The Role of Process in a Software Start-up

Stanley M. Sutton, Jr., *EC Cubed*

"If things seem under control, you're not going fast enough."

—Mario Andretti

In the start-up environment, process technologies and methodologies that focus on advanced levels of process maturity can be out of place. This article gives an overview of a start-up's relationship with process and gives guidelines on how to apply process to a start-up company.

It's been said that "speed kills." However, for a start-up software company, especially one targeting the dynamic commercial marketplace, speed might be what keeps the company alive. Although other factors contribute to the success of a new software product or producer, for a start-up company that aims for "better, faster, cheaper," "faster" usually takes precedence.

Addressing the "better" aspect, one of the Software Engineering Institute's Capability Maturity Model's[1] main goals is to improve process and software quality. The CMM also aims to improve process predictability and manageability, which addresses "cheaper"—at least to the extent of better controlling costs.

These software process goals reflect the diversity inherent in software development: the varying contexts in which it occurs and beliefs of how to most successfully produce the software in such contexts (where success is ultimately defined by the organizations' survival). A highly systematic and measured approach to software process will suit certain circumstances—for example, negotiated or safety-critical software. However, that approach might be incompatible with the typically fast-paced, reactive, and innovative world of commercial software development. This is especially true for highly dynamic application domains such as Internet-based electronic commerce, and even more so for start-up companies without an established product, customer base, or revenue stream.

This contrast, between the need for speed in getting products out and the need for deliberation in maximizing process control, raises the questions of whether process or process maturity can help a start-up software company and how process and process maturity are relevant, if at all. To what extent does the theme of *process* accommodate this diversity of contexts and viewpoints? This article examines some of the issues surrounding these questions and particularly considers how you can apply process to start-up companies developing commercial products.

Characteristics of Start-up Companies

EC Cubed (see the sidebar) exhibits many characteristics that are widely representative of software start-up companies. These characteristics reflect both engineering and busi-

A Representative Start-Up

EC Cubed (www.eccubed.com) provides a representative example of a start-up software company focused on the commercial market. Founded in December 1996, EC Cubed produces a suite of Java-based application components for the rapid development of distributed e-commerce applications. The company has received multiple rounds of venture-capital funding from a growing number of funding sources. At the time of writing, EC Cubed employed about 100 people and was continuing to grow. Most employees served in a technical capacity and were divided between engineering and professional services groups. The company has had two chief executives plus an interim CEO, and in October 1999, it relocated its company headquarters from Wilton, Connecticut, to Westborough, Massachusetts, to be closer to a larger technical labor pool and to facilitate relationships with current and potential partners and customers. Since then, the company has continued to grow, spread geographically, and evolve its business and the way it works.

ness concerns, which define inescapable constraints under which start-up companies must operate.

Youth and Immaturity

The most basic characteristics of start-up companies is that they are new, or at least relatively young and inexperienced compared to more established and mature development organizations. This means that they have very little accumulated experience or history. Thus, immaturities typically exist not only in such companies' process capabilities but also in their organization.

Limited Resources

Another typical characteristic is that resources are limited. The first resources invested in a company typically focus on outward-looking activities: getting the product out, promoting the product, and building up strategic alliances. The sooner the company can accomplish these activities, the better its chances for survival.

Multiple Influences

In its early stages, a company might also be particularly sensitive to influences from various sources: investors, customers, partners, and competitors (both actual and potential). Divergent influences might also exist within the company. To complicate the situation, these influences, although seemingly critical, could be inconsistent. Consequently, the company might continue to adjust and readjust what it does and how it does it.

Dynamic Technologies and Markets

New companies often get caught up in the wave of technological change sweeping the software industry (and IT industries in general): new network technologies, proliferating communications channels, an increasing variety of computing devices, new programming languages, new system architectures, new object and distribution technologies, and more. New software companies are often established to develop technologically innovative products, and developing these products, in turn, might require cutting-edge development tools and techniques.

Start-ups versus Established Companies

Of course, the conditions and challenges that characterize the typical, commercial start-up company can also apply to more established or noncommercial software companies. More established companies are not chronologically youthful, although they might still be immature (in a process sense) or even outdated (in an organizational sense). All software companies must address in some way the issues of time to market, cost, and quality, although the relative emphases can vary (for example, with the type of software produced and the type of market targeted). Also, all software companies are to some degree subject to changing technological and economic environments (although this, too, will vary with the type of software and type of market). Still, software start-ups, especially in today's commercial marketplace, tend to focus the characteristics described here to an extreme degree. They represent a software industry segment that has been mostly neglected in process studies, and it is possible that lessons drawn from start-ups also apply to other development organizations.

Start-ups versus Small Companies

Some start-up companies are also small companies, although by no means are all start-up companies small or all small companies start-ups. Small companies, like start-ups, can face challenges in adopting process-oriented technology and methodology, although the reasons for this vary. (Two important reasons commonly cited for small companies are the lack of CMM process improvement initiatives or their inapplicability to the practices in small development organizations.[2]) However, small companies, especially those that are established, can have certain advantages over start-ups. These can include fewer internal communication and coordination problems; greater flexibility and reactivity (due to lower organizational inertia); a foundation of established products, partners, and customers; and possibly a greater shared history and vision. Thus, start-up software companies and small software companies have some problems in common but face different sets of challenges under different circumstances. In recent years, process improvement for small companies has received attention. Examples are given elsewhere as this article focuses on process concerns in start-ups.

Is Process Immaturity Inevitable?

Process immaturity in start-ups might indeed be inevitable. Many common characteristics of start-ups militate against maturity:

- Process maturity requires repeatability. The SEI CMM's[1] second level is the "repeatable" level, in which specific processes might vary between projects, but the goal is project management for repeatable success. Implementers who develop and use realistic project plans based on previous projects typically expect success. Yet the youth and dynamism of start-up organizations practically preclude repeatability. New processes occur under new circumstances, and the processes are subject to change as those circumstances change. You might adopt new directions and approaches in response to influences from cooperating or competing organizations, and you might take on special, one-time-only projects for customers whose business is considered critical for company survival. Thus, not only individual processes, but also their manageability, could greatly vary.
- Process maturity requires organizational maturity, but young software companies might lack the corporate direction, organization, and experience needed to foster and maintain such maturity. Process maturation requires buy-in up and down the corporate hierarchy and across a breadth of corporate departments. The relevant corporate groups might not appreciate process maturity and might view it as desirable but remote and irrelevant to more immediate and critical needs.
- Process maturity requires resources, both human and technological, for process definition, implementation, management, training and awareness, measurement and analysis, and more. What start-up software company couldn't spend more on speeding up or expanding production, service, marketing, and sales? In start-up situations, especially, limited resources battle with numerous needs, leaving process seldom marked as a top priority (and correctly so, as some might argue).
- Process maturity's primary benefits do not address young, commercial software companies' primary needs. Process maturity tends to promote product quality and process predictability and reliability. In contrast, a start-up company often aims to minimize the time to market and other goals might suffer.

For all these reasons, you can't reasonably expect substantial process maturity in young, commercially oriented software organizations. Indeed, it's not clear that maturity in the traditional sense is appropriate for such organizations.

Does Process Matter?

Process, if not traditional process maturity, can crucially affect start-up companies. In one sense, a start-up's ability to execute is perhaps its key capability. Software development still occurs by means of some process, even if by means of an ad hoc or improvised process. And software development necessarily remains an engineering activity that depends on some degree of rigor and order.

Processes that a company creates in its early stages might form the basis for, or contribute experience toward, established and repeated processes as the company matures. The ability to repeat a process can critically affect a start-up's success. At a company such as EC Cubed, the opportunity to repeat processes arises because the company produces a family of components that the company can treat more or less similarly. Repeatable processes span the life cycle, including development, quality assurance, documentation, and training. Processes can

Process, if not traditional process maturity, can crucially affect start-up companies.

Process definitions focus on the higher levels of the life cycle, their flows and feedback between stages.

be repeated across components and (in some cases) shared between them. Repeating these processes effectively thus forms an important part of planning and scheduling, achieving proficiency, and instituting process improvements.

Of course, a company might eventually define, manage, and improve repeated processes, possibly following the SEI's CMM or through some other plan. Indeed, repeated processes demand process improvement, which can be feasible regardless of the organization's formal level of process maturity.

To the extent that you can define development processes, they can be useful for several purposes. You can design them to help assure product quality and minimize process costs. Process definitions can set expectations about development activities and provide a framework for process and project management, including planning, resource estimation, and milestone tracking. Development processes provide a means to state and communicate important engineering practices across development groups and the broader organization. They also help inform new personnel who join the company as it grows. In the CMM,[1] process definition (Maturity Level 3) follows process repeatability (Maturity Level 2), but in a start-up company, process definitions can help promote and establish repeatability—and its resulting benefits.

Process models and process management provide a basis for relationships with other organizations, including collaborating hardware and software vendors, subcontractors and outsource organizations, service providers and systems integrators, and customer organizations. (Subcontract management forms one of the Key Process Areas (KPAs) in CMM Level 2.) Indeed, a new software company's survival and growth might depend on alliances with other, more well-established organizations, but these other companies might very well be at a higher level of process maturity. Moreover, such alliances might provide a start-up with motivation and leverage for improving its own process capabilities.

How to Approach Process

A start-up company has numerous options for supporting processes that address its more immediate needs while also facilitating long-term process improvements. The following recommendations primarily speak to companies developing commercial products, although some contracting organizations and systems integrators might also find them relevant. You can also view these recommendations as analogous to CMM KPAs that can increase a start-up's chances for survival.

Define Software Processes

You should define your software development processes. As noted earlier, process definitions can serve many purposes related to the understanding, communication, execution, and management of development processes. Process definitions also form a basis for formulating potentially crucial relationships with other organizations.

However, you must appropriately define software processes. You should emphasize capturing significant process features rather than fine details. When change is the only constant, fine details often become irrelevant. Additionally, when speed is a factor, elaborate instructions can be a stumbling block to progress. In either of these cases, an overly detailed process definition can be counterproductive and is likely to be ignored, in practice if not in spirit.

EC Cubed develops products according to a defined software life cycle that spans requirements specification through release (and includes quality assurance). Process definitions focus on the higher levels of the life cycle, their flows and feedback between stages. Lower-level details are left to process participants and group managers, who can adapt their activities according to their experience, the product, and the project's contingencies. This reliance on process participants and managers is appropriate and effective for three main reasons:

- Process definition guides the process's overall structure and flow.
- Process participants and managers are generally aware of the process's state and progress and can closely coordinate their actions.
- The overall approach, combining higher-level structure with lower-level flexibility, has proven effective, and its implementers appreciate the approach's benefits—in terms of product quality, project efficiency, and process adaptability. (Benefit appreciation can be a key motivator of process adoption and improvement.[3])

These conditions certainly do not apply to all software development organizations, but you can often find them in start-up software companies.

Remain Flexible

Flexibility, perhaps even more than structure, plays an important role in a start-up company's development processes. You need flexibility to accommodate changes in personnel and infrastructure, product specifications, resource levels, release schedules, and so on. Often, you can't predict or control the external conditions driving such changes (although even that ability is not always helpful, such as with Y2K!). These conditions often require a rapid and timely response. Redefining a process every time the process parameters change can be prohibitively slow and costly. A defined but flexible process is the best way to maximize continuity while facilitating adaptation. Flexibility greatly facilitates handling process exceptions and deviations; a flexible process definition can provide an effective model for the desired results of irregular executions, without inhibiting the handling of irregular situations.

Use the Right Form of Process Definition

Despite all of the research in process languages, the language used to define the process is not particularly important, *provided* that it permits clear communication and statement of the process's critical points.

If, in a process definition, you aim to capture activity and artifact flow, you can represent this in various ways, including flow charts and natural language. Although typical formalisms of these sorts have oft-cited limitations (ambiguity, incompleteness, informality, and so on), your reasons for defining a process might make such limitations irrelevant. For example, if you represent the artifacts you want to develop in terms of files and databases, it lessens the importance of using a process language as a data definition language, and your ability to represent in a process definition the flow of artifacts (as opposed to their schemas or interfaces) might suffice.

If you need a process's more specialized aspects, such as data definition, transactional properties, or concurrent-activity synchronization, selecting an appropriate language might be more critical. A language with more specialized features, such as one based on a database programming language or Petri nets, or even a specially designed software-process programming language,[4,5] might be appropriate. However, even such specialized or powerful languages do not suit all processes or situations. Additionally, you should probably abstract noncritical points from the process definition in any case.

Generalize

In building up a start-up organization's processes, you should give first priority to generally useful standards and protocols. This most affects companies that produce a family of products that it can treat in a parallel way, or when you can plan a number of alternative or successive versions of a product in advance. For example, a common graphical interface standard that applies across several products will afford a greater return on development than developing particular standards for each product in the family. Similarly, generic review processes and test plans that you can readily specialize for each product-family member are more useful than customized processes and plans. When a company is first starting, you might need to define all processes, standards, and guidelines as special cases. However, you should seize the opportunity to generalize when it arises.

Similarly, you should define protocols and procedures for recurrent development activities—such as design reviews, configuration management, or release builds—as early as possible. Such protocols and procedures form building blocks for higher-level activities, and you can reconfigure and reuse them in many processes. They could be useful even before you can fully define higher-level processes and might remain stable and useful even as higher-level processes are prototyped, refined, and evolved.

Analogously, a general-purpose technological infrastructure is important. Examples include configuration management (a KPA for CMM Level 2[1]); problem reporting and tracking systems; planning, scheduling, and notification systems; and intranets. Such technologies support the activities that practically any software development organization will need, yet they are not tied to any particular life cycle, method, or process. You

Flexibility, perhaps even more than structure, plays an important role in a start-up company's development processes.

Software project managers have long recognized the importance of good people for successful software development.

can apply them in various ways in many contexts and reuse them as the organization and its processes evolve. Such technologies should at a minimum accommodate change, but they could also facilitate change. For example, you can use intranets to communicate new processes and put proposed processes up for organizational review, apply configuration management systems to manage process and product versions, and use problem-reporting and tracking systems to record and repair problems with new (or established) processes.

Learn and Reuse

Some of what I've discussed relates to the theme of "learn and reuse." You can generalize and reuse experience gained or lessons learned from one member of a product family or one step in a process across the product family or development life cycle. Unless special needs exist, you shouldn't adopt procedures, protocols, and technologies for one product or life cycle phase unless you can reuse them for other products or phases.

Some experiences from EC Cubed illustrate this point. According to the defined life-cycle process, the Development teams perform a certain level of unit and integration product testing, after which they pass the products to the Quality Assurance (QA) group for more comprehensive testing. EC Cubed adopted the test plans that Development developed and used as the QA's core of testing. However, QA was responsible for more extensive testing than Development performed. These additional tests often revealed bugs in the code that caused repeated cycles between Development and QA. To meet its needs to test against the full range of anticipated problems, QA developed more comprehensive test plans to address a wider range of test cases. EC Cubed prototyped these plans first for some specific products, then generalized them across the product family. At the same time, the company refined the development process to require Development to test the products more thoroughly before releasing them to QA. This testing significantly reduced the number of cycles between Development and QA, reduced the amount of time each product spent in QA, and improved the overall release-cycle time. Appreciating the benefits of more intensive upstream testing, the Development teams began to adopt the more comprehensive QA test plans, thus further raising the quality of product released by the teams and further allowing the QA group to expand its testing scope.

Get Good People

Software project managers have long recognized the importance of good people for successful software development. Most typically, others have linked the significance of a superior programmer or architect to a product's or project's success.[6] One challenge for many start-up companies is to hire and retain such star developers.

In a start-up company, however, technical survival and success might depend most heavily on the executives and managers responsible for shaping, directing, and implementing technical strategies. The importance of people in these roles derives from the need to keep the company focused and moving ahead. It also derives, perhaps even more fundamentally, from the need to define and redefine the direction in which "ahead" lies. This direction shifts continually, due to the organization's inherently evolving nature and the dynamic and unpredictable context in which it operates. Thus, when looking to build a start-up development organization, you need to select managers who are both leaders and navigators.

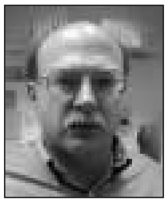
General competence among developers, in all roles and at all stages of the life cycle, remains crucial. However, superstar developers often play less crucial roles than capable but flexible developers. Developers must be able to change direction, take on and execute new tasks, fill new roles, adapt specific results to general cases, and apply previous experiences to new challenges. Thus, when looking for good developers, you should give high priority to general skills and adaptability. Still, you'll need star managers to get the greatest benefits from versatile development teams.

Start-up companies should concern themselves to some extent with process and process improvement, which are the CMM's central themes. Indeed, many of the particular process issues and technologies that the CMM addresses apply to start-up companies. However, start-ups often lack the foundations required for success in the CMM—a historical record of experience, infrastructure for process instantiation and improvement, and repeatable and predictable practices. Thus, while the CMM touches on themes relevant to start-up companies, it can be problematic for a start-up to strictly adhere to the CMM. Consequently, you must select and prioritize process issues and technologies according to the start-up's needs and available resources. A start-up's ability to survive hinges on how (and how well) it does this until the CMM can become relevant and useful. They need approaches that help them operate more efficiently and effectively at lower levels of maturity, where they must first flourish if they are to eventually mature. Although it might be less than ideal from an engineering perspective, start-up software companies often must learn to run before they can walk.

References

1. Carnegie Mellon University Software Engineering Institute, *The Capability Maturity Model—Guidelines for Improving the Software Process*, Addison-Wesley, Reading, Mass., 1994.
2. J.G. Brodman and D.L. Johnson, "What Small Business and Small Organizations Say About the CMM," *Proc. 16th International Conf. Software Engineering*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1994, pp. 331–340.
3. G. Yamamura, "Process Improvement Satisfies Employees," *IEEE Software*, Vol. 16, No. 5, Sept./Oct. 1999, pp. 83–85.
4. L.J. Osterweil, "Software Processes are Software, Too," *Proc. Ninth Int'l Conf. Software Eng.*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1987, pp. 2–13.
5. A. Fugetta, "Software Process: A Roadmap," *The Future of Software Engineering 2000, Proc. 22nd Int'l Conf. Software Engineering*, ACM Press, New York, 2000, pp. 25–34.
6. F.P. Brooks Jr., *The Mythical Man-Month—Essays on Software Engineering*, Addison-Wesley, Reading, Mass., 1975.

About the Author



Stanley M. Sutton Jr. is a visiting scientist at the IBM T.J. Watson Research Center in Hawthorne, New York, where he is a member of the Advanced Enterprise Middleware group. This article is based on experience gained while working as a Quality Assurance engineer at EC Cubed, Westborough, Mass. His interests are software engineering, software and business processes, process languages and environments, middleware, and enterprise software systems development. He received a PhD in computer science from the University of Colorado, Boulder, and is a member of the IEEE Computer Society and ACM. Contact him at IBM T.J. Watson Research Center, Hawthorne, NY, 10532; suttonsm@us.ibm.com.