

The Marriage of Business Dynamics and Software Engineering

Ram Chillarege, *Chillarege Inc.*

Few technology disciplines have the emotion and belief wrapped around the issue of process that software engineering does. As a community of engineers, we are prone to discussions of exactness, and often these quickly disintegrate into arguments of right and wrong methods. Our heated debates about software process, models, and methods can make us forget, however, that a software engineering process is only as good as its ability to create products that meet market needs.

Businesses and markets are great opinion equalizers; because their survival is tied to people's buying habits, they must account for emotion, rationality, behavior, and change. When we fail to realize that business dynamics drive software development—with cost, time, and product quality as outcomes—we make bad judgments. If these don't negatively affect the software development organization, they usually affect the businesses that create the software, which is worse.

To better understand how software process and product intertwine, we must focus on both software development fundamentals and market dynamics. We also need a realistic view of what is both humanly and technically possible. Evolution takes time, and implementing industry best practices won't raise productivity levels if organizational resources can't realistically support them. This article therefore takes a holistic

view of the software product business and offers guidelines for instituting development processes that match market needs and improve competitiveness.

Business dynamics

The software product business has been highly profitable in the past two decades. Several software product businesses demonstrated gross profit margins around 80 percent, returns unheard of in any other modern, legal industry, including transportation, finance, insurance, and health care. But the successes and profits have hidden many ills. When the good times pass, the consequences of poor engineering on efficiency, margins, and return on investment appear as surprises.

Figure 1 shows a financial model commonly touted as a product development ideal. The chart shows a product's cumula-

Integrating market evolution concepts with software engineering processes can help us better evaluate process consequences and institute development processes that meet market needs.

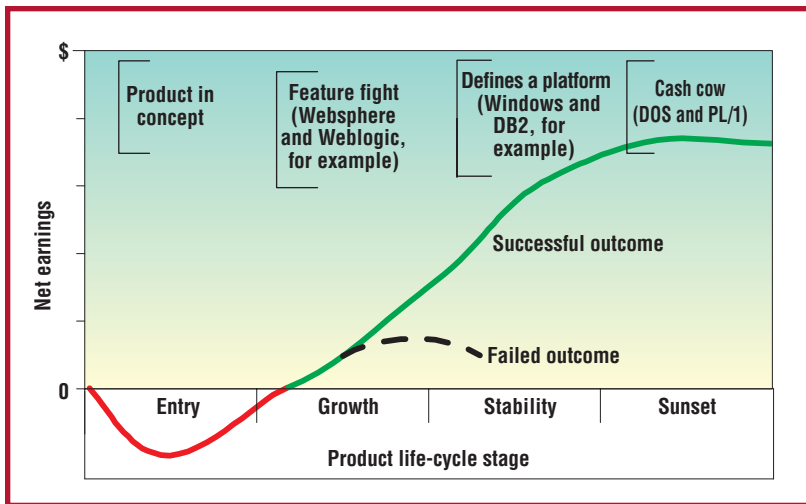


Figure 1. A financial model of software product development.

tive net earnings against a time line, with the classic “hockey stick” shape depicting an initial negative period of investment until payback, when the curve crosses the zero line, and then a meteoric rise following payback. Many successful products have met such an ideal, but of course no guarantees exist in the software business. Products commonly die out, and failure can occur at any stage over a lifetime that might stretch 20 to 30 years.

Software business models and financial implications also differ based on whether the software is a consumer product, an outsourced development project, a service, or an internally funded application development project. Each presents stakeholders with different financial models, margins, and long-term possibilities.

Life-cycle stages

I divide a software product’s lifetime into four stages: entry, growth, stability, and sunset. These stages help us characterize product content, market evolution, viability, and the overall metamorphosis that occurs as software grows from small teams and tens of thousands of lines of code to large teams and millions of lines of code.

Projecting the stages against the financial curve lets us visualize a holistic business model where each stage marks a significant transformation in market forces, competitive positioning, and business advancement opportunities. In *Crossing the Chasm*,¹ Geoffrey Moore discusses how technology product marketing evolves from introduction and market development among early adopters to product growth in mainstream markets. To truly understand how product life cycle affects process, however, we must recognize that separately, but in tandem, the

product code base, process, technology, tools, development team, and other attributes also evolve.

Entry. This marks an individual product concept’s market introduction. Of course, other products within this market might launch at a later time. For example, *Managing Your Money (MYM)*, an early product that defined the personal finance space, had few competitors in the entry stage, but the growth stage saw a new entrant, Quicken. Today Quicken dominates the space, with strong competition from Money, an even later entrant. These products are now arguably in the early stability stage following a shakeout in the late 1990s.

Growth. Moving from entry to growth represents a significant life-cycle shift. Software makers hope their products will either turn profitable or, better still, see strong growth in profitability. At the same time, the product and market move away from novelty toward more basic business values such as predictability and reliability. Feature wars erupt as competition grows and software makers seek to gain customers and market growth. Customers relate to bottom-line value, and products become part of the overall business structure. The growth stage can be long, punctuated by technology, infrastructure, and competition changes.

Stability. Products that reach this stage become legends. They represent a unique combination of business and engineering acumen that creates platforms and new market segments, supplants older products, and promises the sky. But stability, like so much in the IT business, is relative. A stable product can always be unseated. Linux, for example, is changing the operating systems business. Although dominant operating system makers continue to exude confidence over their platforms, responsible product managers lose sleep over the Linux threat.

Sunset. This stage sees no substantial business growth. Products can still make money, and some can even become cash cows²—provided manufacturers find ways to boost profit margins and don’t merely focus on large market share when growth slows. The product creates value in its clientele’s com-

Table 1**Business value drivers in product evolution stages**

Rank	Entry	Growth	Stability	Sunset
High	Innovation, time to market	Features	Predictability	Reliability
Medium	Features	Predictability	Reliability	Maintainability
Low	Predictability	Time to market	Features	Predictability

mitted dependence, and manufacturers reward customer loyalty with high reliability and excellent service.

Product evolution. Products in the entry stage tend to be lightweight and have little or no legacy burden, but as they grow larger and more complex, once-small products become heavy platforms. We can visualize this change using the rough and ready lines-of-code measure. Although specific numbers vary widely, an entry product could easily survive with 10 to 40 thousand lines of code (KLOC). However, competition that drives a feature war might bloat the product to a few 100 KLOC during the growth stage. When a product reaches stability, 10 to 15 years later, it could have a million lines of code. Of course, such progression isn't always monotonic—developers often restructure successful products for performance and manageability and thereby reduce their overall size. But more often than not, average product size grows. Trimming a product significantly costs development resources that are often in short supply.

Along with content changes, products often see major personnel changes as they evolve. The entry-stage innovator usually doesn't lead a product through its growth stage, nor do growth stage heroes negotiate the stability stage. This isn't merely a skills issue. Several factors conspire to change the environment, including weariness, skills, attrition, and geography.

Value shifts

A quantitative measure doesn't really give the full scope of change from one stage to another. Although the financial model affects life-cycle stages, it doesn't determine them. So what can we use to recognize these shifts and guide our actions?

We can capture how products change between stages by recognizing shifts in what the market values. Over a product's life cycle, market values change, and different characteristics become dominant and drive business. Table 1 ranks market values by their relative significance within each stage and shows how these change over the product's life cycle. To some extent, contemporary beliefs influence product management, development, and marketing priorities.

In the entry stage, innovation and time to

market help grab attention and create an image of leadership. Novelty often hides quality and performance shortcomings, but such a product's market lead lasts only as long as no other major players exist. Values shift as other players enter the market and novelty becomes less significant.

The growth stage begins a significant shift in market values' relative ranking. Time to market takes a back seat to features as companies target products to a wider customer base. New customers especially want business enablement, standards, ease of use, and domain specificity. The end of the growth stage sees consolidation and shake-out of second-tier products. Growth slows, but the market is large and doesn't resemble that of the early growth stage.

Stability presents a different playing field with products vying for leadership, not just survival. The transition from growth to stability sees values shift from features to interoperability. Companies are also gauging how large the market can become and whether it marks the beginning of a new segment. The stakes are high and the players big, and classic business values such as predictability and quality reign.

Among the value changes, several prove particularly important for our discussion. Innovation and predictability present an interesting comparison. Figure 2 shows how the relative rankings of innovation and predictability exchange positions over a product's life cycle.

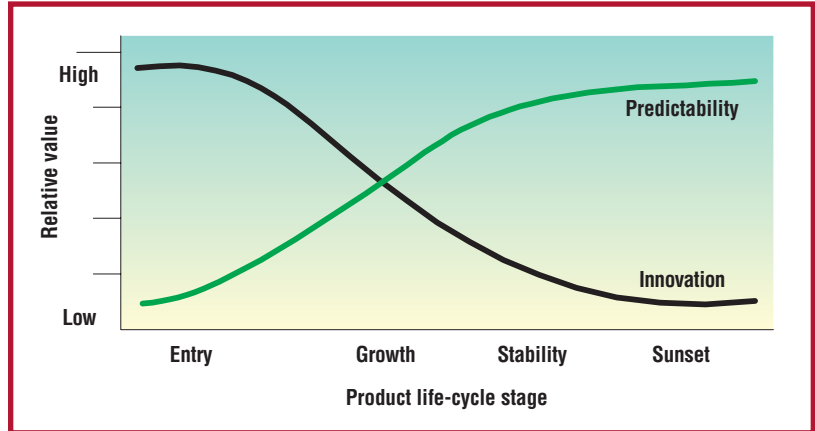


Figure 2. Innovation and predictability shift in importance over a product's life cycle. The x axis isn't to scale, and the growth stage can be much longer than it appears here.

Table 2**Process attributes ranked by process model**

Process attributes	Iterative	Spiral	Gated	Waterfall
Iterations per release	High	Medium	Low	Low
Fast feedback	High	Medium	Low	Low
Speed to change	High	Medium	Medium	Low
Scalability	Low	Medium	Medium	High
Predictability	Low	Medium	High	High
Distributed development	Low	Low	High	High
Multiproduct integration	Low	Medium	High	High

Innovation, which ranks in the top tier during the entry stage, falls off to the lower tiers over the product's life cycle, and crosses paths with predictability somewhere in the middle of the growth stage. We could argue that a product's success hinges on its performance in this stage, and interestingly, neither innovation nor predictability is particularly significant here. What drives product success has varied throughout the software industry's evolution, and it's difficult to predict what the next driver will be. In the desktop era, features drove product growth and competition; in the dot-com era, liquidity reigned. Current trends appear to favor standards, but time will tell. Regardless of the driver, as the growth stage progresses, innovation and predictability clearly reverse positions.

Some values, such as quality, don't change as dramatically. The interpretation of a quality aspect might change, or a market's growing maturity might demand more of some aspect of quality, but a low-quality product rarely succeeds for long. That said, quality aspects such as reliability must keep pace with increased sales volume. I call this the large-volume effect. Although a company can massage a less reliable low-volume product through better customer service, soon the math does not work out to keep up with service and warranty costs, especially in markets that demand low pricing and, therefore, higher-volume sales. Companies must also carefully manage consumer product usability because quality depends almost entirely on how compelling the application is.

The software industry's maturity now calls for higher entry-level quality. For example, thanks to the Internet, client seats can ramp up much faster than ever before. Products with zero-footprint clients that need only a Web browser to run become instantly available to a global user base, and a low-quality application will therefore fail in the marketplace long before it gains a foothold.

Process attributes

When we understand how market values shift over a product's life cycle, we can more easily identify process models with attributes that accentuate market values in a particular stage. As time passes and the values' relative order shifts, this model guides us in making appropriate process changes.

Our software engineering community has debated process models forever, largely without the context of a business model. Consequently, each side argues its position, whether for more or less regimen, with the vehemence of a recent convert. As with any passionate position, a process argument's validity holds within bounds and a degree of isolation. To have a truly fruitful dialogue, however, we must develop a more global perspective of process models.

Chapter 7 of *Rapid Development*³ provides an excellent discussion on the various process models' core concepts, and *Managing the Software Process*⁴ takes a practical view of models' execution aspects. *Agile Software Development*⁵ discusses the philosophies behind successful development teams. As we analyze process strengths and weaknesses, we can compare them based on a common set of attributes. Table 2 rates four process models on such attributes as scalability and predictability. This relative ranking captures each process model's intrinsic strengths and weaknesses.

Although such broad categorization helps us develop a mental model of what a process can offer, we can't forget that software is essentially a labor-oriented business. People implement a process as a series of tasks, roles, and activities, and management philosophy guides this implementation. Therefore, no realized process ever represents a true implementation of the conceptual model. No rules forbid adapting ideas from one model to another, and often a blend results that includes both strengths and weaknesses of the process attributes.

The market doesn't care what process model developers use; the engineering and the process models are only a means to a business end. Customers care about the delivered product, its quality, and credibility of continued service. In the end, we see how "right" a process model is in how it helps a company meet market demands while creating a competitive advantage.

The marriage model

A good marriage between business dynamics and software engineering process arises from a good match between market values and process attributes. An engineering process should help deliver a product while accentuating market value. Although developers could deliver against every value the market demands, this proves unrealistic from a practical point of view. Often, resources are limited and opportunities transient, and the prudent development organization recognizes limitations and ensures its products satisfy at least those values most in demand. This is where the relative ranking of values proves essential. The lower-ranked values aren't irrelevant, but we clearly see the folly in delivering to a lower-ranked value at the cost of a higher-ranked value. That said, we must make these trade-offs within reasonable limits and using good judgment.

In any marriage, people change over time, and what worked at one stage of the marriage doesn't always work in another. Likewise, as market values shift, software development organizations must examine whether the current process model delivers to current market values and offers the most efficient way to run development. It's possible to produce a product with any process model, albeit at the cost of efficiency and quality. A mismatch between process attributes and market values eventually creates cost, increases cycle time, and reduces quality, and other pressures that can break down a once successful engineering operation.

Entry to growth

A low-burden development model best serves entry-stage products. Agile methods with fast feedback address market values such as novelty and speed. They also insure against investing too much in any one area when market direction and other factors remain unknown. The spiral model⁶ captures a more formal elucidation of the process. In iterative methods, however, the process can degenerate to one of continuous change where rework defeats the presumed productivity. Unless developers apply more sophisticated methods to measure and manage development,⁷ inefficiency will result.

The growth stage brings greater demand for efficiency and features. Good product management becomes key, and as develop-

ment teams grow and multiple development streams become common, projects need greater control and predictability. In this context, the less formal iterative methods that capture the early stage can easily break down and spin the development organization out of control. Those with informal processes must reverse their development philosophy to survive. Not all organizations consciously undergo this shift, but most find themselves doing it eventually—and far more painfully, later, if they survive.

Growth to stability

The growth stage often stretches a decade or more, and the engineering processes at each end look nothing alike. The product's credibility in the market depends on release predictability and quality, with sparing exceptions for those companies that can buy some concessions with market domination. New markets, product cross-compatibility, postrelease services, a services business for customization, and other factors increasingly demand predictability and discipline in the engineering process.

We therefore rarely see any process other than a gated one at the end of the growth stage, and this greater regimentation becomes standard for the product's remaining life. We can thus conclude that the degree of choice for software engineering processes lasts only during the entry and early growth stages, which account for no more than a quarter of a product's lifetime.

Process management becomes increasingly critical in the stability stage as product complexity and thus engineering demands grow. Merely throwing people at the problem is rarely an affordable or effective solution. Most process models are no more than activity models, and conformance is hard to drive in a technology purely dependent on intellectual labor. Instead, we need methods to understand activity models, measure cost, and constantly assess process effectiveness. The notion that a process definition alone will meet these needs is misleading—we need process management to successfully apply an engineering process model.⁷

Stability to sunset

Successful growth-stage companies usually have fairly predictable and robust engineering processes. Exceptions exist, espe-

The degree of choice for software engineering processes lasts only during the entry and early growth stages.

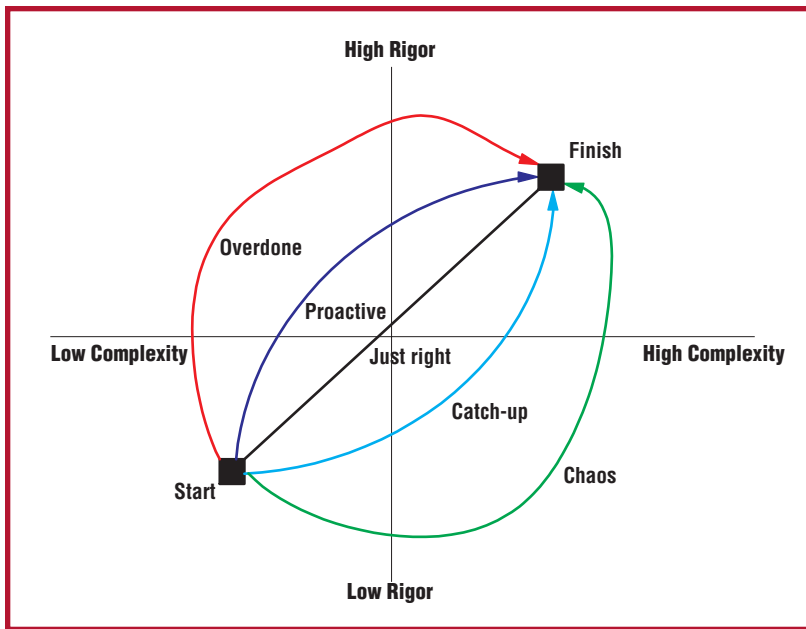


Figure 3. Mapping product complexity and engineering process rigor. A software product can follow any one of the trajectories shown depending on how well we manage both factors.

cially in profitable product lines that manage to afford the inefficiencies of poor process. But the stability phase becomes even more demanding. The product's ability to perform, scale, and interoperate creates a new level of engineering demands to which weak processes usually cannot deliver. Outsourcing's huge cost advantages demand even more robust processes. When such strength does not exist, development organizations can outsource only noncore subsystems.

As development increasingly demands the rigor that gated development methods provide, processes commonly tend toward a modified waterfall. This transition requires organizational maturity and a keen market sense. In addition to rigor, the organization must establish an effective in-process measurement and feedback system because without good feedback, regimented processes become unstable. Also, although much of the development effort may be regimented, exceptions always exist for smaller components that push the envelope on performance or technology. Agile methods serve best to prototype these as long as there's also a clear reintegration plan to the main release.

Stable products will likely be reengineered at least once. Survival in this phase stresses performance, interoperability, and productivity, which might demand code reengineering and restructuring. Although developers often recognize this need early, the heat of the growth stage affords few resources for this purpose. A stable product stands apart from an upstart because of its dependability and interoperability. For example, ask your database administrators

whether they'd choose MySQL, SQL Server, DB2, or Oracle for a new application. The first two are in the growth stage, while the latter two are in the stability stage.


Products in the sunset stage often have a decreasing customer base and can't be made into cash cows without reducing development costs. This requires managing customer satisfaction and warranty costs, and selectively enhancing features to keep the product from dislodging from its client base too soon. Retirement can be fun, but only if it is affordable. Companies managing sunset products must have a well-articulated development process but also improve on this process significantly to increase efficiency and lower costs. Ideally, they'll initiate these changes during the stability stage to avoid burdening themselves with development expenses at the sunset stage. Failure to do so will make their attempts to turn products into cash cows unworkable, and they'll need to discontinue the products.

Although the entry and growth stages require vastly different software process models, the transitions from growth to stability to sunset demand less radical process change. Therefore, gated and more regimented processes will largely govern a product life cycle of 20 years. Discussions that focus entirely on iterative processes and rapid time to market often overlook this fact.

The balance

Figure 3 illustrates the trade-offs involved in managing product complexity and process rigor. Over its lifetime, a product's complexity grows and—ideally—process rigor grows correspondingly. In the real world, however, a product's path can be any one of the curved lines shown in the figure. More likely than not we are either playing catch-up or being proactive. Obviously, poor process management results in the arcs swinging out into chaos or overregimentation.

Managing a software development operation and engineering environment to suit the market is a sophisticated science. The software industry has long been plagued with low efficiency, poor productivity, and sick projects, and

many observers attribute these problems to failure to comply with known best practices. But that is only part of the equation—the business and technical factors intertwine with less quantifiable people issues. Sophisticated diagnostic methods such as orthogonal defect classification⁸ can help organizations pinpoint and prioritize such issues and their impact. Only when the organization can accurately diagnose problems and estimate the return on investment does money spent on process management bear strong business impact. Good methods and tools are necessary but not sufficient. True transformation occurs only when we complement our energies with skills to diagnose, solve, and execute the necessary process change. 

6. B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, vol. 21, no. 5, May 1988, pp. 61–72.
7. R. Chillarege et al., "Orthogonal Defect Classification for In-Process Measurement," *IEEE Trans. Software Eng.*, vol. 18, no. 11, Nov. 1992; www.chillarege.com/odc/articles/odconcept/odc.html.
8. R. Chillarege and K.R. Prasad, "Test and Development Process Retrospective: A Case Study Using ODC Triggers," *Proc. IEEE Dependable Systems and Networks*, IEEE Press, Piscataway, N.J., 2002.

For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.

References

1. G.A. Moore, *Crossing the Chasm*, Harper Business, New York, 1999.
2. B.D. Henderson, *Anatomy of the Cash Cow*, Boston Consulting Group, Boston, 1970.
3. S. McConnell, *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, Redmond, Wash., 1996.
4. W.S. Humphrey, *Managing the Software Process*, Addison-Wesley, Boston, 1989.
5. A. Cockburn, *Agile Software Development*, Addison-Wesley, Boston, 2002.

About the Author



Ram Chillarege is a software engineering consultant whose work focuses on the management–technology interface. He founded and headed the IBM Center for Software Engineering, where he created the Orthogonal Defect Classification method and was awarded the IBM Outstanding Innovation Award for its invention. He serves on the University of Illinois Dept. of Electrical and Computer Engineering board and the

IEEE Steering Committee for Software Reliability, Dependable Computing, and Application-Specific Software Engineering. He also chairs the New York Software Industry Association's CTO Council. He received a BE and ME from the Indian Institute of Science, Bangalore; a BSc from the University of Mysore; and a PhD in computer engineering from the University of Illinois, Urbana-Champaign. He is an IEEE fellow. Contact him at Chillarege Inc., 210 Husted Ave., Peekskill, NY 10566; ram@chillarege.com; www.chillarege.com.

Career Opportunities

PURDUE UNIVERSITY Department of Computer Sciences

The Department of Computer Sciences at Purdue University invites applications for tenure-track positions beginning August 2003. Positions are available at the assistant professor level; senior positions will be considered for highly qualified applicants. Applications from outstanding candidates in all areas of computer science will be considered. Areas of particular interest include security, mobile and wireless systems, scientific computing and computational biology, and software engineering.

The Department of Computer Sciences offers a stimulating and nurturing academic environment. Thirty-six fac-

ulty members have research programs in analysis of algorithms, bioinformatics, compilers, databases, distributed and parallel computing, geometric modeling and scientific visualization, graphics, information security, networking and operating systems, programming languages, scientific computing, and software engineering. The department implements a strategic plan for future growth which is strongly supported by the higher administration. This plan includes a new building expected to be operational in 2005 to accommodate the significant growth in faculty size. Further information about the department is available at <http://www.cs.purdue.edu>.

Applicants should hold a Ph.D. in Computer Science, or a closely related discipline, and should be committed to excellence in teaching and have demonstrated strong potential for excellence in

research. Salary and benefits are highly competitive. Special departmental and university initiatives are available for junior faculty. Applicants can apply electronically by sending a curriculum vitae, a statement of career objectives, and names and contact information of at least three references as a postscript or .pdf file to fac-search@cs.purdue.edu. Alternatively, applicants can send hard copies of their application to:

Chair, Faculty Search Committee
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

Applications are being accepted now and will be considered until the positions are filled. Any inquiries should be sent to fac-search@cs.purdue.edu. Purdue University is an Equal Opportunity/Affirmative Action employer. Women and minorities are especially encouraged to apply.