechanical = object interaction
Acoustical = surface interaction
Thermal = lattice/matter interaction
Chemical = molecular interaction
Electromagnetic = electron/spin interaction.

Utilizing the above, an analogous resource list for computing was developed. Referring to "Figure 7 - Resource Analysis for Software and Hardware"[7] it can be seen that there are many ways to look at software and hardware resources and understand at what level their interactions occur. For example, the analogy to a thermal interaction (at the lattice/matter level) would be to consider the code dynamics (or rather the movement/action of the coding relative to itself) in relation to the system's software while clock, speed, or orchestration would be considered relative to the system's hardware. Additional study of "Figure 7" will provide more insight as to how the interactions between and within the software/virtual and hardware aspects of a computing system can be used to help solve challenges residing within that system.

5. Conclusion

The application of TRIZ and systematic innovation to computer based engineering systems can be greatly improved by following a few changes to the methods employed for electro-mechanical/thermal systems analysis. First it should be understood that the system's coding should not be the primary focus of an innovation exercise as there are often other ways to improve computer based engineering systems without any changes to their coding. As far as the analysis process goes the first step is the modification of the handling of the functional modeling of the computer based engineering system. Initially, the delineation between the control system, embedded/computing system and the software/firmware should be well understood and the basic function of, and inputs to, the embedded/computing system should be comprehended. Specifically, it is the basic function of, and inputs to, the embedded/computing system that establishes the operational and performance requirements of the embedded/computing system and therefore the software/firmware. Next, if coding changes are in order then any functional analysis of that coding should be performed at the module level. It is at the module level that coding functionality, and the associated interactions, can best be delineated and judged. Next, a higher resolution of the action "informs" should be employed to better represent the progression from "informs to "controls." Finally, the software and hardware resources should be studied in order to identify the variety of solution generation opportunities available based on interaction improvements.

SI for Engineering		**SI for Computing**		
Technical		**Software/Firmware**		**Hardware**
(MATChEM +)	***Abstraction****	***(SID-LC)***	***Abstraction***	
Mechanical	Object	**Software/Firmware Package - design**	Design	**Operational System**
Acoustic	Surface	**Interface/Handshaking (w/external systems)**	Protocol	**Connections/Interconnects/Transmission**
Thermal	Lattice/Matter	**Code Dynamics (movement/action relative to "itself")**	Execution	**Clock/Speed/Orchestration**
Chemical	Molecular	**Language/OS**	Instruction	**Component/Appliance (micro and macro)**
Electro-Magnetic	Electron/Spin	**Coding (routines/objects/agents)**	Objects	**Transistors/Devices**
		Bits/Words	Object Code	**Electrons/Holes**
+				
Nuclear	Atomic			
Biological	*various*	**n/a**	n/a	**Machine/Human Interface**

* The Techncial Abstraction column is a modifciation of work by Gregory Frenklach

Figure 7 - Resource Analysis for Software and Hardware

Glossary[8]

- *Auxiliary Functions* – Useful function that acts upon components of the system, usually from within the engineering system.
- *Basic Function (Base Functionality)* **–** Useful function that acts directly upon the product of the functional model, also very likely to be the main function of the engineering system.
- *Component (System Components)* – A material object (substance, field, or substance-field combination) that constitutes a part of the engineering system or super-system.
- *Engineering System* – A system that has been assigned to perform a function. A set of components that work together to perform a function.
- *Function* – An action performed by one material object (function carrier) to change or maintain a parameter of another material object (object of the function).
- *Function Model* - A model of a function being performed. Performance of the function is measured as sufficient, insufficient or excessive. Function analysis deals with the entity component that is a material object and can be either a substance or a field
- *Function Modeling* – A part of function analysis that builds a function model.
- *Physical Contradictions* – A situation in which two opposite requirements are placed upon a single physical parameter of an object.
- *Standard Inventive Solutions* - a set of 76 typical solutions, in the form of substance-field (su-field) models, to typical problems that are also expressed in the form of su-field models.
- *Su-Field (Substance-Field)* - A model of the problem related to the engineering system, but not of the engineering system itself. Also, the substance and field are clearly differentiated.
- *Su-Field Model* – Symbolic model of a problem or solution formulated in terms of interactions between substances and fields (virtual, real, or improved).
- *Super-system* – A system that includes the engineering system as a component. All systems outside the boundaries of the engineering system. Pertinent super-systems includes those that interact with the engineering system. Understanding the relationship between the engineering system and relevant super-systems is important in understanding system problems and identifying resources that can be used in the ultimate solution.
- *Target* – An object of the main function of the analyzed engineering system.
- *Technical Contradiction* - A situation, in which an attempt to improve one parameter of an engineering system leads to the worsening of another parameter.
- *TRIZ* - The Theory of Inventive Problem Solving,

Reference

[1] Drew, C., Hale, K., Hahn, W., Atwal, R., Graham, C., O'Connell, A., & Summer, B. (2014, March 27). *Forecast Alert: IT Spending, Worldwide, 1Q14 Update*. Retrieved from https://www.gartner.com/doc/2694417

[2] Sally Godfrey (2008) What is CMMI? NASA presentation. Accessed 8 Dec 2008. slides 23-24

[3] CMMI Overview. *Software Engineering Institute.* Accessed 16 February 2011. p. 15

[4] David Conley and Kevin Brune (2012) Software Development Organizational Maturity. *Innomation Corp and InnovaTRIZ presentation.* Slide 3

[5] David Conley (2012) Expert TRIZ Field Guide - Rev 2. *Innomation Corp training material.* Page 12

[6] David Conley (2013) SI for Computing Advanced Course - C4 Rev 6. *Innomation Corp training material.* Slides 99-100

[7] David Conley (2013) SI for Computing Advanced Course - C4 Rev 6. *Innomation Corp training material.* Slide 450

[8] David Conley (2012) Expert TRIZ Field Guide - Rev 2. *Innomation Corp training material.* Pages 90-97

Corresponding author(s):

Name: **Conley, David Worrell**

Position: **President / Managing Partner** **Affiliation:** **Innomation Corp / PQR Group**

Postal Address (line 1): **1416 Mesilla NE**

City, State, Zip (line 2): **Albuquerque, New Mexico 87110**

Country: USA

Phone: **+01-(505)-206-3401**

Email: **David@innomationcorp.com / David@TRIZPQRGroup.com**

Fax: **n/a**

Topical area (Select from listed topics on Call For Papers):

II) Technical Aspects of Systematic Product/Process/Service Innovation

A. TRIZ-based systematic innovation