Agile Product & Project Management

REPRINT

CUTTER CONSORTIUM

To Release No More or To "Release" Always

by Israel Gat, Senior Consultant,
Cutter Consortium

This report explains why I consider the whole release concept a myth and addresses a more relaxed approach to the engineering, support, customer, and business design aspects of a release. This is a call to reform the software industry, and quite possibly some related industries.

Executive Report

Access to the Experts

Cutter Consortium is a unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and agile project management, enterprise architecture, business technology trends and strategies, innovation, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you *Access to the Experts*. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts — experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats, including print and online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products, training, and consulting services, you get the solutions you need while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

Expert Consultants

Cutter Consortium products and services are provided by the top thinkers in IT today — a distinguished group of internationally recognized experts committed to providing top-level, critical, objective advice. They create all the written deliverables and perform all the consulting. That's why we say Cutter Consortium gives you *Access to the Experts*.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

The Agile Product & Project Management Advisory Service Executive Report *is published by the Cutter Consortium, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA. Client Services: Tel: +1 781 641 9876; Fax: +1 781 648 8707; E-mail: service@cutter.com; Web site: www.cutter.com. Group Publisher: Kara Letourneau, E-mail: kletourneau@cutter.com. Managing Editor: Cindy Swain, E-mail: cswain@cutter.com. ISSN: 1536-2981.*

Rob Austin · Ron Blitstein · Christine Davis · Tom DeMarco · Lynne Ellyn · Jim Highsmith · Tim Lister · Lou Mazzucchelli · Ken Orr · Mark Seiden · Ed Yourdon

Cutter Business Technology Council

To Release No More or To "Release" Always

THIS MONTH'S AUTHOR

Israel Gat
Senior Consultant, Cutter Consortium

IN THIS ISSUE

For most of my adult life, I have been perplexed by a Pavlovian phenomenon: whenever I, as an engineering manager, released code to manufacturing,[1] the marketing folks reacted by conducting a three-week worldwide analyst tour. As much as I appreciate good public relations for my products, I viewed this phenomenon as a mystery that I might one day solve, perhaps when I retire. As luck would have it, the need to solve the mystery has become more pressing in recent years because agile has transformed the way we develop software. My resolve to postpone exploring this phenomenon until better days arrived came to an end when a frazzled marketing manager said, "Israel, you guys can code faster than my team can market." Flattering as this comment was, it led me to some far-reaching conclusions with respect to the way we develop and use software in general, as well as enterprise software in particular. This report captures the essence of my thoughts on the subject.

First, we examine why I consider the whole release concept a myth. Next, we address a more relaxed approach to the engineering, support, customer, and business design aspects of a release. Taken as a whole, this report is a call to reform the software industry, and quite possibly some related industries.

THE MYTH

Conventional wisdom holds the release concept as a pillar of both the software engineering and the go-to-market processes. From an engineering standpoint, closure, whether real or imagined, is reached on releasing the code. From a business perspective, the release is a well-defined entity that can be discussed with the customer and promoted in the market, irrespective of whether the release is monetized. You can actually think of the release as an "engineering meets the business" construct. Metaphorically, the release is a coin with two faces.

As is often the case with two-faced gods, such as Janus, people tend to forget the dual nature, and that is true of

the release. A body of code that delivers certain features and functionalities is one thing. The use of this body of code by marketing and sales to accomplish business results is quite another. Not only do the two activities differ, but they do not necessarily need to be tied together through a 1-to-1 relationship. For example, as an engineering manager, I oversaw several releases that were successful even though we lacked the marketing resources to promote them.

Like a set of nested Russian dolls, the duality of the term "release" often masks a deeper duality. You can view the end user as the primary target of a release or you can view the release as a vehicle for market development. Philosophical as this differentiation of the release might seem, it actually brings to the forefront essential questions about the nature of software and the nature of the market. We will table these important questions for now and deal with them in depth later on in this report.

The Drinking Water Pool

Consider the following metaphor: A drinking water pool has two pipes — an in-pipe and an out-pipe. The in-pipe fills the pool with water, while the out-pipe draws water out of the pool. Both pipes have faucets on them, as adding water is independent of drawing water and vice versa. For all we know, water added on 5 September might not be withdrawn until 29 October. Assuming an acceptable level of water treatment between the two dates, this kind of asynchronous operation is perfectly acceptable. Obviously, you do not want the pool to overflow or to be empty. However, as long as these two boundary conditions are satisfied, regular water treatment is all one needs to keep things going.

Think of the in-pipe in this example as engineering and the out-pipe as the business. Engineering can post releases at its own pace. The business can selectively choose from the posted releases. In this paradigm, marketing is not obligated to promote a release upon its completion. Marketing might do so in three months; it might choose to promote the current release with another release due at a later time; it might choose to make a release available on a limited basis; or it might choose never to promote a release.

An intriguing question poses itself if you accept this premise of asynchronous operation. If the business is free to determine how it will promote a release in the market, why should engineering be bound to producing releases in the traditional manner? Can everyone benefit from relaxing the constraints that usually *surround* a release and move toward a more flexible, fluid release concept? For example, how about made-to-order releases for preferred customers? Do not infer that I recommend professional services customization and deployment of a standard release for the needs of a specific customer. Rather, I advocate a "custom-tailored suits" approach; we start by taking "measurements" of the customer and produce a custom release by engineering.

Up to a Point

Right or wrong, orthodox business theory teaches us to aim at the market, not cater to the whims of each and every customer in the market. We, of course, aspire to serve as many customers as possible with every new release through segmentation. However, even with excellent segmentation, in software, we typically serve the customer *up to a point*, not fully. The release represents a compromise — one hopes a good one — between the aggregation of all customer requirements and what we can produce under realistic resource and time constraints.

Serving our customers with up-to-a-point software might sound innocuous enough, but I believe it represents the most fundamental characteristic of software. None of us would normally buy a car that lacks a major feature we deem intrinsic to the car's concept and function; say, a steering wheel. We might not be getting a great steering system that affects us viscerally in a certain car, but we certainly get a functional steering system in any car. Unlike software, *up to a point* applies only up to a point with respect to cars: the car is complete without any compromise as to fulfilling the common understanding of what constitutes a car. You might not be able to take an ordinary car off-road or drive it at 200 miles per hour on a racetrack, but a car restricted to reasonable speeds on paved roads definitely satisfies the common definition of a car for most buyers.

If we carefully examine a software release, it is not at all clear whether it is *complete* in the sense discussed in the previous paragraph. Software is categorized as supply chain management (SCM), customer relation management (CRM), network management, or some other kind of software, just as a sedan, an SUV, or a cabriolet defines categories in the car industry. However, a universal definition of completeness does not really exist for any of these software categories. As a matter of fact, the all-too-frequent request for enhancement (RFE) "race" between customer and vendor clearly indicates lack of completeness. Dozens of RFEs per product is not an uncommon phenomenon. Some products actually reach the distinction of hundreds of RFEs cut against them.

The RFE race is shocking enough, but compounding it is a low average percentage of functions and features that customers in enterprise software actually exercise. At *XP 2002*, Jim Johnson of the Standish Group reported that 45% of application features and functions provided in enterprise software releases are never used, while 19% are rarely used.[2] In other words, close to two-thirds of features and functions in a software product are practically worthless to both customer and vendor.

Bottom line, the value of the release construct from the customer's perspective is dubious; a small percentage of functions and features are actually exercised, while a great many are flagged as missing (in the form of the RFE). If you accept this premise, the natural question to ask is, "Well, then, is the release concept really useful to engineering?"

Worshipping the Gods We Created Ourselves

In reality, the release is a fiction that we in engineering maintain through the software configuration and change management (SCCM) tool. Look under the wraps of an SCCM, and you will find any number of branches in the code. The code tree, with all of its coding history, is the real representation of the software. The release is merely a branch in the code tree. One branch or another can usually be designated as "The Release." The decision as to which branch constitutes the official release can, and often does, happen at the last minute. For most practical purposes in the engineering context, the release is primarily a packaging and distribution decision.

Viewing the release as a packaging and distribution decision demystifies the concept and puts the rituals that developed around it in proper perspective. Developing or building software (or any other entity) is one thing; packaging and distributing it is quite another. One cannot help wondering whether a one-size-fits-all release philosophy is appropriate for current circumstances and technologies. What if I chose to package and distribute my software in one way to benefit one customer but in quite a different way for another customer, thus providing a different set of features for each? Customer support might not be excited about maintaining two releases simultaneously, but the two customers would love it!

Obviously, the question of scalability is of paramount importance here. Maintaining two or three releases simultaneously is a fairly common practice and is often part of a software company's currency policy with respect to support for previous releases. The question is: what about 50 simultaneous releases?!

Release No More

During a recent presentation at *Agile 2008*, Jeff Sutherland indicated that PatientKeeper produced 45 releases in one year.[3] This data is consistent with the way significant parts of the retail division at Amazon.com function.[4] Obviously, the notoriously long-duration beta exercised by Google is of the same ilk.

A release-per-week pace practically renders the traditional release concept obsolete. The software becomes much more of a fluid, alive, and evolving entity that is not subordinate to an old-fashioned packaging and distribution pattern. For most practical purposes, the software becomes continuous.

To successfully operate in a "the software is alive and always evolving" manner, four questions need to be answered:

1. How do engineering and support not lose control and cohesion over numerous releases?
2. How does the customer keep up with the flow of releases?
3. What business design is appropriate for an "avalanche" of releases?
4. How does "the software is alive and always evolving" thinking relate to adjacent trends in software?

TOWARD A NEW BUSINESS DESIGN FOR SOFTWARE

Thus far, we've examined the traditional way in which we develop, distribute, market, and sell software as a number of discrete releases. The following five observations were made with respect to the myth of the release:

1. Engineering aspects of the release concept can be decoupled from marketing and sales aspects.
2. No customer uses all of a product's features.
3. Most releases fall far behind in the RFE "race."
4. From an engineering standpoint, a release is merely a branch in the code tree, representing a packaging and distribution decision.
5. Modern agile/Scrum software engineering practices enable a release-per-week cadence.[5]

These observations led to the introduction of "the software is alive and always evolving" concept. This view

of software fluidity raises the following question, "What business designs befit such a concept?"

This section of the report suggests that software that is alive and always evolving poses unique opportunities for custom-tailoring solutions directly from R&D. It also describes an effective way to distribute custom-tailored solutions, and it highlights partial disintermediation that transforms the traditional software value chain.

The following sections specify the various elements befitting software that is always changing. These elements are then tied together to form a unified operational model, which can be used in a variety of software business designs.

Hyperproductivity

Case studies by Jeff Sutherland,[6] Cutter Senior Consultant Michael Mah,[7] and myself[8] highlight the power of agile/Scrum techniques. As these studies indicate, you can simultaneously push any two of the following "dimensions" while maintaining parity with the industry average on the third:

1. Time to market
2. Productivity
3. Quality

Compared to industry averages, these case studies report that time to market is at least two times faster than the industry average in parallel with productivity gains of more than two times.

Market-of-One

In a 2007 presentation on the assimilation of Scrum at BMC Software, I specified the blueprint for using agile to create *markets-of-one*.[9] Basically, the speed with which we are able to develop features and functions enables us to customize software to the level that a customer feels it is being treated as a market. We can customize a vanilla software product by adding features required by only a single customer. In other words, we satisfy the particular needs of one customer as if it that constitutes a whole market.

How do we maintain control of the software and cohesion of operation if we continually produce market-of-one releases? The answer lies in distinguishing between fulfilling customer needs versus the way in which we fulfill those needs. Today's market-of-one for one customer can be consolidated with tomorrow's market-of-one for another. The distinct needs of each customer are fulfilled whether each has its own customized version of the software or whether we provide software that incorporates the special requirements of both. Exclusivity cannot be retained in this manner, but fulfillment of requirements for each customer is.

To illustrate the concept, consider two customers for whom two distinct markets-of-one have been developed by adding the special features each requested to a vanilla product. Customer A uses custom-tailored software X; customer B uses custom-tailored software Y. From the vendor perspective, as distinct as X and Y are, they can run in parallel for a period of time. Over time, as additional markets-of-one are developed for other customers, the proliferation of customized versions of the software needs to be controlled. Hence, at a certain point in time, the software vendor brings together X and Y, forming a single code base Z, which incorporates the custom features in X as well as the custom features in Y. In other words, Z is the consolidation of X and Y. Henceforth, Z is maintained by the software vendor as the software that both A and B use. In good time, Z itself is consolidated with other markets-of-one.

The period of time within which X is combined with Y to form Z is largely dictated by operational considerations of the vendor as well as those prevailing at the customer environment. For example, PatientKeeper maintains "at least 45 releases[10] in the field at any one time."[11] Using VPN technology, PatientKeeper tries "to get everyone on the latest release about once a year."[12]

From a vendor perspective, I am inclined to carry out consolidation of markets-of-one as early and as often as possible. To do so, you must be able to conduct regression and stress tests during every iteration. Once you have achieved this level of sophistication in testing, consolidation of markets-of-one is essentially no different than incorporating new features in an iteration or a release. Whether or not you opt to add a hardening iteration is a question that you are best qualified to answer.

With respect to operational considerations in the customer environment, I believe 2008 vintage technology lends itself to fast distribution, provisioning, and usage of consolidated markets-of-one. The following section describes how to accomplish operational velocity in the target environment in a nondisruptive manner.

The Drinking Water Pool Revisited

Earlier, I introduced the drinking water pool metaphor. This had three components — an in-pipe, an out-pipe, and a pool. I made no special assumptions about the pool; it simply serves as a container that holds water.

While a container is a container from a water retainment perspective, software containers can be, and often are,

quite distinct. The whole software distribution discipline, a US $1 billion-per-year industry, has evolved around the subtle differences between various software containers. For example, a container for Windows is different from a container for Unix; distributing a container to a server is different from distributing to a client.

Following suggestions by Tom Bishop[13] and Billy Marshall,[14] I propose using virtual appliances — "virtual machine images designed to run under some sort of virtualization technology"[15] — as the containers that enable speedy delivery of market-of-one software to the customer. Virtual appliances are available nowadays from a variety of sources. Moreover, virtualization standards, such as the Open Virtualization Format,[16] are starting to provide the compatibility required for large-scale adoption of virtual appliances. Regardless of the specific virtual appliance technology/vendor you choose, virtualization technology enables you to deliver your code to your customer quickly for deployment. The time gained through agile hyperproductivity is not lost in a lengthy process to containerize, ship, and provision the software.[17]

Release and Forget

To carry off this market-of-one concept, hyperproductivity must be maintained from release to release. This becomes a problem because of the common practice in the software industry for the development team to move on to a new project after a release has been shipped. The team might be getting into the next "big bang" release that will supersede the current one; it might be assigned to carry out an altogether different project; or it might be disbanded and its members reassigned to other projects. Whatever the case, much of the responsibility for the release is usually handed over to customer support. The model of operation changes drastically: reactive instead of proactive; specific customer orientation rather than holistic software orientation. A "release and forget" mindset often prevails.

The effects of this mindset are compounded by two patterns that tend to creep over time into the maintenance of software:

1. Workarounds, instead of fixing a defect
2. Lax backlog management, which allows defects to accumulate

It should also be noted that usability issues are often handled in a similar fashion. While the accumulation of usability issues might not affect the stability of the code, it most definitely affects user satisfaction.

Figure 1 depicts the way technical debt affects cost of change over time. Unless the shipped software is of very high quality, the effects of technical debt will eventually outweigh any immediate gains, real or imagined, attained by deferring fixing defects to a "better day." H. Thomas Johnson's words are most appropriate here:

> A business organization cannot improve its long-run financial results by working to improve its financial results. But the only way to ensure satisfactory and stable long-term financial results is to work on improving the system from which those results emerge.[18]

The severity of the technical debt situation is vividly demonstrated by examining just one subcategory: defects. As indicated by Capers Jones, the norm for US firms is a cumulative defect removal rate of 75%.[19] The large majority of software maintained nowadays is subject to the corrosive effects of defects and other categories of technical debt. My own anecdotal data over the years

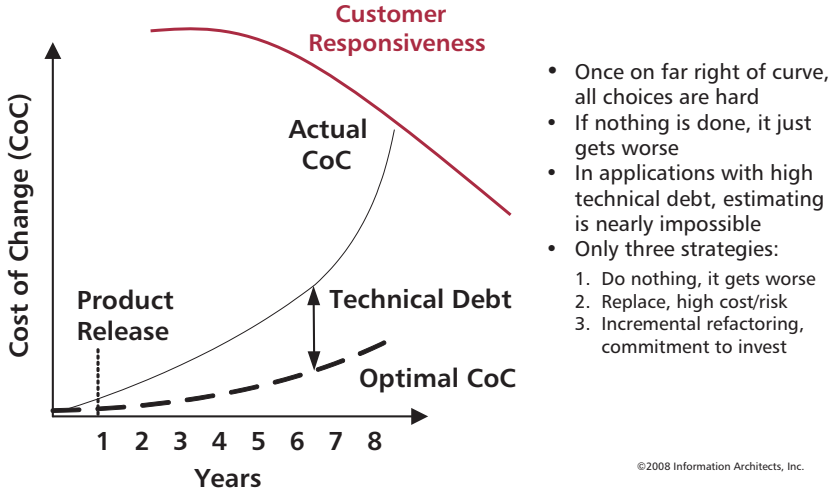

Figure 1 — Technical debt. (Source: Jim Highsmith.)

indicates a severe impact: the ongoing investment in product maintenance could be significantly higher than the income derived from maintenance fees.[20] As Tom Mens points out, "Software maintenance accounts for at least 50% of the total software production cost, and sometimes even exceeds 90%."[21] It makes no difference whether, in accounting terms, you allocate the product maintenance expenses to customer support or to R&D — either way, you pay dearly for accumulating technical debt, including losing the ability to actually do market-of-one because of the higher costs.

Rebalancing the Software Value Chain

I doubt that many software executives would argue against aggressive defect removal at the earliest feasible time. The question I usually get when proposing to "kill defects while they are small," is "Israel, where on earth do I find the money to do so?"

The answer, for companies adopting the market-of-one approach, lies in reducing investment in the various functions between the R&D lab and the customer door. Use the money you save by so doing to enhance the investment in R&D to enable a 95%-99% defect removal state on an ongoing basis.[22]

The beauty in the market-of-one approach is that it reduces various tasks that traditionally had to be performed by intermediaries in the software value chain. For example:

- The more markets-of-one a vendor produces, the less the need for professional services. Much of the customization for the customer is done at the R&D lab.
- The responsibility to determine release content by product management is largely shifted to the customer (who for all practical purposes specifies the market-of-one).
- Virtual appliances enable customer support (and, as appropriate, R&D) to work in a more efficient manner; the virtual appliance provides a controlled environment for debugging and remedial actions.
- With defect removal rate of 95%-99%, customer support's load and role are significantly diminished.

The direct flows — requirements from the customer straight to the R&D lab and custom-tailored releases directly from the R&D lab to the customer — transform the roles of product management, professional services, and customer support. In effect, these functions get partially disintermediated.[23] One can actually think of these three functions as following the path that investment brokers have been forced to take since the 1970s: Charles Schwab's discount brokerage in conjunction with mutual funds and Internet-based market access drastically transformed the role and scope of old-fashioned, full-fledged investment brokerages.

Under the model described above, the customer carries out many of the functions that the vendor traditionally performed inhouse. In addition to specifying the desired market-of-one, the customer participates in agile work at three points:

1. Market-of-one planning
2. Biweekly market-of-one demos
3. Market-of-one retrospective

My experience indicates that the level of time commitment by the customer to collaborate in such a fashion with an agile software vendor is about 100-150 person-hours per market-of-one per year.[24]

A New Paradigm for Software

Table 1[25, 26] synthesizes the various elements discussed here. To summarize, the secret sauce captured in the table is the synergetic application of the agile methodology together with the virtual appliance technology, thereby enabling ultrafast fulfillment of customized user needs. This fulfillment is funded by reducing investment in some of the traditional value chain functions, freeing resources to maintain a 95%-99% defect removal level. The voids left by the partial disintermediation are filled by making the market-of-one customer part of the vendor agile team. For most practical purposes, the market-of-one customer becomes a strategic partner of the software vendor.

BEYOND AGILE

The previous section demonstrated the power of coupling agile methods with virtual appliance technology. Together, these two *ingredients* enable an operational model that allows software vendors to provide markets-of-one for select customers. This level of intimacy brings the customer much closer to the software vendor, benefiting both parties.

In this section, we identify a third "secret sauce" ingredient that further accelerates development and deployment. The critical question then becomes, "What do we do with this ultrafast development and deployment?" We address this question by identifying the business circumstances under which this high speed would be appropriate and also by identifying new business designs that are based on ultrafast development and

distribution. We conclude by pointing out directions where additional research and experimentation related to the use of high-speed development appear quite promising.

Real Working Software Every Two Weeks

As noted above, the secret sauce for the operational model combined two ingredients: agile methods and virtual appliance technology. At the risk of making it a very rich sauce, I suggest adding a third ingredient, which is synergetic with the other two and further enhances end-to-end velocity — *code search techniques*.

To the programmer, such techniques are similar to Google searches for the layperson, one who looks for more information about some entity. The programmer roughly knows what he needs and invokes a code search to locate programs that satisfy that need. For example, a programmer might need a sort routine. To find one, he could search among open source libraries, within his corporate code base, or both. During the search, the programmer finds several sort programs from which to choose. He might determine that a bubble sort routine under a Creative Commons License best suits his needs and incorporate it into his program.

Code search engines are becoming increasingly powerful these days. Search engines such as Krugle[27] already provide the following benefits:

- Programmers can use code searches to avoid reinventing the wheel, which speeds the development process and improves productivity.
- Programmers can search across software change and configuration management containers in which the code might reside. It does not matter whether the code resides in CVS or in Perforce; it will be drawn by the code search. You could actually think of the code search paradigm as using Gmail labels (and Gmail search) versus confining e-mail to a single Outlook folder.
- After using the found code, fixes can be propagated to all programs in which the code is reused. For example, assume the bubble sort routine discussed above was embedded in a program used in ATM machines. If a fix is issued for the bubble sort routine, you can update the 500, 5,000, or 50,000 ATMs at a fraction of the effort it would normally take.

Extensions to current capabilities of code search techniques can make them even more powerful. For example, searches could include not only executable code but test cases. As more people do agile, this would become a slightly different kind of search.

The Ruby programming language is a good example on how effective code search techniques have become. In Ruby, you can search from many reusable open source components. The supply is so rich and the

Table 1 — Design for Alive and Always Evolving Software

Fundamental assumptions	The software is alive and always evolving. Aggressive ongoing refactoring is the most economical way to remove software defects.
Differentiation	Disruptive methodology (Agile).
Operations	Software is maintained at the 95%-99% defect-removal level on an ongoing basis. Customers form an integral part of the agile process and the agile team.
R&D	The development team spends significant amount of time refactoring software on an on-going basis.* A major task for refactoring is consolidating markets-of-one into the main line of the code.
Packaging and distribution	Use virtual appliances to containerize, distribute, and provision software.
Organizational configuration	Partial disintermediation of professional services, product management, and customer support.

*For example, the general norm at Salesforce.com is to devote about 20% of development time to refactoring.

modularization so effective that you should always look there before developing something yourself. Recent reports indicate some open source components are being reused in thousands of projects.[28]

You're less likely to find an appropriate software routine within a corporate environment. Generally, software routines are not reusable without some extra effort to make them so. This practice doesn't often occur inside a typical software company because of the many incentives to produce ad hoc software. The current financial crisis might shed light on the latent opportunity to reuse code within the corporation. The incentive for doing so increases as code search techniques become more granular: it is easier to find and fit a small building block in a program than a large building block.

Each of these three ingredients in our *sauce* — code search techniques, agile methods, and virtual appliance techniques — speeds the "journey" from requirements to deployment in the target customer environment. Between speeding development, accelerating distribution, and simplifying deployment, the compound effect is an ultrafast, end-to-end process. Instead of the biweekly agile demo at the vendor's laboratory, you produce deployed working software every two weeks.

Using Ultrafast Development and Deployment in the Context of Existing Business Designs

Ultrafast development *and* deployment of software open new frontiers for using software. For traditional business designs, I recommend using the following ultrafast development and deployment for meeting two primary needs:

1. **Customer intimacy.** Apply selectively to your top-tier customers. As pointed out earlier, the state of the art at this time is about 50 simultaneous markets-of-one.
2. **Penetrating a new market niche.** Apply as the means to experiment and learn what customers in the new niche really need.

The sauce described above is applicable in different ways. A company might choose to exploit fast development and deployment to provide concurrent custom-tailored releases to its top 50 customers. Or it might follow a model like Google's, opting to deploy 50 releases a year. You would have only one active release at a time under this plan, but many fast releases.

Evolving New Business Designs Around Ultrafast Development and Deployment

Retrofitting ultrafast development and deployment of software into an existing business design can take you only so far. The speed you gain developing the software is easily wasted due to the need to align the various links in your value chain. You might discover that speeding up development and deployment breaks the operational assumptions under which downstream functions operate. For example, you might easily overwhelm your revenue recognition department, your auditors, or both. Likewise, you might overwhelm a function such as IT operations in your customer's value chain.

Evolving a business design around ultrafast development and deployment capability is a more promising way to capitalize on the compound potential of code search, agile methods, and virtual appliance techniques. The following examples illustrate this approach:

- **Money for nothing and your change for free.** In his *Agile 2008* presentation,[29] Jeff Sutherland outlined the way to use agile flexibility in software development contracts. The heart of his scheme is exponential accumulation of value in well-run agile teams versus linear, fixed-price payment terms in the contract. The customer can terminate the contract with the software vendor at any time in which the accumulation of value is sufficient for its purposes, paying just for the amount already billed plus 20% of the unbilled contract value. For example, in Figure 2, the customer pays 60% (50% + 0.2 x 50%) of the amount originally specified in the contract for 90% of the value.
- **One-time purpose software.** Zoho provides an on-demand application marketplace[30] where developers and users can contract for one-time markets-of-one. The Zoho Creator[31] is the vehicle that facilitates interactions between users and developers.
- **Disrupt-the-slow.** Sutherland recently proposed the following "marauder" strategy as a way to exploit the speed of agile:

  > Find a company that has a large amount of resources. Set them loose like pirates on the ocean and they seek out slow ships and take them out. Wildcard did this in the preloaded credit card market and took 100% of the business in the market segment in six months with Scrum.[32]

  It should be emphasized that this is a horizontal strategy. You would not look for opportunities in financial services, healthcare, or any other vertical market. Instead, identify opportunities based on speed of the development process.

Extensions of this strategy are straightforward. For example, you could choose a target according to how slow a company is with respect to reconfiguring its business. Such strategy is likely to be well suited for private equity firms.

- **Ultrafast development to induce change in downstream functions.** When you develop code faster than it can be marketed, sold, or recognized for revenue purposes, you provide a lever to drive business change. The key here is that driving down the cost of change makes change easier and more palatable. Living the agile philosophy means looking at slow speed areas as "opportunities" rather than "barriers to change."

WHERE DO WE GO NEXT?

You could further develop the concepts presented in this report in various ways. The following three areas seem quite promising for additional research and experimentation:

1. **Compress the value chain.** Reexamine the drinking water pool metaphor as a way to separate software development from its use. In this metaphor, disconnect marketing and sales from the "faucet" installed on the out-pipe. Instead, connect the customer directly to the faucet. The customer can now turn on the software faucet, drinking as little or as much as he needs, when he needs it.
2. **Software produced by hyperproductive agile teams as a low-cost input in the technoeconomic system.** With additional traction, software produced by hyperproductive agile teams could potentially satisfy the following four conditions established by Carlota Perez[33] for an input to be a "key factor" in technological revolution:
   a. "Clearly perceived low — and descending — relative cost"
   b. "Unlimited supply for all practical purposes"
   c. "Potential all-pervasiveness"
   d. "A capacity to reduce the costs of capital, labour and products as well as to change them."

   Although agile has been getting good traction in various software development organizations, it is not clear yet what will be the suitable vehicle for massive diffusion of agile. After such a vehicle is found, an intriguing question might be, "What is the potential effect of agile on the current technoeconomic cycle?"[34]

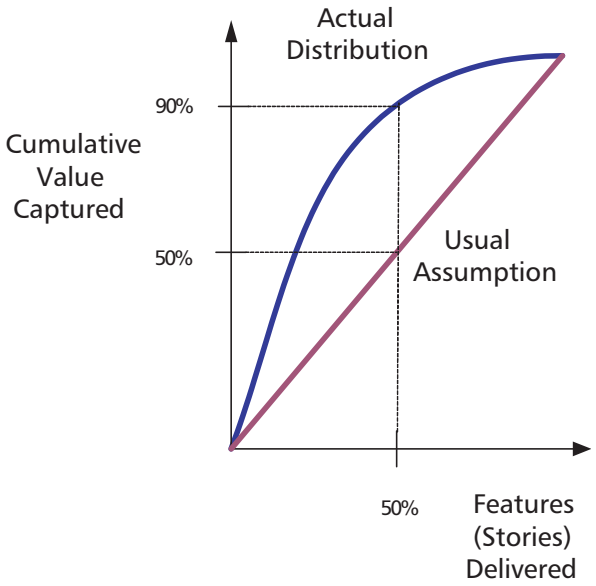

Figure 2 — Nonlinearity of value capture. (Source: Jim Highsmith.)

3. **Being agile without Agile.** The fundamental principles of Agile[35] are not necessarily restricted to software development. Applying fundamental agile principles in other spheres of life could greatly improve best practices beyond software, enhance the diffusion of agile, and lead to enrichment of agile methods along dimensions that we can't yet foresee. As an example, Rally Software is already using Scrum-style stand-up meetings for managing the business.

Much of the agile work that has been carried out in the past few years has focused on "blocking and tackling," on the nuts and bolts of how one implements agile as distinct from what could be accomplished through agile. Such a focus is quite natural for a methodology that struggles to establish itself. Having reached the point where many executives are willing to give agile a try, the time has come for agile practitioners to change the mindset under which they operate. The new mindset I propose is evocatively captured in the following metaphor by Antoine de Saint-Exupéry:

> If you want to build a ship, don't drum up people together to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea.

ACKNOWLEDGMENTS

I am indebted to Cutter Fellow Jim Highsmith, Melody Locke, and Walter Bodwell for reviewing an earlier version of this report and astutely commenting on it.

ENDNOTES

[1]The term "RTM" (release to manufacturing) will be used in this series. In a conventional software process, RTM precedes general availability by a few weeks.

[2]Johnson, Jim. *Chaos Report*. The Standish Group, 2002. Presented at *XP 2002*.

[3]Sutherland, Jeff et al. "Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams." *Proceedings from Agile 2008.* IEEE Computer Society, 2008, pp. 339-344.

[4]Furthermore, Amazon's retail division tends to skip the preproduction stage.

[5]Recent reports point toward even faster cycle time based on on-demand prioritization. For example, see: Ladas, Corey. "Scrum-ban." Lean Software Engineering, 10 July 2008 (http://leansoftwareengineering.com/ksse/scrum-ban).

[6]Sutherland, Jeff. "Pretty Good Scrum: Secret Sauce for Distributed Teams." Keynote presentation at *Scrum Gathering Fall 2008*, Stockholm, Sweden, October 2008.

[7]Mah, Michael. "How Agile Projects Measure Up, and What This Means to You." Cutter Consortium Agile Product & Project Management *Executive Report*, Vol. 9, No. 9, 2008.

[8]Gat, Israel. "Dancing with the Agile Goddess." Presented at *Agile 2008*, Toronto, Canada, 4-8 August 2008 (http://submissions.agile2008.org/node/4377).

[9]Gat, Israel. "The Bitch Goddess of Agile Success." Presented at *APLN Summit*, Richmond, Virginia, USA, 18 October 2007 (www.agileprojectmgt.com/slides/gat07.pdf). Variants on the market-of-one-through-agile theme were delivered thereafter to various APLN chapters in the US as well as to *Agile 2008* in Canada.

[10]For the purposes of this discussion, one can think of a release as the equivalent of the custom software for a market-of-one.

[11]Personal correspondence with Jeff Sutherland.

[12]See 11.

[13]Personal correspondence with Tom Bishop.

[14]Marshall, Billy. "When 'Agile' Becomes 'Fragile'." *Billy on Open Source*, 30 April 2008 (http://billyonopensource.blogspot.com/2008/04/when-agile-becomes-fragile.html).

[15]"Virtual appliance." Wikipedia (http://en.wikipedia.org/wiki/Virtual_appliance).

[16]"Open Virtualization Format Specification." Distributed Management Task Force, Inc., 4 September 2008 (www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf).

[17]A particularly intriguing aspect of virtual appliance builders is their use in the context of the "cloud." The reader is encouraged to build, catalog, and provision an application on such readily available environments as the Amazon Elastic Compute Cloud (EC2). The ease and simplicity of so doing surpasses description in words.

[18]Johnson, H. Thomas. "Manage a Living System, Not a Ledger." *Manufacturing Engineering*, Vol. 137, No. 6, December 2006 (www.sme.org/cgi-bin/find-articles.pl?&ME06ART83&ME&20061210&&SME&#article).

[19]Jones, Capers. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 2001.

[20]Typical maintenance fees for enterprise software hover in the range of 15%-22% of license.

[21]Mens, Tom. "Introduction and Roadmap: History and Challenges of Software Evolution." In *Software Evolution*, edited by Tom Mens and Serge Demeyer, Springer-Verlag, 2008.

[22]For the significance of the 95% threshold, see: Highsmith, Jim. "Zen & the Art of Software Quality." Presented at *Enthiosys Customer Appreciation Day*, San Jose, California, USA, 12 September 2008.

[23]I use the term "disintermediation" here in the broad sense of the removal of an intermediary in a value chain. By partial disintermediation, I mean the reduction in the role an intermediary plays in a value chain. For an in-depth discussion of the subject, see: Evans, Philip, and Thomas S. Wurster. *Blown to Bits: How the New Economics of Information Transforms Strategy*, Harvard Business School Press, 1999.

[24]A customer could, of course, require a few markets-of-one. For example, in the system management arena, monitoring and help desk could be distinct markets-of-one.

[25]Table 1 follows the format established in: Slywotzky, Adrian J. *Value Migration: How to Think Several Moves Ahead of the Competition*. Harvard Business School Press, 1996.

[26]Table 1 does not address such business design elements as customer selection, scope, go-to-market mechanisms, and value recapture. The reason for these "omissions" is simple: one can devise various business models based on the operational model given in Table 1.

[27]See www.krugle.com.

[28]Dinan, Michael. "Black Duck: Open Source Software More Prevalent, Diversified than Thought." TMCnet.com, 9 December 2008 (http://asterisk.tmcnet.com/topics/open-source/articles/47084-black-duck-open-source-software-more-prevalent-diversified.htm).

[29]Sutherland, Jeff. "Money for Nothing and Your Change for Free: Agile Contracts." Presented at *Agile 2008*, Toronto, Canada, 4-8 August 2008 (http://jeffsutherland.com/scrum/Agile2008MoneyforNothing.pdf).

[30]See http://creator.zoho.com/marketplace.

[31]See http://creator.zoho.com.

[32]Personal communication with Jeff Sutherland, 7 August 2008.

[33]Perez, Carlota. "Structural Changes and the Assimilation of New Technologies in the Economic and Social Systems." *Futures*, Vol. 15, No. 5, 1983, pp. 357-375.

[34]Perez considers the current (fifth) technoeconomic cycle to have begun in 1971, when Intel introduced the microprocessor. See: Perez, Carlota. *Technological Revolutions and Financial Capital*. Edward Elgar Publishing, 2002.

[35]In the forthcoming revision of his book *Agile Project Management*, draft of 20 November 2008, Cutter Fellow Jim Highsmith crystallizes three fundamental principles: (1) delivering value over meeting constraints, (2) leading the team over managing tasks, and (3) adapting to change over conforming to plans.

ABOUT THE AUTHOR

Israel Gat is a Senior Consultant with Cutter Consortium's Agile Product & Project Management practice. He is recognized as the architect of the Agile transformation at BMC Software. Under his leadership, BMC software development increased Scrum users from zero to 1,000 in four years. Dr. Gat's executive career spans top technology companies, including IBM, Microsoft, Digital, and EMC. He has led the development of products such as BMC Performance Manager and Microsoft Operations Manager, enabling the two companies to move toward next-generation system management technology. Dr. Gat is also well versed in growing smaller companies and has held advisory and venture capital positions for companies in new, high-growth markets.

Dr. Gat currently focuses on enterprise-level agile deployments. His activities include consulting on R&D transformation, coaching executives on how to implement agile, and developing business designs that take full advantage of agile methods. Dr. Gat holds a PhD in computer science from the Israeli Institute of Technology and an MBA from Clark University. In addition to publishing with Cutter, he posts frequently at *The Agile Executive*. He can be reached at igat@cutter.com.

Jim Highsmith

CUTTER CONSORTIUM

Agile Product & Project Management Practice

Cutter's Agile Practice unites agile methods such as iterative development, test-first design, project chartering, and onsite customers with cutting-edge ideas on collaboration, governance, and measurement to help your firm respond quickly to market changes, increase customer satisfaction, innovate, and deliver high ROI.

Consulting

Agile Assessment

Receive a complete assessment of your organization's use of agile methods, its software engineering practices, and its project management skills and capabilities, performed by Cutter's team of experts and led by Practice Director Jim Highsmith.

Technical Debt Assessment and Valuation

Technical debt is a real cost. Cutter's Technical Debt Assessment and Valuation, led by Israel Gat, helps you determine how much money is required to "pay back" your software's technical debt and answer the question, "Is your software an asset or a liability?"

Agile Enablement

Which agile practices will bring the greatest ROI to your organization? Cutter's Agile Enablement engagement helps you implement agile techniques that are right for your organization, shorten your product development schedule, and increase the quality of the resultant product.

Training

Advanced Agile Project Management: Creating the Flexibility to Thrive

In this two-day workshop, Jim Highsmith explores a variety of advanced topics — from the Agile Triangle to release planning to phase-gate project governance, and more — giving your organization the tools it needs to enhance its ability to deliver successful agile projects and fully embrace the agile ethic.

Agile Business Intelligence

Bring Cutter's Ken Collier into your firm and discover the values, principals, and practices behind Agile Business Intelligence. Learn how you can use this highly iterative and evolutionary approach to building any business intelligence system to help you generate early and frequent analytical results for review and feedback by stake-holders.

Agile Metrics

If you want to ensure your gains from agile are above industry norms, Cutter's Agile Metrics techniques provides real-world metrics of agile projects and how they measure up against waterfall and other projects in terms of productivity, schedule, and quality.

Research and Analysis

Become a client of Cutter's Agile Practice and your team will immediately benefit from access to our online resource library, including Webinars, podcasts, white papers, reports, and articles.

Popular Research:

- "The Agile Triangle — Quality Today and Tomorrow" by Jim Highsmith
- "How Agile Projects Measure Up, and What This Means to You" by Michael Mah
- "To Release No More or To 'Release' Always: Parts I-IV" by Israel Gat
- "Pitfalls of Agile V: Quality Assurance?" by Jens Coldewey
- "The Siren Song of Improving Productivity" by Jim Highsmith
- "Making Agile 'Sticky': Strategies for Long-Term Success with Agile Adoption" by Amr Elssamadisy

Test-Drive the Service

To gain a free trial to Cutter's Agile Product & Project Management resource center or discuss how Cutter's consultants can help you successfully deploy agile practices at your organization, call us at +1 781 648 8700, send e-mail to sales@cutter.com, or visit www.cutter.com.

About the Practice

Agile Product & Project Management Practice

Cutter Consortium's Agile Product & Project Management practice provides you with information and guidance — from experienced, hands-on Senior Consultants — to help you transition (or make the decision to transition) to agile methods. Led by Practice Director Jim Highsmith, Cutter's team of experts focuses on agile principles and traits — delivering customer value, embracing change, reflection, adaptation, etc. — to help you shorten your product development schedules and increase the quality of your resultant products. Cutting-edge ideas on collaboration, governance, and measurement/metrics are united with agile practices, such as iterative development, test-first design, project chartering, team collocation, onsite customers, sustainable work schedules, and others, to help your organization innovate and ultimately deliver high return on investment.

Through the subscription-based publications and the consulting, mentoring, and training the Agile Product & Project Management Practice offers, clients get insight into Agile methodologies, including Adaptive Software Development, Extreme Programming, Dynamic Systems Development Method, and Lean Development; the peopleware issues of managing high-profile projects; advice on how to elicit adequate requirements and managing changing requirements; productivity benchmarking; the conflict that inevitably arises within high-visibility initiatives; issues associated with globally disbursed software teams; and more.

Products and Services Available from the Agile Product & Project Management Practice

- The Agile Product & Project Management Advisory Service
- Consulting
- Inhouse Workshops
- Mentoring
- Research Reports

Other Cutter Consortium Practices

Cutter Consortium aligns its products and services into the nine practice areas below. Each of these practices includes a subscription-based periodical service, plus consulting and training services.

- Agile Product & Project Management
- Business Intelligence
- Business-IT Strategies
- Business Technology Trends & Impacts
- Enterprise Architecture
- Innovation & Enterprise Agility
- IT Management
- Measurement & Benchmarking Strategies
- Enterprise Risk Management & Governance
- Social Networking
- Sourcing & Vendor Relationships

Senior Consultant Team

The Cutter Consortium Agile Product & Project Management Senior Consultant Team includes many of the trailblazers in the project management/peopleware field, from those who've written the textbooks that continue to crystallize the issues of hiring, retaining, and motivating software professionals, to those who've developed today's hottest Agile methodologies. You'll get sound advice and cutting-edge tips, as well as case studies and data analysis from best-in-class experts. This brain trust includes:

- Jim Highsmith, Director
- Sanjiv Augustine
- Christopher M. Avery
- Paul G. Bassett
- Sam Bayer
- Kent Beck
- E.M. Bennatan
- Tom Bragg
- David R. Caruso
- Robert N. Charette
- Alistair Cockburn
- Jens Coldewey
- David Coleman
- Ken Collier
- Ward Cunningham
- Rachel Davies
- Tom DeMarco
- Lance Dublin
- Khaled El Emam
- Israel Gat
- Sid Henkin
- David Hussman
- Ron Jeffries
- Joshua Kerievsky
- Bartosz Kiepuszewski
- Brian Lawrence
- Mark Levison
- Tim Lister
- Michael Mah
- Siobhán O'Mahony
- Ken Orr
- Roger Pressman
- James Robertson
- Suzanne Robertson
- Alexandre Rodrigues
- Dave Rooney
- Johanna Rothman
- David Spann
- Rob Thomsett
- John Tibbetts
- Jim Watson
- Robert K. Wysocki
- Richard Zultner