CRGP Collaboratory for Research on Global Projects
INFRASTRUCTURE ■ FINANCE ■ GOVERNANCE ■ SUSTAINABILITY

Raymond E. Levitt
Chris Fry
Steve Greene
Colleen Kaftan

Salesforce.com: The Development Dilemma

Steve Greene and Chris Fry left their August, 2006 meeting with Parker Harris carrying a far bigger mandate than they had hoped for. As program and development managers, respectively, they had proposed a pilot project to test a radically different approach to software development at Salesforce.com. Founded in 1999 to build a new market in subscription enterprise software services, the company had experienced annual growth rates of 30 to 40 percent, both in customer usage and in head count. Revenues had been growing at more than 80 percent per year, and net income faster than that. But the critical software development function was faltering, even as revenues seemed poised to reach nearly half a billion dollars for 2006.

The existing development processes had been slipping for some time. The pace of releases of new software features—a key measure of value for customers—had slowed from four times per year to once per year, and the latest release was taking even longer than that. Morale was suffering across the organization, and a highly respected senior developer had recently quit after delivering a scathing offsite presentation that criticized nearly everything about the current situation. Furthermore, an infrastructure failure had caused service outages that prevented customers from accessing their customer information during the critical pre-holiday period in 2005. Another outage in early 2006 further eroded users' trust in the reliability of Salesforce.com's software service capabilities.

Harris, one of four Salesforce.com founders and currently EVP-Technology & Products, agreed with Fry and Greene that something had to change. But whereas Fry and Greene wanted to start small and pilot the new method before rolling it out on a larger scale, Harris was thinking big. He'd listened to their description of "agile" or "scrum" development processes compared to the traditional "waterfall" approach, asked a lot of questions, and then instructed them to implement the new method throughout the R&D organization. "We need real change," he said. "Let's skip the pilot and go for the big bang. Our system is broken, and we don't have time to wait—so let's go ahead and fix it all at once."

Professor Raymond E. Levitt, Chris Fry and Steve Greene of Salesforce.com, and Colleen Kaftan prepared this case under the auspices of the Stanford Collaboratory for Research on Global Projects. CRGP cases are developed solely as the basis for classroom discussion, and are not intended to serve as endorsements, primary data sources, or illustrations of either effective or ineffective management practices.

Company Background: The End of Software Revolution

Harris and his co-founders, led by CEO Marc Benioff, proclaimed the 1999 advent of Salesforce.com as the "End of Software Revolution."[1] After a 13-year career at Oracle, the giant enterprise software vendor, Benioff wanted to turn the prevailing enterprise software deployment model on its head. Instead of designing and installing complex, customized software systems and applications to help companies manage various aspects of their activities, Salesforce.com introduced "software as a service" in which customers used web browsers to access centrally managed software applications designed to help run their businesses over the Internet.

Later shortened to SaaS (pronounced "sass"), this new model offered significant upfront cost savings for customers who no longer had to purchase, install, and maintain their own customized enterprise software. Instead, they could pay a monthly fee for hosted services, beginning with Salesforce.com's standard customer relationship management (CRM) programs, which users could shape to fit each company's needs. The new paradigm also came to be called "software on demand" and "software as a utility" for its subscriber-based, externally maintained and broadly distributed availability. The Salesforce.com logo featured the word "Software" with a slash through it, in the fashion of a "No Entry" sign.

The model proved immediately attractive to customers. By January 2001, Salesforce.com counted 1,500 customers, 30,000 subscribers, and 10 internal R&D staff. In 2005, with total revenues above $175 million and net income of nearly $7.5 million, there were some 29,000 customers, 650,000 subscribers, and 200 R&D staff —and the numbers continued to grow. Its IPO in the summer of 2004 set the company's market value at more than a billion dollars. (*Exhibit 1* diagrams the Salesforce.com organization in 2006.)

Analysts and other observers cited Salesforce.com as a classic example of disruptive innovation—a new concept, technology, value proposition, or approach to a market that profoundly alters the competitive landscape.[2] Disruptive technologies were typically cheaper, simpler, and less fully developed than existing mainstream offerings. Their initial customers were those who preferred low cost and ease of use over complex, over-performing products and systems. The challenge for disruptive innovators was to maintain their ability to innovate as they grew larger and more successful.

Salesforce.com's 2005 launch of Force.com, a hosted platform for developing applications on the Internet, opened the gate for customers and third-party developers to create new applications using the Salesforce.com platform. In short order, hundreds of new applications became available for integrating other management tools with existing Salesforce software. And until recently, Salesforce.com had been releasing new user functionality on a regular basis.

---

[1] http://www.salesforce.com/company/milestones, accessed October, 2008.

[2] Clayton M. Christensen articulated the concept of disruptive innovation in *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail* (HPSP, 1997), and elaborated it along with his colleagues in many subsequent publications, courses, and analytical tools.

By 2006, several other nascent applications service providers had followed Salesforce.com into the market for hosted CRM software, but their combined operations were estimated to reach only about five percent of the traditional CRM market. So there was plenty of room for continuing growth—but would Salesforce.com be capable of prospering as a much larger company? Fry and Greene—and many of their colleagues—thought the answer might depend on finding a way to fix the software development process.

The Evolution of Project Management Methodology

Salesforce.com's development process was an offshoot of traditional project management methodology, which had evolved from the tools and processes used for getting work done in the construction, aerospace, and pharmaceuticals industries beginning in the 1950s.[3] In these industries, project management developed largely as a discipline for planning and executing lengthy, complex one-off initiatives such as dams, highways, new aircraft or nuclear submarines, or new drug therapies. In relatively stable market and technological contexts, such projects and programs could be meticulously laid out in advance, and implemented by strict adherence to plan. Detecting, documenting, and correcting any deviations from the plan were critical management functions.[4]

The 1990s movement toward reengineering the corporation used project management practices to standardize and simplify smaller, repetitive tasks throughout an organization.[5] Reengineering suggested that each task should have an identified customer, suppliers, and an owner responsible for managing it to achieve a set of schedule and quality metrics. It created accountability and focus for cross-functional tasks, such as processing insurance claims or bank loan applications, that needed to flow through multiple departments in large bureaucratic organizations. This approach proved valuable in standardizing and simplifying workflows, developing a customer focus, and creating a single point of accountability for critical tasks that could otherwise get logjammed amidst the parochial priorities of individual departments.

At the dawn of the twenty-first century, global competition and the speedy spread of knowledge brought a new type of project management challenge, as firms from rich, high-wage countries scrambled to develop differentiated offerings in rapidly commoditizing industries. Many traditional high-tech manufacturers and service providers responded by shifting from making and selling standard sets of products to delivering more sophisticated custom solutions, uniquely crafted and integrated to meet

---

[3] This section draws on Raymond Levitt, Lecture Notes from *Converting Strategy Into Action*, Stanford Advanced Project Management executive program. See http://apm.stanford.edu.

[4] The tools and processes for doing so were encoded in the Project Management Body of Knowledge (PMBOK), an internationally recognized standard for professional certification maintained by the Project Management Institute.

[5] See, for example, Michael Hammer and James Champy, *Reengineering the Corporation: A Manifesto for Business Revolution* (New York: HarperBusiness, 2001), and Thomas H. Davenport, *Process Innovation: Reengineering Work Through Information Technology* (Boston: Harvard Business School Press, 1993).

each customer's specific needs. Others focused on developing a capacity for continuous product innovation. A few brave competitors opted to offer *both* innovative products and integrated solutions, with all the organizational complexities such a dual strategy implied.

The new array of project challenges ranged from large, complex, initiatives to smaller concurrent daily operational improvements. The single common denominator was the pace of competition. The most dynamic firms in the fastest growing sectors of the global economy—semiconductors, computers, financial services, IT, and non-profits, for example—needed a discipline for managing large numbers of big and small projects in rapidly changing markets and technologies. Product, project, and program managers needed a more flexible, process-light framework in which to apply their organizational methods and tools.

The big CRM solutions companies adopted a "waterfall" approach to software development, which—while more nimble than the traditional aircraft, construction, and pharmaceutical industry methodologies—still relied on the sequential performance of largely separate functional activities, based on a predefined plan and budget. Project priorities were established centrally, usually by customer-facing product managers who determined user needs and then worked across functional lines to push the project forward. *Exhibit 2* shows the typical sequence of functional activities in waterfall development.

Perils of Growth at Salesforce.com

Salesforce.com started out with what Parker Harris called a much more organic development organization, with a small group of people collaborating closely on each release. As the company grew, however, it gravitated naturally toward the phased waterfall development model. But as had become apparent over the years, any change or deviation to the original plan interrupted progress and delayed the target release date.

Parker Harris described the progression since the early days:

> We started out as a small team—like 15 people in the early years. A team like that doesn't really need a lot of process. We were doing releases or upgrading our service as frequently as possible. So when we started the company, we upgraded every four weeks, and then six weeks, and over the years that slowed down, so it's been almost four times a year, and then it was two times a year, and then it was once a year.

Harris recalled how the development process evolved:

> We were perfecting processes as we grew. So we said, OK—we'll definitely need to have a really good section in there for localization, we need to make sure we do usability, and let's add load/stress testing. We added these things simply because they were important, and our customers were looking for more quality, more performance. We also added people, so the team grew significantly. And what used to be an organic process because we just iterated and that was the natural thing to do, as we added

> all these other things we had to do, it grew into this more waterfall process, where everything went down the line.

The R&D group eventually organized around "feature teams"—project groups who were assigned to develop centrally-designated user capabilities. Whenever the product managers identified a desirable feature to develop, they would assign people from across the functions to the team charged with building it. People were often assigned to several teams at once, and many had trouble prioritizing their work across teams.

By the time Steve Greene and Chris Fry came to Salesforce.com in early 2005, the R&D headcount had grown to well over 150. Fry described one drawback he observed in the feature team model:

> There'd be a weekly meeting for each team. Everyone would show up and discuss all the problems for an hour, and then just go back to their desks. There was no accountability. You'd go to all the feature team meetings you were assigned to, and the next week you'd go back and say, "What was it I promised to do?" And nobody was tracking if you were on 20 teams or just one.

Steve Greene elaborated:

> So basically, you're pushing prioritization down to the individual contributors, because they're on five teams and they would decide—on a daily basis—what's the most important thing, and there'd be confusion across the organization because everybody had a different idea of what the highest priority was. And management didn't have visibility into this, because they couldn't attend the 20 meetings every week.

Parker Harris began to notice other problems with the feature team / waterfall approach:

> We'd set formal user expectations and then product management would build a prototype and write a functional spec. Development would then write a technical spec, and on down the line. The timelines were fixed, and everybody would end up padding their estimates and still being late, and then blaming the delay on other people upstream in the process.
>
> At the end, everyone blamed QA [quality assurance], because QA is at the end of the waterfall. Then everyone started pointing fingers at each other, and it also caused really weird behavior where people would work outside the process: "Maybe I shouldn't pay attention to this process, and just be late with my stuff—the release won't go out without it, and if my area's late, maybe we'll get more attention later on"…and so on.

By summer of 2006, the R&D headcount reached nearly 300. Many new hires came from bigger companies such as SAP, with centralized approaches to software development. They brought with them a big company culture, according to Steve Greene, so that by mid-2006, even though Salesforce.com was considerably smaller than most of its competitors, "We were having just as much difficulty delivering releases as the big

companies were." Case in point: the schedule for Release 144 had already slipped several times, and some of the features were still not ready for shipping.

Greene described the internal effect of the delay:

> We had huge morale problems. Because if you're a technologist, you want to build something useful, and you want to get it out there to your customers to use as quickly as possible. You don't want to just talk about it. And we had people who had been at salesforce more than a year—people we hired from larger organizations, with the promise that they could accomplish things quickly here—some of them had never actually released a feature to production.

Fry agreed:

> The institutional knowledge of how to do releases wasn't as strong as it should be. And our development people like to create things that people use, but we weren't getting that positive feedback into the team, so there was low morale across the board.

On the customers' side, the service outages of late 2005 and early 2006 exacerbated the development issues. Customers depended on "dial-tone reliability" for on-demand services, and when they were unable to access their data, they began to question the new business paradigm. If Salesforce.com couldn't assure customers of reliable service, and couldn't even deliver new features on a regular basis, why should any customer base its operations on the hosted services model?

The Shinkansen Project

In summer of 2006, Parker Harris asked Steve Greene and Chris Fry to explore the "release train" development model that many considered to be the driving force behind the rapid development capabilities at eBay. To underscore the ugency of speeding up Salesforce.com's development process, the three named their initiative Shinkansen, after the famed Japanese bullet trains—arguably, along with France's TGV and Germany's ICE, the fastest in the world.

An eBay-commissioned joint benchmarking study in 2006-2006 described the release train system as follows:

> As the term implies, this process is like a train that has a fixed number of seats for passengers and a pre-set schedule. Companies decide in advance the number of releases they'd like to issue each year, as well as the size of each release. The release size is usually based on the required level of effort as defined by person days, or "developer days." Teams of developers work furiously to complete their new products in time for a

certain release train. If they miss the train, they must wait for the next release.[6]

Release trains departed every two weeks on schedule at eBay. When a releasable feature made it onto the train, it would then integrate with other new features and the existing platform much as rail cars couple together, and undergo extensive quality testing. This late integration model proved less attractive for Salesforce.com, in part because the company had recently developed an automated testing capability for continuous integration of most new code. The developers were responsible for testing and integrating whatever fell outside the automated process. Many considered this capability to be an important competitive weapon for Salesforce.com.

Fry and Greene also encountered widespread resistance to the idea of an externally-developed process imposed from on high. After three months of trying to plan how to introduce the release train methodology, the project group decided to abandon the cause.

On the heels of this less-than-promising endeavor, Greene and Fry decided to propose an "agile" development approach in August 2006. Both were familiar with the method, but they did further research to prepare a detailed description for Harris, and set off to persuade him to try it. If Harris agreed, they hoped to turn the Shinkansen team into an agile development team.

Background on the Principles of Agile Development

Fry and Greene began by summarizing the history of agile methods for Harris. The Agile movement represented a philosophy and a shared language as well as an approach to software development. Rooted in the 1980s concepts of lean manufacturing and product development, it had evolved to fit the software development challenges of the early 2000s.[7] Extreme Programming (XP), Scrum, and Lean were similar approaches often subsumed under the Agile umbrella. (*Exhibit 3* lists "Twelve Principles of Agile Software," along with the "Manifesto for Agile Software Development," both authored in 2001 by a large group of professional developers.)

Key Concepts

Agile methodology specifically targeted product developers' needs in fast-moving markets. It used standing cross-functional "scrum teams" to turn out frequent, incremental, and potentially releasable features on a regular basis, commonly after a month-long "sprint." A full-blown release might incorporate three one-month sprint

---

[6] http://www.prtm.com/uploadedFiles/Strategic_Viewpoint/Articles/Article_Content/PRTM_eBay_benchmarking.pdf

[7] Information in this section comes from many sources, including K. Beck, *Extreme Programming Explained* (Addison Wesley, 2000), M. Poppendieck and T. Poppendieck, *Lean Software Development* (Addison Wesley, 2003), and M. Cohn and D. Ford, "Introducing an Agile Process to an Organization," *Computer*, June 2003, pp. 74-78. See also http://www.mountaingoatsofware.com/daily_scrum

cycles, each of which followed a specific rhythm. *(Exhibit 4* charts the activities in a typical sprint-to-release calendar. *Exhibit 5* distills the scrum lifecycle.)

Every thirty-day sprint began with a planning meeting, in which a product owner identified desirable features, or a product backlog of "user stories," in order of priority. (A typical user story might read, "I can predict customer purchasing patterns," or "I can compile hourly sales by distribution channel.") The scrum team evaluated the backlog list, agreed on the number of top priority items they could deliver during the sprint, and committed to doing so. Once that commitment was in place, no further requirements or changes could be introduced during the month-long sprint. The scrum team of no more than 12 people—including product managers, developers, quality engineers, user experience designers, usability testers and technical writers—worked together every day to accomplish their sprint commitments. They used highly visible shared tools, such as wall-mounted task boards and Excel spreadsheets, to keep track of their progress in completing the tasks in a "release burndown." Team members who had completed their tasks would step in to assist other team members who needed help completing their tasks.

Each 30-day sprint ended with a sprint review, in which team members demonstrated their accomplishments to the team and management during the sprint. Sprint reviews were meant to be simple, natural discussions of new capabilities, rather than highly prepared formal presentations. This reinforced the Agile principle of only including code that is truly "done"—i.e., coded, usability tested, QA tested and documented, and thus ready to be integrated into the next release.

The objective was to "deliver fast and deliver early," while avoiding the typical scope creep and roadblocks that tended to plague extended waterfall development projects. Every sprint could produce shippable capabilities, whether the company was ready to release them or not. One intended side effect was to eliminate the crisis management and panic associated with more ambitious, sequential, waterfall-based feature releases with lengthy (and often unpredictable) development calendars.

Roles and Rituals

Agile processes required designating a product owner, a scrum master, and scrum team members (often for multiple teams coordinated by a "scrum of scrums") for each sprint. People tended to stay in these roles over time, so that working relationships and team procedures could evolve to fit the team members' preferences and the requirements of their specific tasks.

Perhaps the most critical daily event for a scrum team was a short morning "stand-up" meeting—typically lasting no more than about 15 minutes—which set the context for the coming day's work. These meetings were held at the same time and the same location every day, and were open to anyone who wanted to attend. There was a strict distinction between those who were "involved" and those who were "committed" in terms of their rights to participate in the meetings.

Scrum experts used a chicken and pig metaphor to describe the difference:

> There is an old joke in which a chicken and pig are talking and the chicken says, “Let’s start a restaurant.” The pig relies, “Good idea, but what should we call it?” “How about ‘Ham and Eggs?’” says the chicken. “No thanks, says the pig, “I’d be committed, you’d only be involved.”

Only committed scrum team members were allowed to talk at daily meetings. Each one had to answer just three questions, designed to reinforce team commitments and remove roadblocks:

- What did I do yesterday?
- What will I do today?
- What is blocking me—i.e., what impediments are in my way?

The scrum master’s job was to remove the impediments every day, so that the team could meet its sprint commitments. Allowing “chickens” to listen in on daily meetings was meant to promote visibility throughout the organization without impeding the teams’ work.

Risks of a Large-Scale Rollout

Greene and Fry were pleased with Harris’s strong endorsement of their proposal and with the mandate to “make it happen” throughout Salesforce.com. However, they both remarked that large-scale implementation of the new methodology seemed much riskier than trying a pilot project. They hated to think about the consequences of another failed attempt to change the development process at Salesforce.com—especially if that failure was organization-wide and highly visible rather than small-scale and contained. Yet it seemed as though they had already hit a low point with the aborted Shinkansen project and the delay of Release 144. Could things get any worse?

Maybe they could. For one thing, no one had ever adopted Agile methods on such a large scale before. Even one of the most prominent experts in the field declined to participate as a consultant, citing the long odds against a “big bang” rollout. Another suggested that to make the change on such a broad scale might be possible, but only with extensive (and expensive) outside help.

Moreover, it was hard to imagine that anybody at Salesforce.com would be interested in a process dictated from above. The senior developers in particular were skeptical of any process or reporting requirements that would take them away from their “real” work. And the Shinkansen project had underscored the not-invented-here mentality that reigned among Salesforce.com’s talented development team. Any major new program would certainly meet great resistance unless the developers could call it their own.

But Parker Harris stood firmly behind the big bang initiative. He wanted Fry and Greene to design and roll out their own version of Agile methodology for Salesforce.com—a hybrid of the existing approaches tailored to the needs, temperament, and circumstances of the in-house developers—and he wanted them to figure out how to do the necessary training internally.

Was Harris correct in suggesting they could do it without outside help from consultants? And would any developers agree to devote their time and energy to a changeover, with Release 144 still stuck in project management limbo? As they thought about how to proceed, Fry and Greene recalled Harris's comment at their August meeting: "Salesforce.com hasn't made a big investment in processes because we haven't known which processes to implant." If Harris was so sure that the Agile was best, how much of the company's time, energy, and money could they expect him to invest into making it work?

Moving Ahead with ADM

Greene and Fry assembled a cross-functional rollout team and adopted an agile approach to the implementation. Members of the implementation team attended outside Scrum Master Training programs to learn about the methodology they would soon be modeling for the rest of the development group.

The new team decided to use an Agile-type process to "burn down the bugs" in the long-overdue Release 144. From one day to the next, they would introduce the new methodology throughout the R&D organization by showing everyone how to apply this new approach to development on a pressing problem. In the process of adopting Agile methods, they also intended to customize the approach to fit the Salesforce.com organization. Their home-grown version would be known as Agile Development Methodology, or ADM.

They divided the broader development organization into some 30 scrum teams, each with six to ten members. The rollout team held 45 one-hour meetings to introduce the new methodology, and proposed to accomplish the bug-burndown for Release 144 in three one-month sprints. All subsequent meetings welcomed open attendance, but only committed team members were allowed to speak. Managers from other areas had a hard time not chiming in, so the scrum masters and team members frequently evoked the chicken-and-pig rule. But just having managers in the room seemed unusual to many developers, and some claimed to resent it.

As part of the new ADM approach, "drive bys"—managers poking their heads into developers' offices to request new features or requirements that they wanted—were outlawed. Parker Harris worked hard to reinforce this prohibition on drive-bys to change the culture of top down opportunistic "software redesign" at Salesforce.com.

Early Progress Report

From the outset, the rollout team encountered both enthusiasm and resistance. Some developers were relieved to find a process that might help them turn out useful features in a timely manner. Several willingly volunteered to invest their time in helping their colleagues get comfortable with ADM. Parker Harris himself promised to be an evangelist for the process, and he directed the company's project management office to support the initiative to the fullest.

But there were detractors as well. Within a few weeks, the early enthusiasts collected a list of negative comments from their colleagues, including the following:

- "Scrum doesn't account for the fact or the reality of the waterfall. You cannot deny this by superimposing scrum over it."
- "It seems like we spend more time talking about scrum…than we spend time talking about and working on salesforce.com."
- "In many ways, scrum seems like an inflexible, bureaucratic process akin to something at the Department of Motor Vehicles."
- "Management is not proactive as we wait for decisions…Scrum gives me the feeling that Big Brother is watching and monitoring everything we do."
- "The lingo is ridiculous!"
- "Ditch the stupid, annoyingly dumb Excel spreadsheet!"
- "Stop trying to implement scrum, and look at how many releases we can really do in a year."

To make matters worse, by the middle of the third sprint in December 2006, it was obvious that the teams would not be able to complete all the prioritized features for Release 144. This created a dilemma for Greene and Fry: should they finish the sprint on time, and deploy what was "done" and releasable, or should they ask Parker Harris and the executive team for a delay?

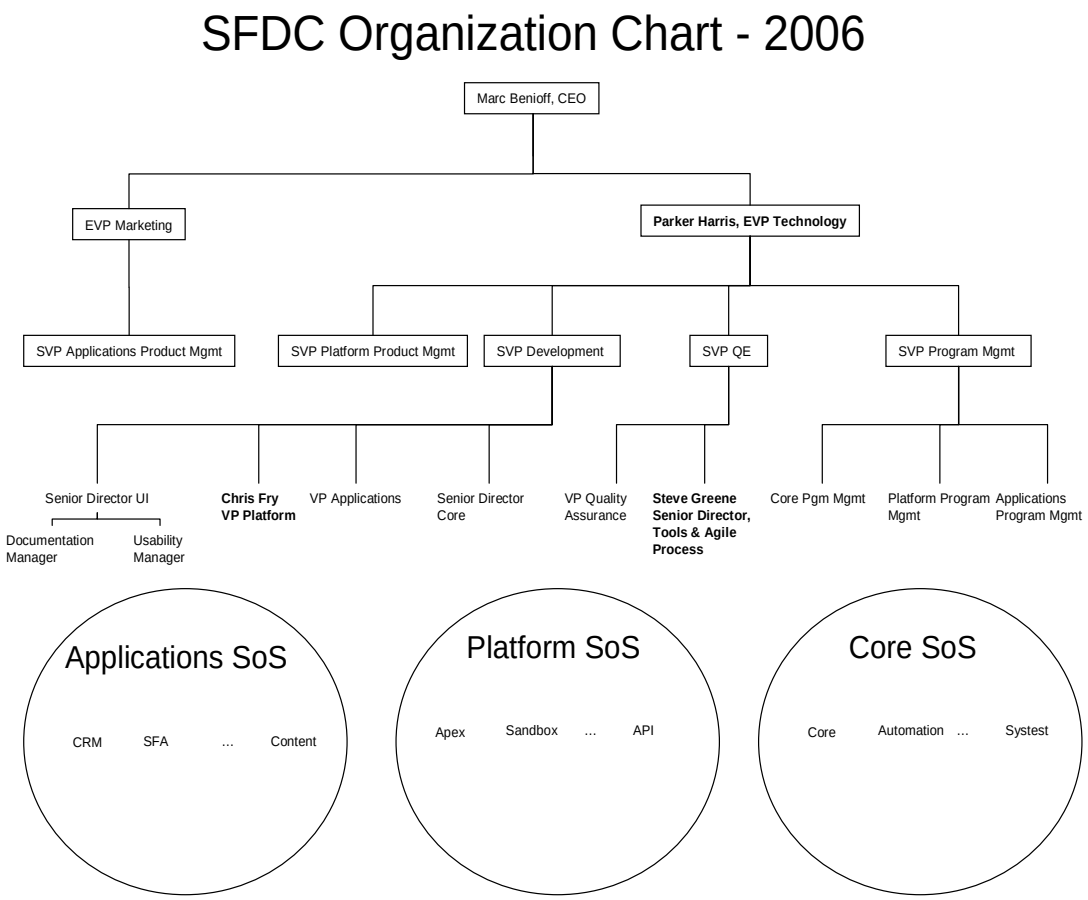
What Next?

On the one hand, to push the release out yet again could undermine the message that Salesforce.com really intended to implement ADM throughout the R&D organization and beyond. But to demo an incomplete set of features might introduce ADM in less than favorable light. Would ADM then go the way of Shinkansen and other failed initiatives? If so, what would be the consequences, internally and in the marketplace?

Most people involved in the rollout believed that they could use the ADM process to teach, adapt, and reinforce ADM itself. This would take time—probably measured in multiple sprints. But where should they go from here? How should they identify the priorities for each successive sprint? For example, should they focus on bringing the most resistant, poor-performing groups up to speed, or start with the ones that were doing somewhat better and eager to learn more? Would it be better to have most groups performing adequately, or should they work on creating a few "stars" to serve as examples for the rest of the organization? And would Salesforce.com be able to keep adapting ADM to meet the needs of the organization as it continued to grow?

Finally, and more immediately, how should they handle Release 144—not to mention 146, 148, and others still to come? Fry and Greene thought the company's continued success could depend on making sure they were asking the right questions, and then getting the answers right.

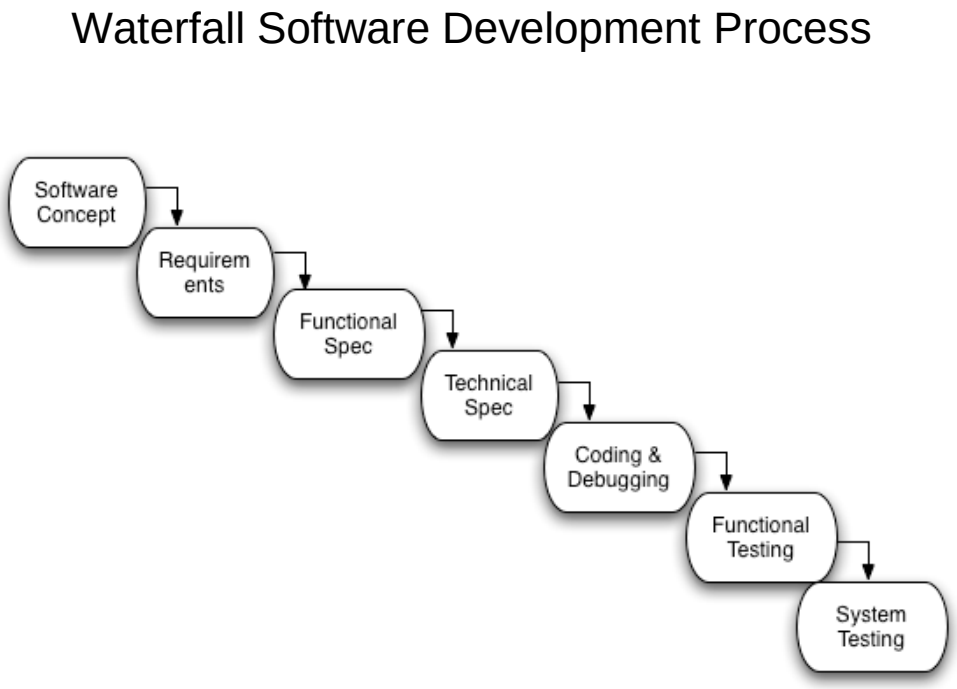
Exhibit 1: Salesforce.com Organization Chart, 2006



Source: Company documents.

Salesforce.com: The Development Dilemma

Exhibit 2: Chart of Waterfall Development Process



Source: Company documents.

Salesforce.com: The Development Dilemma

Exhibit 3: 12 Principles of Agile Software and Manifesto for Agile Software Development

We follow these Twelve Principles of Agile Software:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

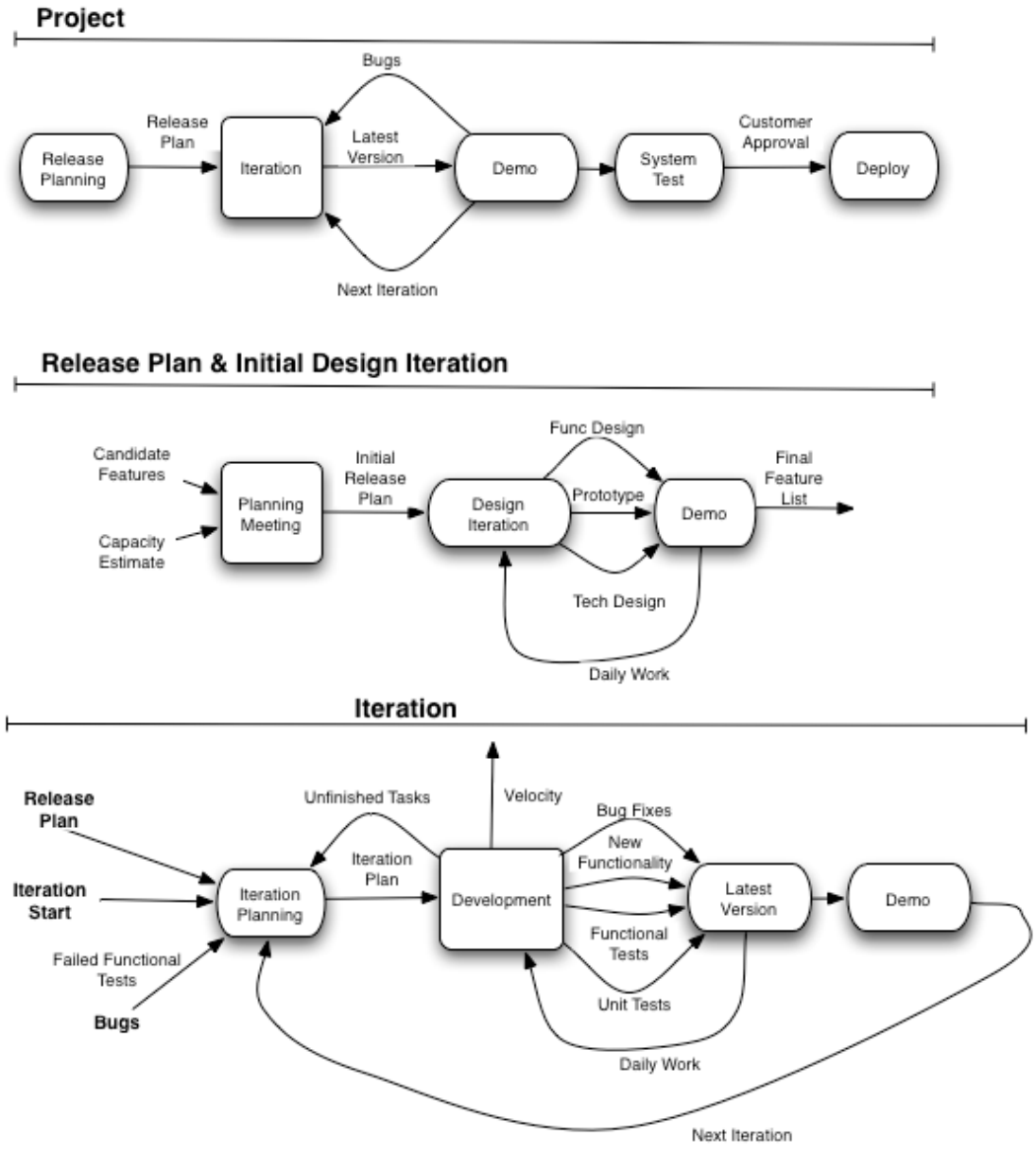
That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.

Source: ***http://www.agilemanifesto.org/***

Salesforce.com: The Development Dilemma

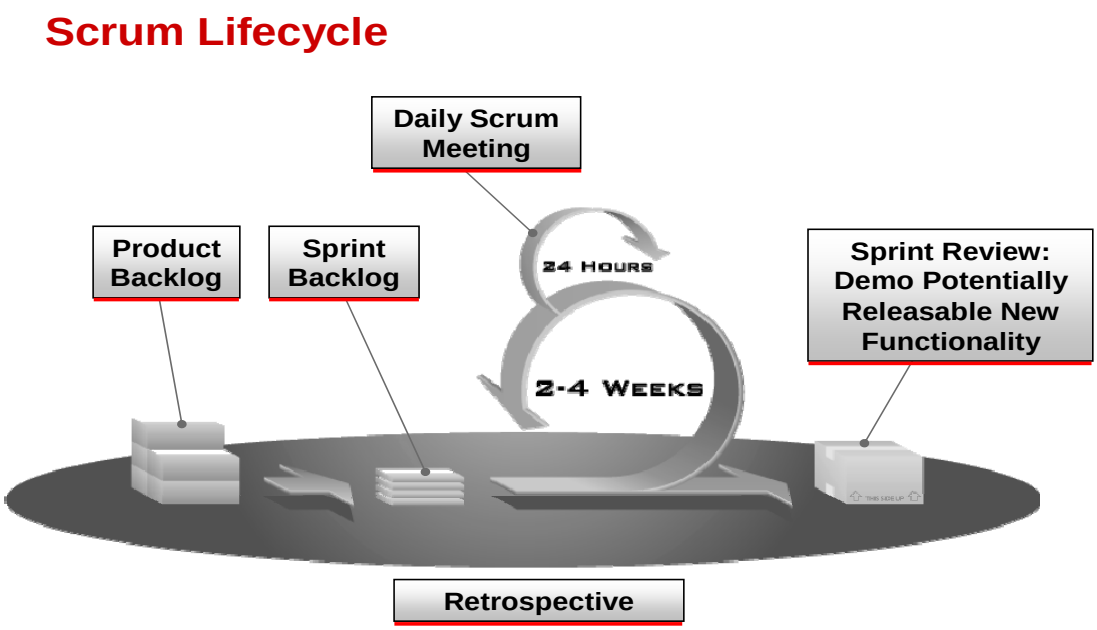
Exhibit 4: Diagram of Sprint and Scrum Activities



Source: Company documents.

Salesforce.com: The Development Dilemma

Exhibit 5: The Scrum Lifecycle



Source: Company documents.