didi

Results of Applying the Cognitive Engineering Design Methodology

Version 05c, July25, 2018 • Brad Paley • brad@didi.co

The Problem: Multiple Applications for Each Single Task

Today's desktop application environment forces a trader to work with many disparate interfaces just to do a single business task. This is true for most knowledge-work roles (e.g. sales, portfolio managers, analysts), and the same problems and solutions play out in each of them, so we'll use trading as our example in this overview.

The current trading desktop accreted over the last 40 years as thousands of separate financial capabilities were brought on line. Data vendors wrapped feeds with applications in order to sell them, and engineers responsible for delivering analytical processes or action capabilities (e.g., liquidity venues) focused narrowly on each offering. A perfectly natural development process, but it left the effort of integrating all these applications to the trader.

Traders had no choice but to conduct business in this segmented environment, so they developed hopscotch attention sequences and clever window configurations. In order to do a task quickly, without breaking mental flow with cumbersome window swapping, applications were tiled side by side. As more applications became necessary there was no choice but to add more screens, and today's six- or eight-monitor trading workstation evolved.

One indication of how this limits trading effectiveness is the battle for screen real estate: *innovations are kept away from traders*—there's no room. And huge but still limiting screen real estate is a minor issue compared to complexity: forcing a trader to switch among multiple applications for almost every task imposes a considerable unnecessary burden on trading. Specifically: it slows every trader's thought processes and adds several kinds of risk. This is the root of significant inefficiencies in global markets, and virtually no targeted effort has been applied to solving it.

Why has this problem been overlooked? Partially because its causes are subtle: only recognized if we study overall trading behavior, not just single applications. And partially because it's unique to complex knowledge-work disciplines like trading, so it hasn't been addressed by software giants like Google, Amazon and eBay who recognize that the front end limits business—this problem doesn't limit a Web search or shopping cart. It's ours alone to solve.

Even when financial firms do address the front end, it's universally addressed as a post process: get the functionality running and *then* hire a designer to make it pretty—or let the engineers do it ("we're in a hurry…"). When the issue does come up people assume UX and Visualization address it—but once the functionality is locked down those disciplines can only provide branding or presentation, or a fashion-driven refresh—*one segmented application at a time*. As we'll see some of the problem is in individual applications, but mostly it's an emergent property at a higher level of organization: the complex ecosystem of desktop applications, at its worst where traders constantly have to thread their business processes through scores of screens. And it can only be solved at that desktop ecosystem level.

Until recently, there was no methodology that applied the right basic sciences to the ecosystem-level problem of business workflow in a rigorous, systematic way. It's virgin territory for optimization. It follows that effort dedicated here can generate returns far greater than effort applied where people have been optimizing for decades, like networking infrastructure, and may generate returns as high as those coming from areas where thousands of smart people have been looking for breakthroughs for years, like machine learning. The sciences lacking are the sciences of how the *mind* works, particularly Cognitive Science. And since improvements in cognitive workflow are completely orthogonal to more hardware/math-driven innovation they only add to or multiply those results. But business can only leverage innovations that make it safely into the *minds* of businesspeople. It's time to address those minds.

Accurately stating a problem is the first step toward solving it. Our core problem is the *segmentation of thought processes* that make up a trading workflow. It's not the design of individual interfaces, but the sheer number of applications and the scattered way they evolved: idiosyncratically, to fit different niches, using different tactics.

A New Approach: Cognitive Engineering

The Cognitive Engineering Design Methodology ("CEDM") targets the *domain expert's mind* as the scarce resource that needs to be optimized. CEDM captures all the data and action needs for one business task, then defines one "tool" that provides for all the task's needs. It visually, intellectually, and behaviorally structures each tool by applying basic scientific findings to the thought processes used for that task. There's no other reason to have a screen.

Some of the most unique results of Cognitive Engineering come from the *refactoring of the desktop*. A properly factored tool ecosystem helps people and systems collaborate to accomplishing business goals by simplifying switching among tasks, optimizing individual tasks, adopting new tools in a comfortable and orderly way, and ultimately shaping the back-end infrastructure more closely around better articulated business needs. We'll describe what desktop refactoring is, then address each source of value in turn.

Refactoring the Desktop…

Factoring, in engineering terms, is the act of identifying and consolidating common elements in a complex system. A trader's desktop is complex partially because the process of trading is complex—but partially because it has never been the subject of an explicit, rigorous factoring. Considered as a whole, as a system intended to support the trading process, it has been accreting over the last four decades: kludged together piece by piece by well-meaning and often talented individual actors, but each paying attention *only to their own offering*.

When factoring it's important to have an explicit goal. For example, factoring an ecosystem of trading applications around the ease/budget of the developer or the financial intent of the vendor will give different results than factoring it around the needs of the trading firm or the individual trader. Here, we address those latter goals, looking for trader effectiveness, client engagement, and reduced risk. CEDM reveals optimization opportunities in task switching, optimization of individual tasks, and a smooth transition to a better organized future.

…Eliminates Most Attention Switching

The current desktop accreted over history; it's accidentally factored around data feeds and action opportunities. It evolved driven by them, so what we see reflects them: a separate application for each data stream or action.

If we want to intelligently factor around (and optimize for) trading processes, we need to look not at the current applications, or even the desktop, but at the process that creates value in the firm: trader workflows and the entities that comprise them: the individual *tasks,* and the *steps* needed to accomplish each task.

One simple reorganizing principle changes the desktop to fit the way people think: we must *completely, effortlessly support each task* by designing a tool just for that task. By doing so we *eliminate* within-task switching, and reduce the negative effects of between-task switching. In detail, when the desktop environment has properly factored tools:

Fewer task switches are necessary. Completing a single task using the current trading desktop requires many switches between applications: more switches than there are applications since a trader sometimes has to revisit the same application more than once. Application switches have been counted in the dozens for a single trading task. A tool specified by CEDM requires exactly zero. (This switching cost is illustrated—for most trading tasks *underestimated*—by the red highlight area in Appendix A.)

Less information is carried in human memory. Because switches now occur at *task boundaries*, and tasks are defined as being (relatively) self-contained, most of the information carried in one's head during one task can be discarded when moving to the next. For instance, if a trader needs to consider multiple things like client needs, portfolio makeup, market conditions for a stock/sector, news, client credit, and the firm's proprietary goals when placing a trade, they may be juggling dozens of decision inputs while in the process of doing the trade. But that all collapses into a very simple thought chunk (e.g. "100,000 shares of IBM done"—or possibly just "done") when moving on to the next task.

The remaining switching is easier and less risky. Factoring around tasks enables another kind of improvement: since it defines new tools, we can apply a new design strategy within each tool—and across all tools.

When tools share a look and feel, it's quicker, more comfortable, and there's less *context-switching risk*: the trader doesn't have to swap one application's look and feel out of their minds to activate the (often very different) visuals/encodings/rules/behaviors of the next.

There's also lower *flow-interruption risk* because switching is quicker and requires less working memory. (These two improvements are also possible with best-of-breed traditional UX "refreshes.")

Sharing a design strategy also means switching imposes lower *misinterpretation risk*: data used in similar ways is visually encoded in similar ways: an apple always looks like an apple, a rate looks like a rate, a position looks like a position—and they each look different from the others.

This clarification-through-encoding isn't possible in traditional UX redesign efforts because the task-based operational usage of each element ("exactly what is this being used for," captured in a standard CEDM task breakdown) isn't captured in typical functional requirements. Pathologically, most UX practitioners prefer to make all numbers or all fields look the same, in service of a superficial design consistency. This can be attractive, but doesn't leverage human capabilities well—in fact it works against us: *we assign meaning to visual differences*. They help us navigate in the world, and likewise on the screen—but only when they properly encode usage differences as visual differences.

…Organizes Workflow for Mobile Computing Platforms

Since every decision-making parameter and possibility for action has been precisely identified by the CEDM task decomposition, there's no need to switch away from a tool to complete a task. If the task is sufficiently small *all* of its affordance can be shown on a single page: the size of a tablet—or even a smartphone.

Tools contain everything that's necessary for a task (1). The same CEDM characteristic that eliminates most switching also allows a workflow-driven recasting of tools for mobile computing platforms. Since every decision-making parameter and possibility for action has been precisely identified by the CEDM task decomposition, there is no need to switch away from a tool to complete a task. If the task is sufficiently small all of its affordance can be shown on a single page—the size of a tablet or even a smartphone.

Within-task switching retains intellectual context. In some cases the full tool is too large to be comfortably read or manipulated after it's been shrunk to fit the device. Nonetheless, it's necessary to show the entire tool: recognizing it will activate the task-related concepts in the expert's mind. And seeing the whole tool gives an indication of task complexity, and how much is completed—especially where task elements can be completed "out of order." But chopping it up, e.g. into pages, *imposes all the risks associated with switching*.

CEDM has many visual techniques that make affordances available without losing context. Intellectual context is strongly driven by perceptual context—e.g., if you forget why you went from the kitchen to the garage, going back to the kitchen is often enough to remind you. Context-retention techniques include zooming the entire tool, or just a subset of affordances; allowing them to overlap their context, or applying a fisheye-like transformation so it's still visible; and *semantic level-of-detail zooming*: where tool-defining or task-critical affordances shrink more slowly that others do. (Selecting which of these techniques is correct for the task—or inventing another—is also part of the rigorous CEDM process.)

…Optimizes for Each Tasks Within Individual Tools

When we tailor a tool to support just one task we make a far better tool. A Swiss Army Knife often has a screwdriver, but in a workshop that has real screwdrivers we'd never reach for the Swiss Army Knife. Current applications are generally engineered to support as many functions as possible due to limited engineering budgets, a backlog of user requests, adherence to traditional UI/UX design practices, and a lack of understanding of the advantages to a more task-specific approach. Designing a task-specific tool has five key advantages:

Tools contain everything that's necessary, and only what's necessary (2). Since tools are narrowly scoped around one specific task, and that task is deconstructed at a step-by-step, field-by-field granularity, tools are concise and exact: we know every item needed so we don't leave things out—and we don't add things "just in case." This guarantees the task can be accomplished, simplifies tool use, and eliminates distracting elements. The tighter definition also makes tools more economical to develop and easier to test. In fact, with Paper

Prototyping *we can test tool for completeness/correctness in schematics even before we write specs*. So when they're deployed, they just work; no need to recall and recode them because something's missing.

Tools transcribe the work process. Tasks have a natural flow in the mind. CEDM extracts that flow—even when experts can't verbalize it themselves—and decomposes it into atomic, sub-second steps. A step details the information needed or action desired, but also the *importance* of the step, *the format that best fits the mental need at this point in the process* (e.g., a three-decimal-point number, or just an up/down signal), *and the context that makes it understandable* (e.g. a stock's volatility in the context of related stocks, or a price next to a chart or volume distribution for the day).

Affordances in tools are shaped to be instantly recognizable. Each affordance that shows information or allows an action has visual attributes that distinguish it from other affordances (e.g., a rate is rendered to be recognized as a rate, not a price or position). And when the attributes correctly transcribe the way a trader thinks in that step, its local usage is recognized and the overall tool is distinct since it's composed of distinct affordances. This makes CEDM-specified tools natural to use, and also excellent for training: each taught idea has a distinct and natural look. This means navigating dozens of tools is as natural as working at a real-world woodworker's workbench; not an analogy: CEDM designs engaging the same cognitive mechanisms.

People interact directly with the data. Affordance that show information also allows action on that information. There is no need to go to another screen and navigate to a place to act—no need to move one's eyes at all.

Some tools can be shared among related roles. Different business roles sometimes share tasks, e.g. a trader and a portfolio manager might want to look at—or even share—a single view of orders that need to be executed. Rather than rebuilding that order view screen separately in the EMS and OMS, or even porting it from one application to the other, exactly the same tool—the same code—can be deployed for each role.

Tools are light-weight development efforts. Since a tool has an extremely limited scope, and since it shares affordances with other tools, a new tool is easy to develop. And an old tool is easy to modify.

…Changes Incentives to Cause an Orderly and Demand-Driven Migration

New designs are often very difficult to deploy; difficult to get people to migrate to.

This is directly due to the effort people put into learning the old tools, added to the difficulty they expect in learning the new ones. The difficulty of learning applications designed following traditional UI/UX processes stems from two key characteristics of those processes, both are related to the fact that they generally limit themselves to a standard, one-size-fits-all set of widgets (window, icon, menu, pointer, grid, etc.) that were invented in the mid-60s (!) by Ivan Sutherland at MIT, then extended in the late 70s by Xerox PARC. (Little more than cosmetic make-overs have been applied in four decades.) Using standard widgets within overly-simplified interface guidelines creates two problems:

Problem one is that those 60s/70s widgets are limited: showing the raw data and data type (e.g., integer) but not its *meaningful usage* (e.g., how many shares, or new orders). When different usages look exactly the same the only way to recognize one is by reading its label (a waste of screen area—and a slow, distracting, forebrain-limited linguistic process), or by remembering exactly where it is on the screen, or how many clicks away in which menu or tab set. This is rote learning: painfully slow and unpleasant; something not everyone is good at—and few enjoy. Change a label or position and people get frustrated—with good reason: you've broken the "application coping mechanisms" they've laboriously created to help them find what they need in the relatively arbitrary system placements.

Problem two is subtler: since the expert is only shown raw data, once they find it they have to "think past the screen" to connect it with their meaningful mental landscape. Imagine a room you know well. Then imagine trying to walk through that room if every chair, table, picture, rug, etc. were replaced with a labeled rectangle: one of those 60s/70s widgets. You'd find just trying to move frustrating, until you—by rote—memorized how each rectangle is related to the real object in the room. Again, not an analogy; the same cognitive mechanisms are at work: we've accidentally built something like the child's find-the-picture card memory game *Concentration* for today's traders: labeled rectangles hide their cleanly distinct "thought objects" and remove them from their rich and meaningful mental landscapes. This kind of once-removed reference (rectangles referring to the meaningful entities in one's domain of practice) is called "indirection" in computer science.

Tools specified by the CEDM process eliminate both the rote learning and indirection problems. Remarkably: even though the new interfaces are often startlingly different people embrace them immediately, since:

Tools are distinct and recognizable; they look like what they do. Much as a *Scientific American* illustration of a complex physics problem can clearly lay out many objects and relationships, CEDM-designed tools show each task with all its meaningful characteristics. Ironically, they're *easier* to recognize and use even though more information is displayed—only because it's displayed as differentiating, meaning-evoking visuals.

Tools directly show the domain of practice. They're not baskets full of abstract references, to be connected by rote to business processes in an expert's head. There is no indirection. They *are* the business process.

And a redesign that gently deploys task-specific tools makes improving/replacing legacy applications a comfortable development ramp and pleasure to traders, rather than months of productivity lost in begrudged retraining, since:

The most critical, oft-repeated, and frustrating tasks are addressed first. So people are willing to consider the new approach with an open mind. ("Anything will be better than what I have…")

The new tool works side by side with the old one. So people don't have to learn it all at once, and if they need to, in a high-pressure moment, they can fall back on their automatized behavior and use the previous tool. Deploying as "sidecars" to the old application also means developers can first optimize the most business-critical tasks, not having to immediately re-code the 80% of a system that's required but rarely used.

The new tools are recognizable, comfortable, quicker, and less risky. So people actually want them, and may demand that new functionality be specified and designed in the same way. They may also demand that legacy functionality be re-coded to match. This verifies the methodology's success.

The old system can be painlessly retired. Over time, the new tools will cover all the necessary business tasks and people will just naturally stop using the previous system.

…Guides Infrastructure Development by a Precise Understanding of Business Needs

A Cognitive Engineering deconstruction of a business task shapes a better front end, but it also helps shape a more cost-effective, better organized, and more responsive back end:

Tools contain everything that's necessary, and only what's necessary (3). The same encapsulation that focuses a trader's attention on the exact information and actions needed for a task provides an exact roadmap for which services need to be provided by the infrastructure—*and which services need* not *be developed*.

How and why data is used, not just what. CEDM itself (and the tools that it specifies) make clear how data is used, and therefore how it needs to be provided. This can inform proper keying and caching strategy when designing a database, or indicate where two systems need to be brought into sync if, e.g., one publishes every second and one every hour.

A more targeted back-end is doing less. So it can respond more quickly to demands placed on it. It can also be less complex, so developers can spend more time optimizing it and responding to new requirements.

Summary: New Approach, Entirely New Class of Results

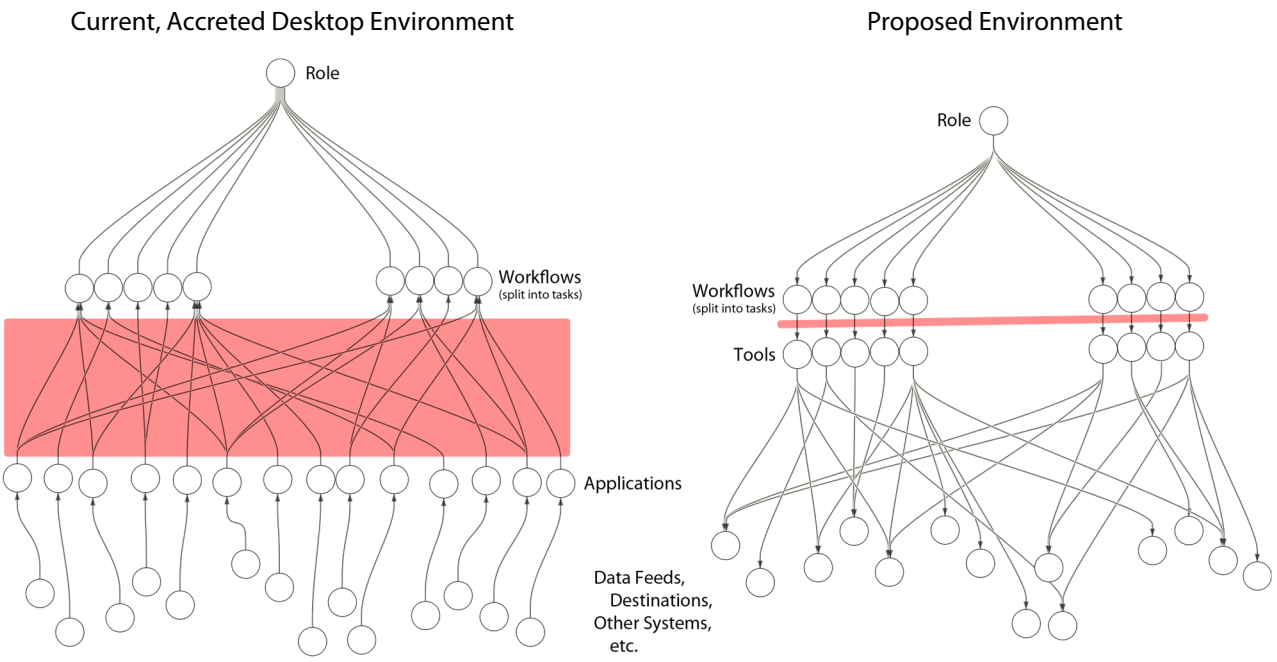
The Cognitive Engineering Design Methodology applies basic scientific findings in a new way, to the emergent problems in today's complex ecosystem of desktop applications. It addresses problems barely recognized because they play out on a higher level of abstraction, and it addresses them with rigor and precision, using processes and information existing disciplines have no access to.

The results it delivers are therefore disproportionate, since it operates in virgin territory. And they add to or multiply the results of other technological efforts, since they're orthogonal. And many of the other classes of innovation efforts—especially the increasingly complex ones—need more sophisticated ways to get their findings *into our expert's minds* before they can become a living part of real business workflows and create value for the firm.

We can leverage our newest tools effectively only after we reduce the cognitive load caused by current desktops.

Appendix A:

Cognitive Engineering Refactoring of a Typical Trading Environment



The ecosystem of desktop applications can be refactored to minimize load on the true scarce resource in today's trading/sales/analytic environment: the minds of the people involved.

The left diagram represents how applications have historically been developed: driven by system capabilities, e.g., data sources or places to take action. The right shows an ecosystem of tools developed to support specific tasks.

The red area in each highlights where people spend their time and mental resources to switch between applications or tools. On the left business tasks typically require many applications, so there are many mental jumps between systems to accomplish each task. On the right, people need to switch only when they change tasks. A complete Cognitive Engineering needs analysis identifies all the data sources and actions needed for each task and makes sure it's all in each tool.

Note that there would be just as many unnecessary switches even if all the applications had compatible user interfaces, so standard UI/UX practices have little to offer on this issue. There *is* a modest gain from compatible user interfaces—but the primary gain comes from reducing the number of task switches, and from the fact that the switches are made *between* tasks: they don't interrupt a complex thought process.

The multi-connect complexity doesn't immediately disappear, but it's hidden in the inside the infrastructure—dealt with *once* by engineers, not *every day, by every trader, for every trade*.

And as time goes on, back end systems will be shaped more by what's necessary to the business, not just what's possible with the technology. Service developers will have more systematic access to the business goals (via CEDM schematics and outlines), so they can reap the gains of a domain/problem-mapped architecture: simpler overall structure, related data centralized or synchronized, and more flexibility to follow changes. And infrastructure development will be more cost-effective: implementing only what's really needed to drive business.

Tools are scoped narrowly for specific tasks; they're lighter-weight efforts than current do-everything applications, designed around data feeds and technology. Developing them is quicker, and they're more customizable.

Appendix B:

Glossary of *Cognitive Engineering Design Methodology* Terms

Cognitive Engineering is the rigorous application of Cognitive Science, Visual Perception, Psycholinguistics, Memory Studies, Psychophysics and other areas of science to enhance human performance. The *Cognitive Engineering Design Methodology* was developed over 20 years by W. Bradford Paley, his colleagues, and his students in the Engineering Department of Columbia University; it details a Cognitive Engineering approach of studying expert behavior to help develop computer systems meant to support and enhance the expert's ability to accomplish their goals. It is orthogonal to related fields (e.g., UX, Interaction Design, Information Visualization), differing primarily in its depth and fine-grained study of the internal mental representations of experts. It is most effectively practiced very early in the system/application design process since the workflow/task decompositions it generates can guide development of infrastructure and business logic as well as final visual interfaces.

Exemplar is a carefully worked-out still image or prototyped active illustration of a screen or software tool, meant to share the visual essence of a real tool, but executed before a full Cognitive Engineering deconstruction of the subject task has been completed. (It may actually work if it's implemented, but it may also lack certain specific decision-making parameters or represent some things less optimally than it would with if designed after the full deconstruction. It can be used as a Proof of Concept, or to socialize the tool or the design approach, and can also be the starting point for a full tool design process.)

Role, ***task***, ***workflow***, and ***tool*** are common words, used herein with a slightly more specific meaning related to the Cognitive Engineering Design Methodology.

A ***role*** is the function of a person, usually an expert in some domain (e.g., a trader, client, or analyst), as exhibited within the context of an organization.

A ***workflow*** is a continuous set of processes carried out by someone in a role to accomplish role-related goals. It may span seconds or weeks, though within the trading domain typically spans minutes. It may be temporally overlapped or multiplexed with other workflows (i.e. attention may be switched away to another workflow and then back).

A ***task*** is a small, typically self-contained segment of a role. Tasks can be identified in way that sounds loose but is in fact well-defined, as "things one might put on a to-do list." This task-defining heuristic works because people naturally segment their workflows into "automatized" tasks: one knows when one is done with one task and can switch attention to (re-activate mental concepts related to) another task.

A ***tool*** is a piece of software designed to support the execution of a task. Tools can support multiple tasks, but this works best when those tasks are adjacent in a workflow or strongly related in other ways, otherwise the affordances needed in the tool for the unrelated tasks can be distracting, or prevent the addition of affordances that better support the tool's primary task(s).

Automatized means that a set of actions has been learned so well (e.g., tying one's shoes) that it can be accomplished without conscious control. Experts initially learn work-related sequences consciously: as propositional memory; repetition allows the sequence to make the transition to procedural memory (called "muscle memory" by athletes and dancers). This has the benefit of allowing the sequence to be accomplished quickly and frees the conscious mind to attend to other issues. It has the downside (for interface designers) of preventing experts from being able to answer the seeming simple question "what do you want?" because experts generally *no longer have conscious access* to the individual actions and needs within an automatized sequence.

An ***affordance*** is an opportunity for action in the world, e.g., a doorknob is an affordance for opening a door, a chair an affordance for sitting, a time series chart an affordance for understanding how a stock price moved over time. (The term was coined by J.J. Gibson in his book *An Ecological Approach to Perception*, a core resource for Cognitive Engineering. It was later introduced to the user interface design community by Don Norman, but is typically limited in that jargon to refer to buttons, scrollbars, and other standard widgets. We use the more general, original scope of the word because it helps us recognize when a standard widget provides poor/awkward access to the intended action, and therefore gives us the incentive to invent a more effective affordance.)

Schematics are the Cognitive Engineering Design Methodology analog of wire frames. They differ in that they can be tested by users (using paper prototyping techniques). And they provide a concrete basis to allow an economic cost/benefit analysis to explicitly and quantifiably balance development costs with business impact for every feature in a design.

Information Layering is a step in the Cognitive Engineering Design Methodology. It is done after a task has been fully deconstructed and understood, and after schematics have been developed. It uses visual means to direct attention to an affordance in the proportion something has business relevance and at the time it has business importance.