Enterprise Scrum:
Scaling Scrum to the Executive Level

Abstract

Our company manages 25 teams across 6 products using a single top-down Enterprise Scrum. We know of no other company doing this, yet it provides extreme visibility and control at the CXO level. It promotes agile thinking enterprise-wide, driving non-engineering departments to adopt Scrum. We believe it is making us more profitable.

We estimate effort in team months, run quarterly Sprints, assign whole teams to stories, meet in weekly stand-ups, etc. We start, postpone or cancel whole projects. When priorities compete, we often decide by comparing project profit margins, e.g., net-present-value over effort. Our President is the ultimate product owner.

On the individual project level, we still use 2-4 week Sprints and all the trappings of the classic Scrum process.

New challenges arise: Moving teams between projects requires rapid build environment setup. Architects must justify infrastructure projects with net-present-value. Our process became more "lean" to adapt mid-quarter.

1 Introduction

The engineering department is one of the most expensive and sophisticated creative groups in many companies. Its output can determine the company's success, and so maximizing its productivity should be a primary concern of the enterprise: Do its products compel customers to buy? Do its infrastructure projects reduce costs? Are its systems inexpensive to maintain and use?

For single products, Scrum gave us the transparency to answer productivity questions more thoughtfully and the tools to adapt more rapidly [9]. The main scaling challenge small companies faced was this: How do we scale Scrum to maintain agility when a project requires more than 9 people?

Bottom-up techniques arose to scale Scrum to large projects: Scrum-of-Scrums [1], Feature Teams [2], balancing stability with features [3], etc. And yet, even with these methods, as projects got larger, interdependencies increased and visibility decreased.

At the CXO (CEO, VP, CTO, etc.) level in larger companies, decisions and resource planning becomes very difficult with bottom-up approaches: we can see *everything*, but too much information obscures visibility so decision-making becomes hard. We found ourselves pushing some critical decision-making responsibility down, hoping for the best, but could not make optimal trade-offs, whether decisions were made at the top or at lower levels.

Bottom-up techniques do not produce the "sticky-note" simplicity of Scrum at large scales. ScrumMasters can spend hours computing roll-up estimates, assembling product backlog items into Epics and tracking interdependencies. Determining when a project is "Done" becomes a spreadsheet exercise that aggregates many Product Backlog Items. Expensive tools arose to help people do this, but their complexity makes their utility questionable.

1.1 The Sutherland Challenge

Our Enterprise Scrum approach emerged from trying to find a simple solution to "The Sutherland Challenge."

Jeff Sutherland, one of the developers of Scrum, works for OpenView Venture Partners, a venture capital group that works closely with portfolio startups to maximize productivity and profitability. Sutherland teaches OpenView's portfolio companies to use Scrum, and advises its general partners to diagnose problems in portfolio companies.

He recommends that general partners ask CEOs these questions:

1. What is your current velocity?
2. What are the blockers impeding your progress, and what are you doing about them?

If the CEO cannot answer the first question, she cannot balance features against effort. She cannot

properly coordinate non-engineering activities with forecasted release dates.

If the CEO cannot answer the second question, there isn't sufficient communication between engineering and top management to remove impediments quickly. Engineers could get stuck for a long time with no resolution.

1.2 Our Response

The Sutherland Challenge was designed for startup companies, but I asserted our CEO should be able to answer the same questions for our multi-product offerings, albeit at a chunkier level. What would this mean?

Initially, I attempted standard bottom-up approaches, with the velocity of the entire 250-member engineering department expressed as "story points per month." It would not be simple. Our teams were big and small, with Sprint lengths from 2 to 4 weeks long.

To get an aggregated velocity, all Scrum teams would have to calibrate and unify story points units. It would require ScrumMasters to roll-up story points from Product Backlog Items into Epics. Our tracking tool did not allow for easy ranking of Epics. Finally, it would require that our 25 ScrumMasters be trained to use the same method. None of this seemed very agile.

My quest had a time-box: our department wanted to show that hiring additional engineers could provide greater profitability to the company at a yearly budget meeting. Our company makes budget decisions yearly, and budget planning was looming.

As a temporary solution, we had architects and team leads estimate large projects (including some products) with "Enterprise Story Points." These Enterprise Story Points were roughly equivalent to "team months." We did not use rollups, because they would take too long. Furthermore, we decided to create a first "enterprise sprint" to coincide with our company's normal quarterly planning periods.

We decided, temporarily, to run weekly stand-ups at the company level. When things stabilized, our thinking went, we would move to less frequent stand-ups.

It turned out that all of these temporary approaches remain relatively permanent. Our company was able to satisfy The Sutherland Challenge. We have a velocity of about 5000 Enterprise Story Points per quarter. We hear about blockers every week at our Enterprise Standup. Our VP of Engineering can tell you all the blockers and what we are doing to remove them.

2 Enterprise Scrum

Enterprise Scrum proceeds from the notion that scaling Scrum to a multi-product enterprise should be dirt-simple. It requires thinking about every attribute of Scrum—Sprint length, estimation units, backlog item size, feedback-loop frequency, the resources to be assigned, the planning process, the ranking system—and scaling them to meet the planning requirements of the whole company.

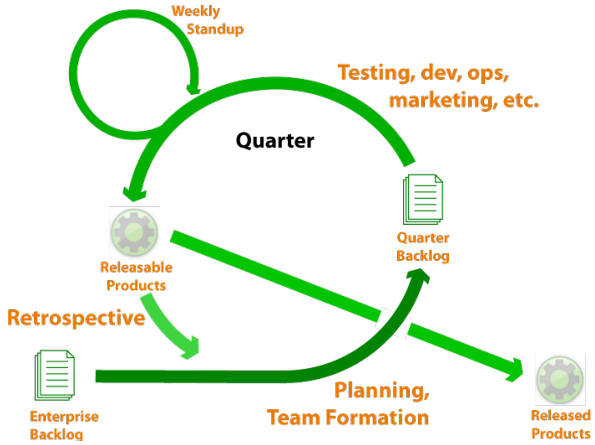

Figure 1. Enterprise Scrum

Figure 1 shows the feedback loops of Enterprise Scrum. A Quarter is equivalent to a Scrum Sprint: it includes planning, work, production of feasibly releasable products, and a retrospective. A weekly standup provides a forum for development teams to discuss progress, short-term plans and difficulties, and find collaborative solutions to impediments.

2.1 Roles

There is perhaps an ideal set of roles in Enterprise Scrum, but at this level diplomacy, skill set and authority will make perfect role fulfillment unlikely. In our company, the VP Product Management coordinates and reconciles value research and prioritization, and, in a sense, is essentially the Enterprise Product Owner. The VP Engineering is responsible for effort, and manages engineering, including hiring, firing, budgeting and day-to-day operations. The Enterprise ScrumMaster enforces the Enterprise Scrum process. Several Product Managers each hold responsibility for value research and prioritization within one or more

projects. Several Project Managers each hold responsibility for enforcing the Scrum process within one or more projects. Tech Leads from different development groups and operations can thoughtfully discuss technical limitations, opportunities and blockers. These people comprise the Enterprise Scrum Team.

Normal Scrum divides Pigs from Chickens, with Chickens provided only limited speaking rights at meetings. In Enterprise Scrum, we consider all members of the engineering, operations and product management group as Pigs, with the ability to raise issues and ask questions in meetings. Outside groups, such as Finance and Marketing, are considered Chickens.

2.2 Enterprise Backlog Items

Enterprise Backlog Items (EBIs) are the work product of the Enterprise Scrum Team. Each EBI is a project that must be at least one Scrum team-month in effort (a Scrum team consists of the standard 5-9 people), and it must be feasible to finish an EBI in three months. In practice, this means an EBI must be between 1 and 9 team-months of effort (e.g. may require 3 teams for 3 months).

2.3 Product Managers

A board of Product Managers, including top-level Product Owners, a Chief Architect, and the Director of Business Intelligence, prioritize all engineering work at our company. Our top-level Product Owners each represent one or more product lines. Our Chief Architect is responsible for infrastructure components, scalability, fault-tolerance, modularity and maintainability. Our Director of Business Intelligence manages financial reporting systems.

2.4 Quarterly Planning

The Product Managers meet periodically to identify high-priority Enterprise Backlog Items to work on in the coming three months. EBIs currently in-progress are allowed to finish, but EBIs that have not yet started can be reprioritized or canceled (this is more "lean" than "Scrum," for those familiar with the distinction). New EBIs can be introduced and prioritized at a high rank, but never interrupt an in-progress EBI.

Once it identifies a high-priority candidate project and expresses it as an EBI, the Enterprise Product Owners obtain estimated effort by one of two ways: either the team(s) likely to work on the EBI uses Planning Poker [6] to determine the number of estimated team months, or they consult with the Chief Architect (who often delegates this) to obtain an estimate. Enterprise Product Owners, other than the Chief Architect, cannot use their own effort estimates.

The Enterprise Product Owners then try to rank-order EBIs by consensus, but conflicts can arise that require arbitration by the VP Product Management, VP Engineering and CEO. Our engineering department provides velocity targets for coming quarters (now established using historic data) that determine how much work we could accept. We call this velocity limit "the line", because EBIs above the line will likely get worked on by engineering and EBIs below the line likely won't. Of course, during the planning process, value and ranking arguments focus on EBIs that are "close to the line."

Ultimately, these priorities make their way to an official 4-hour Quarter Planning Meeting, where a few conflicts or questions remain.

Because rank-ordering EBIs seems to require apples-to-oranges comparisons, we have started to rely on net present value (NPV) estimates. This allows us to compare infrastructure projects, which often make more efficient use of hardware or employees, with product features.

Whether an EBI provides cost-savings or additional revenue, NPV provides a neutral measure to make comparisons. The ratio of NPV to estimated effort is roughly profit margin; the rank-order of EBIs should be approximately in descending order of the profit margin.

The result of Quarter Planning is a rank-ordered subset of EBIs that could be performed in a quarter. The ranking is then submitted to a corporate planning board, which includes the CEO and all VPs, for approval. They can (and do) change the ranking of these EBIs. They are aware that a low-ranked item might not be worked on, and that other service priorities (operations, marketing, customer support) rely partly on the rank ordering.

2.5 Team Formation

Immediately following Quarterly Planning, we host a Self-Organization Meeting attended by a representative of every existing Scrum team. Such representatives are selected by the team, not product owners or ScrumMasters, and are empowered to

reallocate the people they represent to different teams.

Representatives come armed with sticky-notes containing the names of people they represent. In the meeting room, each EBI is written on a large paper sheet. Representatives then mill around, discussing options and assigning their people to different EBIs by placing sticky-notes on the appropriate sheet. We ask that representatives focus on making assignments to the top-priority EBIs first.

In early parts of the meeting, the special skills of some people make assignments obvious and rapid. In later parts of the meeting, trade-offs get discussed and difficult decisions are made.

Some EBIs are shorter than a quarter. In these cases, representatives are asked to try to assemble cross-functional teams that can move from one EBI to another with minimum membership changes. They are asked to make tentative transition assignments, but Scrum teams are not truly committed to an EBI until they start working on it.

In the Self-Organization Meeting, representatives may assert that an estimate is unrealistic for the people who would be assigned the work. They may also discover that optimal assignment makes one or more EBIs impossible to staff or complete in the quarter. If this occurs with a low-ranked EBI "near the line," it causes few problems (controversy likely accompanied this project already). But if it occurs with a high-ranking EBI, it indicates that the EBI wasn't properly estimated. The easiest fix for this problem is an ad hoc estimation, done on the spot.

The result of the Self Organization Meeting is a prioritized commitment from the Engineering Department and a tentative team assignment. We call this the Quarter Backlog. The Engineering Department, as a whole, is saying, "We believe we can do this in the next quarter, and we are going to focus first on the highest priority items in the list."

2.6 Work

Each team then proceeds to work on its top-priority EBI. We do not split teams between EBIs, but we may put several teams on the top-most items

The Enterprise Scrum approach allows for independent self-organization at the project (e.g., "Enterprise Backlog Item") level. Theoretically, teams working on an EBI need not use Scrum. (Today, all but one EBI project currently uses Scrum.)

Individual teams can choose the sprint length they prefer. Larger projects can choose the Scrum-of-Scrums form they wish: some have an integration team, some don't; some have a coordination council, some don't.

We are aware that some Scrum scaling methods recommend synchronizing sprint-length throughout engineering, but we likely never will. We don't have enough meeting rooms to accommodate 25 teams running sprint review, retrospective, planning simultaneously. Our use of Enterprise Scrum makes synchronized Sprints largely unnecessary.

Engineering support services, such as operations, user-experience testing, user-interface design, marketing work and customer support, are now starting to use the rank ordering of EBIs to guide their own priorities.

For example, if the Operations Department receives requests to deploy several products in a week, by default it deploys the highest-ranked EBI first. This ensures that the projects with the greatest profitability get the most love. However, in consultation with the VP of Engineering, we can change these priorities.

2.7 Enterprise Scrum Meeting

The Enterprise Scrum Team meets every week, in an Enterprise Scrum Meeting. It has two primary agenda items: the stand-up and the solver session.

Time	Item
9:00am	Pick note-taker and bailiff
9:02am	Standup. Late fines enforced
9:27am	Assemble Solver agenda
9:30am	Solver session (optional)
9:59am	Meeting ends

Table 1. Enterprise Scrum Agenda

We ask a meeting attendee to take notes, and another to serve as Bailiff. The Bailiff watches for late attendees and assesses fines. This avoids disrupting the Enterprise ScrumMaster (and therefore the meeting).

The stand-up portion (I admit it: we don't all stand up) is 30 minutes long, and team members are required to attend. We drive this part from the Quarter Backlog, starting at the top-ranked EBI and working down. For each EBI, we ask the Product

Manager: What did you work on last week? What will you work on this week? What blockers impede your productivity? People can ask questions, but long-discussions are deferred for the second half of the meeting.

When an EBI report seems too hollow, such as "We worked, we are working, no blockers," the Enterprise ScrumMaster or VPs dig deeper. (It's rare that everything is running perfectly.) We frequently ask a Project ScrumMaster or Tech Lead to confirm a report, or "provide more color."

The second half of the meeting is the Solver Session, where we address blockers or critical issues. It is optional; only those affected or who can help typically remain.

2.8 Meeting Communication

Our company is physically dispersed. We use GoToMeeting to communicate with remote participants.

We ask a meeting attendee to take notes. These notes are emailed broadly to anyone who wants to subscribe; by doing this, we communicate company priorities widely and frequently.

2.9 No Quarterly Demo

At present, we do not run a quarterly review/demo meeting. Several EBIs are completed per quarter, and many are released publicly. We are uncertain whether a demo meeting would be valuable, or how it would be structured.

2.10 Quarterly Retrospective

Unlike normal Scrum, we host our Enterprise Retrospective *after* Self-Organization. The greatest frustrations occur during Quarter Planning and Self-Organization, and there is no Quarterly Demo. Therefore, it was felt that we should diagnose problems and propose solutions immediately after the main action.

3 Challenges

Our Enterprise Scrum model has all the feedback loops of normal Scrum writ large. Those feedback loops expose problems and provide a mechanism to solve problems as they appear.

I like to say, "In Scrum, the complaining never stops … and that's a good thing."

3.1 Flow Leveling for Limited Resources

At the enterprise level, problems emerge that require classic "level the flow" Kanban solutions:

For example, quarterly sprints suggest that several releases might occur at the end of the quarter, but our operations group cannot manage lots of simultaneous product releases. Running normal Scrum at the project level made this even worse: some projects could produce a feasibly releasable product in 4 weeks or less, and Product Managers wanted to release those features to users. We therefore try to include a variety of short and long projects in a quarter, scheduling them so operations sees more continuous flow.

Other services, such as user-experience testing, branding, customer support and user-interface design, are required to deliver value to end-customers. These groups can experience fluctuating demand and must therefore prioritize their work.

Our first cut at flow-leveling was to use the Enterprise Backlog as a default priority ranking for services. For example, if two projects needed deployment services from Operations, we deployed the higher ranked EBI first. While higher ranked EBIs were demanding services from Operations, lower ranked EBIs could not get services.

This worked surprisingly well. In most cases, the highest ranked EBIs were the right projects to receive service.

Because Operations was still overburdened, we started to wonder whether we could manage its capacity using Operations velocity. This led to assessing "Ops Story Points" and attempting to level Ops demand.

The jury is out on whether Ops capacity management actually levels the flow of work to Ops.

However, a happy side effect of estimating Ops Story Points for EBIs emerged: Developers and product managers began to understand that easy-to-deploy systems could increase their velocity "to the happy end-user," the ultimate done-criteria. Several groups started automating RPM creation, implementing hot-deployment strategies and providing more complete deployment documents to drive their Ops Story Points down.

3.2 Fungible Teams

Normal Scrum favors fungible people, who write software, test code, design schemas, or create release artifacts as needs arise. The team wins or loses as a group, and so its members should pitch-in to help regardless of the task.

Enterprise Scrum favors fungible teams. Teams gain efficiency as they work together over a long time. Instead of breaking up these teams in a Self-Organization Meeting, we should find ways to move whole teams to new EBIs. We have found this to be challenging, but have preserved some teams in radical reassignments, and we continue to work on ways to do this. We believe preserving teams improves morale, team communication and productivity.

3.3 Agile Programming Environments

IDE, database, source control and dependency management setup can be extremely complicated. This is essentially "technical debt," which also appears in normal Scrum, but it is much more apparent in Enterprise Scrum when teams move between different products.

We use Maven, local databases, and other tools to reduce the overhead of switching between projects, but it is an ongoing effort to make the mechanics of switching projects efficient. This problem is not as prominent with single product engineering groups.

3.4 Net Present Value

We are finding we can feasibly consider "net present value" calculations at the coarse granularity of the Enterprise Backlog.

For example, a server-team added a 6 team-month project to the Enterprise Backlog and requested prioritization. But the description was indecipherable to everyone outside the server-team: it was a refactoring project to reduce hardware requirements. In an early quarter planning meeting, it dropped off the quarter backlog. The group realized it had to articulate the project's value better. They computed the cost-savings that would result from the project, and discovered a multi-million dollar Net Present Value. In the next planning meeting, it was added to the quarter.

We have discussed Net Present Value with others in the Scrum community, and get a mixed reaction. Some argue that it stifles creative exploration. Others state that all decisions should be made with NPV as a basis. Some are exploring methods by which experimentation can gain a "Net Present Value" for the information it reveals, such as through Real Options [8]. Our current approach is to encourage calculating and discussing NPV when it can be responsibly determined, but not use NPV/effort as a required ranking mechanism.

We suspect we will deepen our use of NPV as we become more comfortable assessing the value of market and technology experimentation. NPV is widely used in other expensive and exploratory fields, such as in oil exploration.

3.5 Corporate Governance

Perhaps the biggest challenge of all is scaling Scrum to the operation of an entire company. Enterprise Scrum is encroaching into this area, certainly, in part because it promotes the use of Scrum in other large, struggling creative departments, such as marketing. But we are unsure whether it would work in less creative or low-leverage departments.

Scrum works well with other egalitarian management approaches, such as Market-Based Management [10]. We are exploring these notions further.

4 Conclusion

Our Enterprise Scrum process estimates projects in "team months," runs quarterly Sprints, assigns one or more full teams to each project, meets in weekly stand-ups, etc. At the project level and below, we continue to use normal 2-to-4 week Scrum. Limited ops and marketing capacity motivates "flow leveling" in planning. Enterprise Scrum promotes cross-product communication throughout the company, and allows us to make more thoughtful tradeoffs.

At this writing (June 2009) we are finishing the second Quarter of Enterprise Scrum. We required diplomacy and CEO buy-in to start it. Organizational resistance may be the main barrier to other organizations trying it, because top executives must be willing to give it a serious try.

Our current results are very good: We are establishing a culture of transparency that seems now to pervade the company; execs now understand where all the engineering effort is deployed, product managers are doing a better job of determining market value, we are able to preserve teams longer-term, service organizations (Operations, Security

team, IT, etc.) use the quarterly backlog to roughly prioritize requests and feel empowered to say "No" when overwhelmed. Finally, we are evangelizing this approach beyond engineering to gain understanding about where it best applies.

Everyone knows what projects are succeeding, and which are having trouble. When a problem could impede a release, it becomes rapidly visible and can trigger immediate executive action.

We are producing more frequent releases that better target user needs. Our engineers are becoming more flexible and better aligned with company success. Enterprise Scrum works.

5 Appendix: Fractal Scrum

[This section describes the "fractal thinking" that led to creation of Enterprise Scrum, but is optional reading.]

We created Enterprise Scrum by treating normal Scrum as the seed of a fractal. A single-team Scrum can be viewed as a fractal iteration of multi-team, then multi-project Scrum, as described here.

I assert that even simple Scrum exhibits *fractal self-similarity*, a property you can also use to scale it. Self-similarity is a property of many edge-of-chaos systems [4]. The pioneers of Scrum recognized the relationship of software engineering to chaos theory [11].

To create Enterprise Scrum, I analyzed normal Scrum as a fractal, focused on Scrum's fundamental properties, and then scaled them up.

5.1 Fractals in Scrum

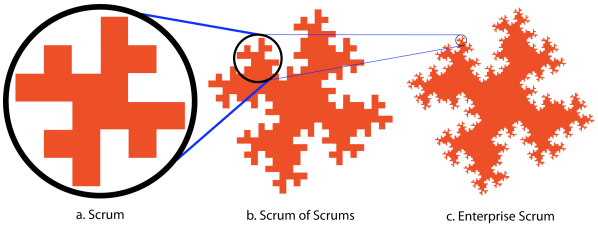

Figure 2. Fractal Self-Similarity

Fractals are geometric shapes that exhibit self-similarity: some properties remain the same regardless of scale. For example, Figure 1c is a fractal. If you look at the gross structure in Figures 1a, 1b and 1c, the images are similar (squint if you are having trouble). Going from Figure 1a to 1c, the metrics, complexity and fine detail become greater, but the general outline remains the same.

Normal Scrum exhibits self-similarity, so I'll use it as an example.

Scrum has two feedback loops at different time scales. Every sprint, the whole team discusses what they did (Sprint Review), what they will do (Sprint Planning) and what blocks progress (Sprint Retrospective). Every day, the standup meeting requires team members to discuss what they did, what they will do, and what blocks progress. The rough purpose of each feedback loop is the same: share information, get feedback, adjust goals, and escalate impediments. Everything but the time-scale is roughly the same.

Scrum has two estimation scales. Whole teams groom and estimate effort for Product Backlog Items, using *story points*, a unit often roughly the size of estimated programmer days. The whole team is considered responsible for a Product Backlog Item. Individual team members usually generate and estimate effort for Tasks, using *estimated programmer hours*. That same person is responsible for completing the Task.

The purposes of estimation is the same: allow the team and individuals to prioritize activities on both relative value and relative effort, limit effort to respect the capacity of the team and individuals, allow the team and individuals to focus on a small number of activities, and gain greater understanding of the actual capacity of the team and individuals. They differ in *scale*.

Typical Scrum teams establish done criteria at different scales, though people sometimes don't realize it.

For example, a *Task Done Criteria* might include

1. unit tests were written and succeed,
2. a local build with all tests still succeeds,
3. the code was reviewed by another team member,and
4. the work was checked into the source control system.

A *Product Backlog Item Done Criteria* might include

1. automated feature test was written and succeeded,

2. continuous build system on all platforms still succeeds,
3. team agrees it is good quality and marks it Done, and
4. deployment package was produced and checked into the repository.

Finally, a *Sprint Done Criteria* might include

1. upgrade and revert processes were performed on a clean integration system,
2. operations staff did a successful dry run install,
3. Product Owner reviewed the Sprint Backlog Items, marked them Accepted or rolled them forward, and closed the Sprint, and
4. product package was queued up in the Operations Backlog.

Let's review the Done Criteria parallelism. The 1s confirm the work was done. The 2s ensure other components in the ecosystem won't fail. The 3s cause an external party to validate the work. The 4s publish the work. They differ in *scale*. Now, when you look at the subcaptions of Figure 2, they might make more sense.

5.2 Scaling Scrum to the Enterprise

We've analyzed what exists, normal Scrum. Let's create something new, by scaling Scrum up to meet the needs of a whole engineering department. Enterprise Scrum looks like regular Scrum, with everything writ large:

1. Story point size is *estimated team months*, estimates come from architects,
2. *Enterprise Backlog Items* (EBIs) can be no smaller than 1 team-month of work,
3. Sprint size is three months, i.e., a Quarter,
4. Standup meeting frequency is every week, and
5. Teams (not people) sign up for Enterprise Backlog Items.

Sound simple? Once running, Enterprise Scrum *is* mechanically simple, but initiating it is not. Your executive staff will take time to adopt Enterprise Scrum. Enterprise Scrum increases executive accountability, always a little scary. Enterprise Scrum exposes executive decisions so publicly that they can be second-guessed, and this threatens control. Without Enterprise Scrum, other departments could blame delays on the engineering department. I don't think you can adopt Enterprise Scrum unless your CEO and relevant Vice Presidents support you.

In other words, in this fractal the diplomatic challenges scale up with the rest. The same organizational impediments you faced with Scrum adoption will emerge in Enterprise Scrum, writ large. But the rewards scale too; if your company can do it, you win big in productivity.

6 References

1. Ken Schwaber, The Enterprise and Scrum, ISBN 978-0735623378, Microsoft Press 2007.
2. Craig Larman and Bas Vodde, Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum, ISBN 978-0321480965, Addison-Wesley 2008.
3. Dean Leffingwell, Scaling Software Agility: Best Practices for Large Enterprises, ISBN 978-0321458193, Addison-Wesley 2007.
4. Heinz-Otto Peitgen, Hartmut Jürgens, Dietmar Saupe, Chaos and fractals.
5. Dan R. Greening, Scrum Self-Similarity, http://scrumerati.com/2009/05/scrum-fractals.html
6. Mike Cohn, Agile Estimating and Planning, ISBN 978-0131479418, Prentice-Hall 2005.
7. Dan R. Greening, Marketing Scrum Experimentation, http://scrumerati.com/2009/05/marketing-scrum.html
8. Chris Matts, Real Options and Agile Software Delivery, http://bit.ly/eA1vl, Agile 2006.
9. Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, ISBN 978-0130676344, Prentice-Hall, 2001.
10. Charles G. Koch, The Science of Success, Wiley, 2007.
11. Ken Schaber's web site is called http://controlchaos.com.