Creating & Using Basic Hard Science: Componentology

Raju Chiluvuri

[1] Componentology.org & Neuronology.org
raju@Componentology.org

Abstract. Componentology is a new branch of basic science, which is the right theoretical foundation to successfully conduct applied research in CBSE (Component-Based Software Engineering), software engineering, design, structure, and architecture. The purpose of Componentology is to accumulate comprehensive hard scientific knowledge and insights by systematically studying all aspects of the reality of physical Components, as well as all kinds of useful parts (e.g., nature and essential properties of various kinds of parts/components); the anatomy, structure, design, and construction or architecture of physical CBPs (Component-Based Products); and the methods or mechanisms of real CBE (Component-Based Engineering) of physical products. We are forced to create hard science Componentology since we could not find any evidence that Componentology (or its approximate equivalent pure/basic science about the physical reality of ideal CBPs and ideal Components, which are the essential building blocks to build ideal CBPs) is available. It is not difficult to create Componentology, since there are many pure/hard sciences (e.g., botany, zoology, chemistry, particle physics, genetics, or microbiology) today that have successfully studied the reality of the complex real-world things (including invisible microbes) that are dozens of times more complex. It is impossible to successfully conduct applied research in CBSE by relying on flawed theoretical foundations, and the existing theoretical foundation (i.e., comprising theories, concepts, or descriptions) about so-called components and CBE for CBSE is flawed and pseudoscientific.

Keywords: Components; Component-Based Product (CBP), Component-Based Engineering (CBE); Componentology; Component-Based Software Engineering (CBSE); Body of Knowledge (BoK).

1 Introduction to a New Basic Science & Applied Research

It is possible to increase overall software productivity, quality, and agility (e.g., rate of innovation by reducing cost, complexity, and time to redesign and test) five to ten times by making necessary inventions to design and build large software products as ideal CBPs (Component-Based Products). The problem is that no one in the software world knows what is meant by a CBP. To build every large software product as an ideal CBP, it is essential to objectively gain pure/hard scientific knowledge such as (i) valid scientific understanding and insights into various aspects such as the structure, anatomy, and mechanisms of physical CBPs, and (ii) the nature and essential properties of

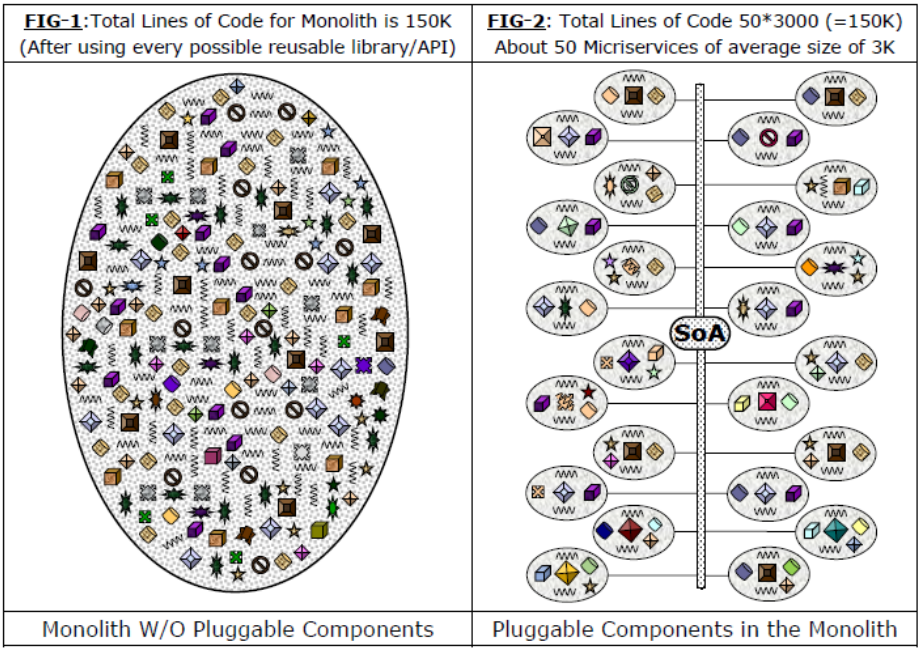
physical components, which are the ideal building blocks to build each large product as an ideal CBP.

Our research and development have two parts - (i) basic research to create new hard science Componentology (i.e., for using it as valid theoretical foundation) (ii) conducting applied or engineering research by relying on valid theoretical foundation.

1. Objectives of basic research: Create a new branch of hard science, Componentology, which is the essential theoretical foundation for successfully conducting applied research. It is impossible to achieve the objectives of applied research without creating & relying on this new valid pure/basic science Componentology (see the first three pages in Section-3 below that compares the two paradigms).
2. Objectives of applied research: To invent all the necessary technologies, tools, and methods necessary to build any large software product as an ideal CBP, which can increase overall productivity and quality by ten times (see my grand vision and objective for ideal CBPs in Section-2 below).

How is it possible to make the necessary inventions to build any large software application as an ideal CBP, if no one in the software world wants to know what is an ideal CBP (e.g., its structure and anatomy) and the essential properties, nature, and mechanisms of ideal pluggable Components, which are the essential building blocks to build each CBP?

Figure 1 & 2. Comparing structure/architecture of two competing paradigms.



2 Descriptions for ideal CBPs & ideal Components

Let me summarize my grand vision for the ideal CBSE (Component-Based Software Engineering): Building each large software product as an ideal CBP (Component-Based Product) as in FIG-2. Each ideal CBP must be built by "plugging in" multiple optimal-sized real software components, into an instance of reusable software Component/Class "software/virtual system board" (e.g., represented by "SoA" in FIG-2). Today, no one else in the software world knows what an ideal CBP is or has seen such real CBPs. No one else has created or used the mechanisms of SoA that are essential to build CBPs.

Let me define an ideal CBP (Component-Based Product): A product built by plugging in multiple optimal-sized software components as illustrated in FIG-2, where an optimal-sized (or optimal complexity) software component is a component that (i) is created by implementing 2500 to 4500 lines of custom application code, and (ii) can be redesigned and tested individually at any time of the lifecycle of the application.

To accomplish this grand vision that can increase productivity, quality & agility by ten folds, it is required to figure out and invent each of the missing pieces such as:

1. Gaining scientific knowledge about the essential nature and characteristics of physical components to break up the monolith in FIG-1 into multiple optimal-sized self-contained modules (shown in FIG-2),
2. Tools, technologies, and mechanisms to implement each self-contained module as a pluggable component (see FIG-2), such as implementing each component as a class-definition, that can be used to instantiate an object instance.
3. Invent the virtual system board (see the SoA in FIG-2) that implements necessary mechanisms which can automatically facilitate communications and collaboration between the components that are plugged in.
4. Invent necessary tools, methods, methodologies, and mechanisms to minimize (or preferably eliminate) the cost of assembling and disassembling (or replacing) any of the components.
5. Two vital conditions are: (i) the custom application code for each component must be of optimal size and complexity, and (ii) cost of plugging in each component must be under 0.1% (preferably zero) of the cost of implementing the components.

The largest portion of our research efforts has been invested in achieving this vital goal: The most desirable characteristics of ideal components for building ideal CBPs are (i) optimal size/complexity, and (ii) minimizing costs of assembly and removal/replacement for each component (e.g., preferably reducing the cost to negligible, or nearly zero). We already secured 8 US-patents (see Section-6) for all the above inventions, which can realize the grand vision to increase overall productivity by ten times.

Each component can be implemented as a Class-definition, assembled by instantiating an object instance of the Class-definition and by adding the object instance to SoA – it requires just 2 lines of code. Removing the 2 lines disassembles the component. It is forbidden to implement more than 2 lines of code to plug in each component in FIG-2. Except for the 2 lines of code to plug in each component (by instantiating its object instance and adding it to the SoA), it is prohibited to implement any more custom code

(e.g., for initialization or in the communication code for the components) in the code of the application that uses a reusable SoA. It is prohibited to implement any custom code outside of the class-definitions for each of the pluggable components.

In the case of physical CBE for electronic products such as computers, smart phones, and network-switches, almost every function or feature is implemented in one of the components as in FIG-2. The purpose of the PCB (Printed-Circuit Board) is to facilitate communication or collaboration between the components (that are plugged into the PCB or system board). The reusable software component "SoA" in FIG-2 is equivalent to the PCB but far more intelligent than the PCB (which is a dumb plastic plate with a network of wires custom made for each product-model, so not reusable).

Assume that it is required to implement 150K lines of custom code for implementing a software application as illustrated in FIG-1. Today, the application code is implemented as a monolith as illustrated in FIG-1 (i.e., as spaghetti code, which is also derogatively called as "Big Ball of Mud"). In the case of the proposed real CBSE, less than 0.25% of custom code is implemented outside of the pluggable components.

Over 99.7% of this application code must be implemented in one of the pluggable components, where each component can be redesigned and tested individually to keep up with changing needs. The code for the application comprises of nothing more than just a reusable software board (shown as SoA), and no more than 2 lines of code to plug in each of the pluggable components into SoA as in FIG-2 (e.g., by including an Object instance). Each pluggable component can be redesigned and tested individually, outside the application, at any time in the future to keep up with changing needs.

This grand vision and goal for real/ideal CBSE is achieved by conducting basic/pure scientific research that accumulated valid hard scientific knowledge for Componentology.org and conducting applied/engineering research by relying on the valid hard scientific knowledge for Componentology.org.

Brief introduction to Service-Oriented Architecture (SoA) in FIG-2: SoA is a well-known and widely used approach for inter-process communication between multiple executables, which requires no elaborate explanation. In the case of SoA in FIG-2, I merely adapted this concept to facilitate intra-process communication within a single executable, specifically for the interactions and to facilitate communications between object instances for pluggable components. A top-level view is provided at: http://real-software-components.com/CBD/Sample_SoA_CASE_tool.html and comprehensive implementation is provided at: https://patents.justia.com/patent/11275567.

It is indisputable that great industrial engineering inventions, such as Eli Whitney's interchangeable components, the stationary assembly line patented by Olds, and Ford's moving assembly line, vastly increased productivity and quality by reducing the costs of assembling and replacing components. The purpose of the SoA is to minimize the cost of replacing any of the components in FIG-2. The objective is to provide outstanding service access to each of the components: http://componentology.org/raju/Service-Access.pdf. Any of the components can be replaced by substituting old components with new ones (or a new version of the components) in the code (see listing-1), and a new version of the application can be created by recompiling the application code. Reseach or innovations for hot swapping of components is beyond the scope of this paper.

In the context of Component-Based Products (CBPs), each product evolves by independently evolving its components. As nearly 90% of software engineering involves modifying existing code, it is most desirable to minimize the cost, complexity, effort, and time required for code redesign. For a non-component-based monolith in FIG-1, even minor changes necessitate compiling and testing the entire application. However, in the case of an ideal CBP in FIG-2 where the core (shown in SoA) contains no custom code, modifications are made within one of the pluggable components. This process requires only the compilation and testing of the specific component.

2.1 A Sample City_GIS application that has just 3 sub-components.

Assume the necessity of creating a geographical information system for a city, such as City_GIS, which can function as an application or a container pluggable component comprising three sub-components. The task entails the presentation of three pluggable components, or PCs, namely: (i) A city road map with real-time traffic conditions, (ii) Notable landmarks as interactive components that show real-time data, and (iii) Real-time air traffic information that show flights on the city map.

You may enlist the services of three developers, denoted as A, B, and C, and ask each to develop each of the three Pluggable Component Classes (PCCs). The process of building each PCC is not significantly different from creating and testing a miniature independent application. For instance, implement a complete custom application code for each PC within the mini application and encapsulate the code in a class definition. Each PC can be tested and refined individually, just as we test a miniature GUI application. Once all the components (or mini-GUI applications) are prepared, they can be assembled as demonstrated in Listing 1.

Listing 1: Code to Assemble 3 Pluggable Components

```
1.    Void CGM(Out) {  // CGM of City_GIS_PCC assembles the components
2.
3.       // Include pluggable component created by developer "A"
4.        RepComp CityMap = new CityMap_PCC (ACi, ZipCode);
5.        this.canvas.AddChild (CityMap, 0, 0, null);
6.
7.         // Include replaceable component created by developer "B"
8.        RepComp LandMarks = new CityLandmarks_PCC (ACi, ZipCode);
9.        this.canvas.AddChild (LandMarks, 0, 0, null);
10.
11.       // Include replaceable component created by developer "C"
12.       RepComp AirTraffic= new CityATC_PCC (ACi, AirportCode);
13.      this.canvas.AddChild (AirTraffic, 0, 0, null);
14.
15.      this.canvas.CGM(Out); // Display the GIS for City
16.   }
```

As demonstrated in Listing 1, it only requires two to three lines of code to plug in each of the PCs, by incorporating an object instance for each PCC. The term "ACi" refers to the ‘Application Context Instance”, which comprises a reference to the "SoA"

object depicted in FIG-2. The SoA facilitates communication and collaboration among the PCs, eliminating the need for manual implementation of communication code to enable collaboration between them. If City_GIS is implemented as a PCC, it can serve as a sub-component for another PCC, and so forth. Even when the application comprises 100 PCs, it still necessitates no more than 2 to 3 lines of code to plug in each of them. Each PC in FIG-2 can be redesigned and tested individually at any point during the software product's evolutionary life. Hence, each component can be removed or replaced simply by deleting or modifying those 2 or 3 lines.

3 Evidence to expose the existing flawed dominant paradigm.

Definition of "BoK for CBSE": "***BoK for CBSE***" stands for "BoK (Body of Knowledge) that is used as (and/or necessary) theoretical foundation for conducting applied reseach in CBSE (Component-Based Software Engineering)".

Componentology is a new field of basic and hard science created to accumulate comprehensive hard/pure scientific BoK for CBSE by striving to maintain highest scientific rigor, objectivity, and integrity to scientifically study all aspects of reality such as the anatomy, structure, design, and construction of physical CBPs (Component-Based Products); the essential properties, mechanisms, and nature of physical components that are essential building blocks to build each product as a CBP; as well as the methods and mechanisms of real CBE (Component-Based Engineering).

3.1 Exhibits B1, B2 & B3 – The existing "BoK for CBSE".

It is not difficult to prove that the existing theoretical foundation to conduct applied research in CBSE is fundamentally flawed (i.e., pseudoscience). Exhibits B1, B2 & B3 comprise of a few samples/specimens of knowledge (e.g., descriptions or beliefs) in the existing theoretical foundation (i.e., BoK for CBSE).

Exhibit-B1: https://wiki.c2.com/?ComponentDefinition

Exhibit-B2: http://real-software-components.com/raju/ExhibitB2_BookForCBE.pdf

Exhibit-B3: http://real-software-components.com/raju/WhatIsComponent2.pdf

The first few paragraphs of the first chapter of almost every textbook that introduces a discipline of science or engineering teaches the foundational truths/axioms (also known as the first principles) of the discipline. Hence, the best place to learn about the foundational assumptions (i.e., core first principles) for the existing dominant paradigm for CBSE is to read the first chapter of the most popular books on software components. Exhibit B2 is one of the most popular books on software components, which says "Components are for Composition and are Reusable". But the reality is that most of the components for ideal CBPs are custom designed to fit perfectly and perform optimally for just one product model, and so are not reusable, not composed, but pluggable.

3.2 Comparing existing BoK for CBSE & new BoK of the Componentology

Exhibits A1, A2 & A3 in sections 3.3.1, 3.3.2 & 3.3.3 (see below) comprising objective descriptions and observations of **Componentology** illustrate the valid scientific

theoretical foundation we used for conducting our applied research in real CBSE. The following table illustrates that it is difficult to find a proven principle (e.g., empiricism, objectivity, testability, and falsifiability to name a few) of the scientific method which is not violated by the existing BoK for CBSE (i.e., Exhibits B1, B2 & B3). Hence, we chose to create hard scientific knowledge (instead of using existing voodoo scientific knowledge) to be relied upon for conducting our applied research in real CBSE.

Table 1. Comparing BoK of two competing paradigms.

The existing BoK represented by Exhibits B1, B2, & B3	**The BoK we have accumulated for hard science of Componentology**
The existing knowledge (in the Exhibits), such as many descriptions (which conflict with each other) for Components, CBE, and methods for CBE, have no objective empirical basis in the real world and are contrary to evidence such as valid and verifiable observations about physical CBPs and Components for CBPs.	Our knowledge (see Exhibits A1, A2 & A3 in section 3.3.1/2/3 below) such as scientific descriptions for CBPs, the Components necessary for CBPs, and the necessary methods and mechanisms for CBE, is based on empirical evidence and is consistent with hundreds of valid and verifiable observations made about their physical or real-world counterparts.
There are multiple descriptions for components in Exhibits B1 & B3 that are ambiguous and subjective, and they are inconsistent with each other. It is impossible to have more than one valid scientific description for physical things.	We have only one objective comprehensive description for CBPs, and only one scientific description for Components for CBPs. Each description is consistent with all known observations & empirical evidence.
Today, it is forbidden to test or falsify the descriptions for so-called components in Exhibits B1 & B2. It is considered heresy and disrespectful to question the validity of these descriptions (e.g., to falsify) by producing valid verifiable counter evidence.	The objective hard scientific knowledge we have accumulated (such as scientific descriptions, methods, and theories for CBPs, CBE, and Components) can be tested to be refined/improved and can be falsified by finding any new evidence that is not consistent with the knowledge.
Today, it is forbidden to present counter evidence or observations about reality that is not consistent with deeply entrenched beliefs, 55-year-old ***flawed first principles,*** or misconceptions about components and CBE.	Like other hard sciences (such as botany, zoology, & chemistry) that study physical things, it is forbidden to violate any scientific principle or ignore any evidence in Componentology when studying physical components & CBPs.

The column on the left represents the BoK for the existing deeply entrenched dominant paradigm for CBSE, while the column on the right represents hard scientific BoK vital for making necessary inventions to build each software product as a real CBP.

Our patented inventions (which relied upon hard scientific knowledge as theoretical foundation) can transform Software Engineering, which has been employing the inefficient non-CBE paradigm, into an engineering paradigm that is ten times as efficient in terms of productivity, quality, and agility for rapid rate of innovation (since each component in FIG-2 can be redesigned and tested at one-tenth the cost and time). These inventions prove that the voodoo scientific BoK detailed in the left column of the above table is the root cause for the infamous software crisis.

Summary: Any discipline that justifies blatant violations of basic rules of the scientific method (and is deceptively promoted as a science) is a voodoo science. Any major discipline that justifies using flawed beliefs or misconceptions as first principles is a voodoo science. The existing flawed BoK represented by Exhibits B1 & B3 have many subjective and ambiguous descriptions for components, where the descriptions contradict each other and are also inconsistent with evidence. Only 'God' has more mysterious, subjective, and ambiguous myths/descriptions, as if no one has seen CBPs and Components, which are the essential building blocks to build CBPs. It is a mistake to use such cryptic beliefs/descriptions as the theoretical foundation to conduct applied research in CBSE, so we created hard science Componentology and used Componentology (see Exhibits A1 & A2 below) as the theoretical foundation to conduct our applied research for CBSE.

3.3 Exhibits A1, A2 & A3

3.3.1 Exhibit-A1: Brief summaries of first principles (or foundational truths).

The foundational truths/assumptions (i.e. that which must be used as first principles in the foundation of software engineering) defined below are indisputably correct and verifiable scientific observations in the context of all other real engineering disciplines (e.g. mechanical, automobile, aerospace, computer or electronic disciplines), where each can be used to design and build each large or complex product (e.g. car, airplane, jet fighter, computer, spacecraft, electronic machine such as telecom-switch, medical equipment such as an MRI machine, and other large machines or machinery for factories) as a CBP.

1. Foundational Truth (also known as a First Principle): Any product can be a real-CBP (Component-Based Product) if and only if the product is built by assembling real-components, as illustrated in FIG-2 (each oval-shape represents a pluggable component).
2. Auxiliary Basic Truth: Any part can be a real-component for building real-CBPs if and only if the part can be assembled (i.e. multiple such real-components are assembled or plugged in to build a real-CBP as in FIG-2).
3. Auxiliary Basic Truth: Any engineering discipline or paradigm can be real-CBE (Component-Based Engineering) if and only if the discipline can design and build every large product as a CBP (i.e. by assembling multiple real-components, as illustrated in FIG-2).
4. Subsidiary Basic Truth: Components (e.g. devices such as CPU, DRAM, hard drive, keyboard, software, or other electronic products) that communicate with

each other by exchanging data and signals can be plugged in (e.g. into a printed-circuit-board or system-board shown by SoA in FIG-2).

The first steps and starting point for any new scientific inquiry or investigation (e.g. for any new hard scientific discipline such as botany or zoology, when the hard scientific discipline was in its infancy) is to make objective observations with an open mind and without any preconceived notions & prejudice. Computer Science defined first principles (e.g. to acquire knowledge that is essential to comprehend reality of software Components, CBPs, and CBE for software) 50 years ago by violating this elementary principle, since the foundational assumptions are contrary to evidence and observations that can be made about their real-world physical counterparts.

3.3.2 Exhibit-A2: Basic Research Questions about Real CBPs & Components

Question-1: What is the striking difference between specific kinds of parts that are certainly components to build CBPs and all other kinds of parts that are certainly not components, in the context of the mechanical, aerospace, and electronic engineering paradigms for designing and building large products (e.g., spaceships or airplanes)?

Answer-1: A part can be a component if and only if the part can be "loosely coupled," where the term "loose coupling" can be defined using the phrase "loosely coupling a part." This phrase refers to (a) "assembling a part" (e.g., an engine, gearbox, or wheels) in the context of engineering paradigms for mechanical or aerospace engineering disciplines, and (b) "plugging in a part" (e.g., a CPU, DRAM, or hard drive) in the context of engineering paradigms for electronic or computer engineering disciplines.

Question-2: What is the striking difference between (1) a product that is certainly a component-based product (CBP), which is designed and built by employing component-based engineering (CBE) paradigm, and (2) a product that is certainly not a CBP, which is designed and built by employing non-CBE paradigm?

Answer-2: A product can be a CBP if and only if the product is built by assembling multiple components. For example, any product (e.g., an automobile, an airplane, machinery for a factory, or a computer) that can be disassembled into components (and be re-built by assembling the components) is certainly built by employing the CBD/CBE paradigm. However, any product (e.g., a house, skyscraper, software application, or bridge) that cannot be disassembled and re-assembled is certainly not built by employing the CBD/CBE paradigm. Hence, paradigms for engineering disciplines such as mechanical, computer, electronic, and aerospace engineering are certainly CBD/CBE paradigms, while paradigms for engineering disciplines such as civil, chemical, and software engineering are certainly not CBE paradigms.

Question-3: What is the greatest engineering invention or innovation that has increased quality, agility, and manual productivity (over 100 times) more than any other artifact or invention in the history of industrial engineering?

Answer-3: A very specific kind of part that can be assembled and disassembled, which is an essential building block to build large products as CBPs. Such parts are widely known as "Components" (in the context of all other disciplines of engineering, such as mechanical, aerospace, electronics, and computer). In other words, a part can be a Component if & only if the part can be assembled or plugged in (i.e., loosely coupled).

The 100-year-old invention of Ford's moving assembly line (which resulted in a seven-fold increase in manual productivity) was built on the strong and essential foundation provided by the 200-year-old invention of interchangeable components; interchangeable components (which resulted in a hundredfold increase in manual productivity) were created on the strong and essential foundation provided by real Components (i.e., a very specific kind of part that can be loosely coupled, so that each component can be easily plugged in/out or assembled/disassembled). Comparable or analogous "Component-based" innovations are not possible in software without inventing real software components, which are the essential building blocks to build CBPs.

3.3.3 Exhibit-A3: Simple RQs (Research Questions) for Scientific inquiry

1. Name a product that is certainly a CBP?

Answer-1: Cars, Computers, and Airplane

2. Why is each of these products a real-CBP?

Answer-2: Each car, computer, or airplane is built by assembling multiple components. No product can be a CBP if it is not built by assembling multiple components.

3. Name a physical product that is certainly not a CBP?

Answer-3: Houses, Buildings, Bridges, or software products

4. Why is each of these not a CBP (Component-Based Product)?

Answer-4: Each house or building is not built by assembling multiple components.

5. What are the striking differences between real-CBPs and non-CBPs?

Answer-5: Any product can be a CBP if and only if the product is built by assembling multiple components. For example, any product (e.g. cars, airplane, or computer) that can be disassembled into components (and re-built by assembling the components) is certainly a CBP. However, any product (e.g. a house, skyscraper, software product, or bridge) that can't be disassembled and re-assembled is certainly not a CBP. Hence, engineering disciplines such as Mechanical, Computer, Electronics or Aerospace are certainly building products as CBPs, while engineering disciplines such as Civil or Software are certainly not building products as CBPs.

Research Questions to comprehend differences between real-CBE & non-CBE

6. Name an engineering discipline that is certainly employing real-CBE to design & build products as CBPs?

Answer-6: Mechanical, Aerospace, Electronics, or Computer engineering discipline.

7. How can you say each is employing real-CBE?

Answer-7: Each discipline is building products as real-CBPs. No discipline can be a real-CBE if it is not building large products as CBPs.

8. Please name an engineering discipline that is certainly not employing real-CBE (i.e. which discipline is not designing & building products as CBPs)?

Answer-8: Houses, Buildings or Bridges are examples for non-CBPs, where non-CBP imply a product that is not built by assembling multiple components. So Civil engineering is not employing real-CBE.

9. Why is it not a CBE (Component-Based Engineering)?

Answer-9: Each house or building is not built by assembling multiple components. That is, Civil or Software engineering has been building products as non-CBPs.

10. What are the striking differences between the real-CBE and the non-CBE?

Answer-10: Any engineering discipline can be a CBE if and only if the discipline can build large products as real-CBPs (i.e. by assembling multiple components). For example, disciplines such as Mechanical, Aerospace, Electronics, or Computer are examples of real-CBE, since each builds every large product (e.g. cars, airplane, or computer) as a real-CBP. However, not every product (e.g. house, skyscraper, software product, or bridge) is a CBP. Hence, engineering disciplines such as Civil or Software are not employing CBE, since they are not building products as CBPs.

Many software researchers and engineers claim to be leading experts of Software Components, CBPs, and CBE for software, so it is expected that they at least have an elementary knowledge and understanding of existing theories and concepts about Components, CBPs, methods for CBE, and proof for the theories and concepts. Anyone who considers themselves to be a scientist or researcher must not have any problem answering the elementary questions below about physical Components and physical CBPs if they are honest and have integrity.

The Exhibits in A (i.e., A1, A2 & A3) & Exhibits in B (i.e., B1, B2 & B3) illustrate that "Componentology" is the Heliocentric model of software engineering to replace the existing Geocentric paradox of software engineering.

4 Anatomy & Structure of Scientific Knowledge

Basic research and applied research are two vital parts of the research ecosystem to address any technological problem, where pure or basic research creates a theoretical foundation comprising scientific knowledge such as well-tested theories, concepts, methods, and observations or evidence, which are vital to gaining deep insights to understanding the reality or real-world things or phenomena. It is impossible to conduct applied research without having a valid theoretical foundation (e.g., Exhibits in A) or bad theoretical foundation (e.g., Exhibits in B), just as it is impossible to begin constructing a house in thin air without creating a (good or bad) foundation.

The following table shows the two vital layers or parts of the research ecosystem, which are (i) basic research and (ii) applied research. The purpose of basic research is to accumulate pure scientific knowledge (i.e., theoretical foundation), which comprises scientific theories, concepts, methods, and descriptions. Pure scientific knowledge in the basic science of any scientific discipline can be divided into two categories, which are (i) first principles and (ii) non-first principles (derived theories or concepts), where each of the non-first principles (i.e., knowledge such as a theory, concept, method, or description) is acquired by relying on one or more first principles and additional empirical evidence (e.g., valid experiments or observations).

Pure scientific knowledge in the basic science of any scientific discipline can be divided into two categories, which are (i) first principles, and (ii) non-first principles,

where each of the non-first principles (i.e., knowledge such as a theory, concept, method, axiomatic belief, or description) is acquired by relying on one or more first principles, and additional empirical evidence & observations. Therefore, basic science ends up in a crisis and as a voodoo science if core first principles are fundamentally flawed. And it's a fool's errand to conduct applied research by relying on voodoo science: http://componentology.org/VoodooScience.pdf

Table 2. It is vital to have valid first principles & theoretical foundation.

Applied Research Layer #2: Applied Research for CBSE is conducted (by relying on the available good or bad scientific foundation - layer #1 below - comprising theories, concepts, axioms, or beliefs in basic science) to invent tools, methods, & technologies to create real software components; and to use components to build software products as ideal CBPs by inventing more tools, methods, and mechanisms of CBE.
Theoretical Foundation Layer #1: Basic Science comprises of two parts:
I. First Principles: Foundational Axioms, Discoveries, and Theories
II. Derived BoK comprises of advanced theories, concepts, methods & descriptions

It is indisputable that it is impossible to reach a valid scientific conclusion (either in the basic science layer or the applied research layer) by relying on flawed beliefs, assumptions, myths, or illusions. If core first principles are flawed, it is impossible to create valid basic science, and if the basic science is flawed it is impossible to successfully conduct applied research by relying on the flawed BoK in basic science.

It is impossible to 'begin' constructing a house in thin air, without having a (good or bad) foundation for it. Similarly, it is impossible to even 'begin' applied research (in layer #2) for CBSE (Component-based Software Engineering) in thin air without having a few good or bad concepts, axioms, and beliefs about parts and components in the theoretical foundation. Applied research can never be successful if the existing theoretical foundation (i.e., Exhibits B1, B2 & B3) is fundamentally flawed.

A good science implies that it comprises of valid scientific knowledge including well-tested or validated theories, methods, concepts, & descriptions, which are acquired without violating proven principles of the scientific method. A bad science implies that it comprises of flawed scientific theories, beliefs, unproven concepts, definitions, or descriptions, which are acquired by violating proven scientific principles. Examples of bad science include (a) the 16th-century geocentric illusion and (b) the existing theoretical foundation for CBSE comprised of flawed scientific beliefs such as concepts or descriptions of so-called components and CBE.

Applied research cannot be successful if it is forced to rely on bad or flawed basic science. There is no exception to these two rules for applied research in CBSE: (1) it is impossible to even begin conducting applied research without having a (good or bad) theoretical foundation (Exhibits in B is the theoretical foundation for existing CBSE research), & (2) applied research can never be successful if its theoretical foundation is bad or flawed. Hence, we have created Componentology to replace the existing bad theoretical foundation with hard scientific BoK.

Any scientist is a voodoo scientist if he cannot understand that it is vital to have a valid science (i.e., a good foundation in layer #1) to conduct applied research. The existing theoretical foundation for CBSE research has been accumulated over more than 50 years and comprises of countless theories, definitions/descriptions, concepts, and methods for so-called software components and CBE for software. This knowledge in basic science layer #1 shapes our mental and conceptual perception of reality of the physical or real world.

Since it is impossible to begin applied research for CBSE without having any good or bad ideas, concepts, or beliefs (i.e., first principles), computer science about 50 years ago created ideas/concepts based on bad axiomatic beliefs as core first principles, such as bad descriptions for components. Hence, I am forced to create Componentology (i.e., a good conceptual model), since existing knowledge such as theories, concepts, & descriptions are illusory and fictional without having a valid testable basis in reason, logic, or reality, which are included in basic science and all of them shape our understanding and perception of the reality of components (or various kinds of useful parts) when conducting applied research for CBSE.

It is impossible to begin applied research for CBSE having zero concepts, axioms, or descriptions, such as about components, the role they play, how they are used, and the anatomy & structure of component-based products. There must be something in layer #1. Layer #1 cannot be empty to begin any work in applied research layer #2. Since researchers have been conducting applied research for CBSE in layer #2 for many decades, layer #1 today has many concepts, theories, axioms, and descriptions for components and methods of CBE. It is not difficult to prove that most of this knowledge that exists today in layer #1 is flawed.

If it is possible to create and use hard science as its theoretical foundation for any discipline of engineering (or applied science), it is mandatory to create and use hard science as the theoretical foundation of the discipline. More than 50 years ago, it was certainly possible to create hard-science Componentology, but I am inclined to agree that it was not possible to contemplate the use of Componentology as the theoretical foundation to conduct research in CBSE. So, using pseudo-science (see Exhibits in B) 50-years ago was a discretionary and excusable mistake, when primitive programming languages (e.g., Fortran) could not create large objects (i.e., pluggable components). Today, it is mandatory to create and use hard science Componentology.

The research ecosystem has two vital parts/steps. The first step entails establishing a robust theoretical foundation rooted in valid first principles, as depicted in Layer 1 of Table 2 on page 10. Subsequently, the second step involves conducting applied research, situated in Layer 2 of Table 2, while relying on the pertinent theoretical knowledge. Attempting to address a technological problem without first acquiring the necessary scientific knowledge is akin to putting the cart before the horse – a strategy that inevitably proves ineffective.

When required to address a technological problem (see section 2), it is imperative to initially identify all the relevant theoretical knowledge essential for its resolution and, with an open mind, engage in basic research to fill any knowledge gaps. Successfully addressing a technological problem becomes an insurmountable challenge if the

foundational theoretical knowledge in Layer 1 is marred by flawed theories or descriptions (as in Exhibits B). Therefore, it is crucial to eliminate flawed theories and descriptions from the underlying theoretical knowledge in Layer 1. The imperfections present in Layer 1 are often amplified in the inventions created in Layer 2. There is a very strong interdependency between these two vital parts of the research ecosystem.

5 Three Cardinal Rules/Principles of the Scientific Method

Anything that violates the law is unlawful. It is unlawful to suppress relevant evidence. Likewise, anything that violates proven principles of the scientific method is unscientific voodoo science. It is scientific misconduct and deception to suppress valid evidence. Componentology has been created through scientific rigor by putting forth our best efforts to strictly adhere to the principles of the scientific method. The primary reference is basic and proven principles of the scientific method.

"By denying scientific principles, one may maintain any paradox." - Galileo

It is valuable to understand the mistakes committed when accumulating the existing flawed BoK represented by Exhibits in B, which violate the following three cardinal rules of the scientific method. It is deplorable scientific misconduct to deliberately violate any of the following three cardinal rules and justify those violations:

1. It is scientific misconduct and dishonest to justify acquiring scientific knowledge (i.e., for theoretical foundation) by blatantly violating proven principles (e.g., empiricism, objectivity, testability, and falsifiability) of the scientific method. Any theory or description that is promoted as scientific is voodoo science if it is not objective, and is precluded from being testable or falsifiable. Today, it is disallowed to test or falsify the descriptions in Exhibits B1 & B2.
2. It is scientific misconduct and dishonest to ignore or suppress valid and verifiable evidence, and it is scientific misconduct (if not fraud) to cherry-pick data/evidence. Today, it is disallowed to use valid observations and evidence (e.g., acquired by RQs of Exhibits in A3) to falsify the descriptions in Exhibits B1 or B3.
3. It is scientific misconduct and dishonest to justify using flawed beliefs or myths as first principles to create a large and most widely practiced scientific or engineering discipline such as software engineering having tens of millions of active practitioners. Review the web-pages listed in section-2 about vital role and function of first principles: http://componentology.org/Lawsuit/VitalBasicKnowledge.pdf

Another perspective to establish the mistakes: Please read synopsis of the most cited and influential academic book of all time, "The Structure of Scientific Revolutions" by Thomas Kuhn, who is one of the 20th century's greatest philosophers and historians of science. The first paragraph of this synopsis at https://www.uky.edu/~eushe2/Pajares/kuhnsyn.html says: A scientific community cannot practice its trade without some set of received beliefs. These beliefs form the foundation of the "educational initiation that prepares and licenses the student for professional practice". The nature of the *"rigorous and rigid" preparation* helps ensure that the received beliefs are *firmly fixed in the student's mind*.

[Mistake-1] Scientists *take great pains to defend* the assumption that scientists know what the world is like...

[Mistake-2] To this end, "normal science" will often *suppress novelties which undermine its foundations*.

[Mistake-3] Research is therefore not about discovering the unknown, but rather *"a strenuous and devoted attempt to force nature into the conceptual boxes supplied by professional education"*.

Computer science researchers have been committing all three mistakes predicted by the most cited academic book of all time, which was published in the year 1962 and not just survived but ushered in a paradigm shift in our understanding of scientific progress and nature of scientific knowledge. It is most deplorable to justify such mistakes in the 21st century. The goal of Componentology is gaining unbiased objective BoK and insights, as if a dozen scientists were asked to create Componentology 60 years ago before the advent of digital programable computers. The requirement is to study the objective reality scientifically by strictly adhering to the principles of scientific method. Assume that three groups are formed to conduct independent basic research, with each group comprising two scientists and two engineers. Each group conducts its research independently, meticulously accumulating observations, data, and evidence.

Finally, all the groups collaborate to reconcile any differences based on observations and evidence and collectively write a book on Componentology. If such an approach had been in place, the researchers at the 1968 and 1969 NATO Software Engineering Conferences would never have made flawed assumptions about CBE and Components for CBE. These assumptions led research down the wrong path and ultimately resulted in the crisis as illustrated here: http://real-software-components.com/raju/ModifiedKuhnBlackHolePhase.pdf.

Researchers must keep in mind that the BoK for the existing dominant paradigm has evolved over the past 54 years by relying on flawed assumptions made during the 1968-1969 NATO Software Engineering Conferences and has been polluted with misconceptions about CBE and Components for CBE (See Exhibits B1, B2 & B3). Those assumptions were made when it was inconceivable to create Objects and pluggable Components. To address the crisis, software researchers must know and address the three mistakes predicted by Dr. Thomas Kuhn in his book. Replacing a large and most widely practiced flawed dominant paradigm with a new valid paradigm is the most complex of endeavors, since reality would be perceived as strange and heretical: http://componentology.org/Misc/ReplacingFlawedParadigm.pdf

6 Miscellaneous Information & Background

Our applied research is to increase each of the three industrial engineering factors below and we have secured multiple US patents, which include a GUI-API capable of building ideal software components that can be plugged in to build complex component hierarchies for GUI-CBPs (Patent Nos. 7,827,527; 7,840,937; & 8,527,943), patents relating to real software components to build ideal Component-Based Products (Patent

Nos. 8,392,877; 8,578,329; & 9,058,177), and patents for a CASE-tool (e.g., SoA in FIG-2) to automatically assemble pluggable components (10,949,171 & 11,275,567).

It is well established in the discipline of industrial engineering that it is possible to increase productivity by increasing the degrees of (i) division-of-labor, (ii) specialization, and (iii) automation. Almost every engineering activity has a certain degree of each of these factors, and productivity can be increased by increasing the degree of one or more of these factors. Inventions such as interchangeable components and Ford's moving assembly line increased productivity by substantially increasing these factors.

Dr. Alan Key, the father of OOP (Object-Oriented Programming) and winner of the AM Turing award (considered the Noble Prize in computing), compared Software Engineering with the building of ancient pyramids 4500 years ago (please read first page only at): http://tinlizzie.org/~takashi/IsSoftwareEngineeringAnOxymoron.pdf.

Dr. Fred Brooks (considered the father of Software Engineering) is another AM Turing award winner who persuasively argued in his seminal research publications that there is "No Silver Bullet" to kill the software crisis or the problem of spaghetti code: http://worrydream.com/refs/Brooks-NoSilverBullet.pdf. He is also the author of the influential book on Software Engineering: "Mythical Man-Month".

This paper: http://www.es.mdh.se/pdf_publications/4272.pdf, claims to have conducted the most comprehensive survey on 28 years of CBSE, by analyzing 1231 published research papers on CBSE dating from 1984 to 2012. I could not find any research paper that is not consistent with the BoK in Exhibits B1, B2 & B3. The foundational assumptions (or first principles) for the existing dominant paradigm were admittedly made in 1968 & 1969 NATO Software Engineering Conferences (i.e., McIlroy (1968)).

I could not find any research paper that is consistent with the first principles in Exhibit A1. Exhibits in B and Exhibits in A contradict each other, no differently to how the BoK for heliocentric model and the BoK for geocentric model contradict each other. The existing BoK for CBSE (e.g., Exhibits in B) cannot be hard science, since it is based on axioms having no basis in physical reality, and is inconsistent with observations. Software history is littered with failed attempts to solve the software crisis, which all failed because they did not recognize that the flawed first principles and knowledge accumulated about CBE/CBPs (by relying on the flawed axioms) are pseudo-scientific.

Let me provide just three examples for failed endeavors: https://www.technologyreview.com/2003/11/01/233636/everyones-a-programmer/. ATP/NIST.gov wasted $150 million on "Component-Based Software" during the 1990s. Also, DoD has been spending $540 million every year in merely alleviating the pain of the dreadful disease that we can cure by using hard science Componentology: https://www.cmu.edu/news/stories/archives/2020/july/sei-contract-extended.html

No one asked for proof of the assumption that 'the Earth is at the center' made 2300 years ago (based on primitive knowledge), and no truth in history has faced more hostile resistance than the Heliocentric reality. Similarly, no one asked proof for the assumptions made in the 1968 NASA conferences (based on then available primitive knowledge), but the reality of Componentology is facing significant resistance. Any mistake must be fixed as root: http://componentology.org/Misc/FixingAtTheRoot.pdf

7 Summary & Conclusion

The goal is making necessary inventions to build every large software product as an *ideal CBP*, which requires making necessary inventions to create and assemble *ideal pluggable components*. It is impossible to make these inventions without creating and using hard science *Componentology* as theoretical foundation in Layer #1 of Table 2.

Which one represents the truth & reality of real CBPs (Component-Based Products):

1. In FIG-1, each product is built by using reusable ingredient parts such as cement, steel, metals, alloys, plastic, and paint, where the reusable ingredient parts (shown using various shapes) are falsely referred to as components in software engineering today.
2. In FIG-2, the same product is built by plugging in multiple components (i.e., object instances), where each component is built by using respective reusable parts (i.e., software parts that are akin to ingredient parts). Reusable ingredient parts (that are falsely referred to today as components for CBSE) are shown using various shapes.

In both cases, it is possible to use the same kind and number of reusable ingredient parts (represented by various shapes), but in the case of FIG-2, there is an intermediate step of creating pluggable components. That is, the monolith in FIG-1 is broken into multiple optimal-sized microservices (based on the scientific understanding of the nature and essential properties of ideal physical components), and each microservice is implemented as a pluggable component, which can be plugged in as in FIG-2. The most desirable trait for ideal components is (not necessarily reuse) but to minimize or eliminate the cost of assembling & cost of replacing/removing each component in FIG-2.

There must be a root cause for any discipline of hard science or engineering to end up in an inexplicable crisis, whose BoK is filled with beliefs/descriptions (e.g., Exhibits in B) that are subjective, untestable, unfalsifiable, and contradictory (see the left side of Table 1). The best way to address the crisis is to (i) find the root cause that created the crisis and (ii) address the root cause. It is impossible to address the crisis without finding and addressing the root cause. The root cause for the software crisis is the 50-year-old flawed, foundational, preparadigmatic assumptions (or first principles) at the root (i.e., in layer 1 of table 2): http://componentology.org/Misc/FixingAtTheRoot.pdf

We employed first principle thinking (popularized by Elon Musk) and critical thinking, and both discourage using references or analogies from the existing flawed paradigm. The existing BoK (in Exhibits B) is littered with anomalies and is clearly contrary to evidence and observations about their real-world or physical counterparts. In the context of research in science and engineering, no mistake is more deplorable than justifying violations of proven principles of the scientific method and suppressing evidence. See the three scientific principles in Section 5.

The two primary components to measure the size of any dominant paradigm are (i) the number of active practitioners, and (ii) the size of its BoK (Body of Knowledge). The existing software engineering paradigm is larger (in terms of number of practitioners and the size of its BoK) than the 16th century geocentric paradigm. The resistance to subvert a large flawed dominant paradigm would be proportional to the size of the paradigm: http://componentology.org/Misc/ReplacingFlawedParadigm.pdf

References

1. A comprehensive study of 1231 papers on CBSE published for 28 years until 2014: http://www.es.mdh.se/pdf_publications/4272.pdf. Introduction section suggests that the McIlroy (1968) provided foundational assumptions for the CBSE research.
2. Dr. Alan Key, compared Software Engineering with the building of ancient pyramids 4500 years ago (please read first page only at): http://tinlizzie.org/~takashi/IsSoftwareEngineeringAnOxymoron.pdf
3. Dr. Allen Key comparing software engineering with http://tinlizzie.org/~takashi/IsSoftwareEngineeringAnOxymoron.pdf
4. Dr. Fred Brooks paper "No Silver Bullet" to address the software crisis or the problem of spaghetti code: http://worrydream.com/refs/Brooks-NoSilverBullet.pdf.
5. The synopsis: http://componentology.org/raju/SynopsisKuhnBook.pdf of Thomas Kuhn book lists three mistakes of researchers in case of flawed dominant paradigm, which are: http://real-software-components.com/raju/Kuhn3Mistakes.pdf
6. The best place to fix a mistake is at the root. If core foundational assumptions (or first principles) of a discipline are flawed, it ends up in a crisis and the only way to address the crisis is fixing the first principles: http://componentology.org/Misc/FixingAtTheRoot.pdf
7. Replacing any large and most widely practiced flawed dominant paradigm with a valid paradigm faces hostile resistance and is a very complex endeavor: http://componentology.org/Misc/ReplacingFlawedParadigm.pdf