

# Enterprise Scrum: Scaling Scrum to the Executive Level

Daniel R. Greening  
Citrix Online  
[dan@greening.org](mailto:dan@greening.org)

## Abstract

*Our company manages 25 software engineering 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 projects, meet in weekly stand-ups. We start, postpone or cancel whole projects.*

*Within individual projects, we still use 1-4 week Sprints and all the trappings of the classic Scrum process, including, in some cases, Scrum-of-Scrums.*

*New challenges arise: Shared resource constraints suggest Kanban methods. Net Present Value can justify prioritization, but creates controversy. Moving teams between projects requires rapid programming environment setup. The process forces executives to justify decisions. We want simple improvement metrics, but they seem elusive.*

## 1 Introduction

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

For single software 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 larger 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. 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 what I call “The Sutherland Challenge.”

I met Jeff Sutherland, one of two inventors of Scrum, on a bus in Toronto. He works for OpenView Venture Partners, a venture capital group that works closely with portfolio startups to maximize productivity. Sutherland teaches OpenView's portfolio companies to use Scrum, and advises its general partners to diagnose and repair productivity problems in portfolio companies.

Sutherland told me 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 coordinate non-engineering activities with release forecasts. 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 startups, but I told our President, who leads a 900-employee division of Citrix, that he should be able to answer the same questions for our multi-product division, albeit at a chunkier level. He then asked my boss, the VP of Engineering, for this information, and he, in turn, asked me for the data, unsure whether to be amused or annoyed. What would this mean?

Initially, we attempted standard bottom-up approaches, with the velocity of the entire 200-member engineering department expressed as "Story Points per month." It would not be simple. Our teams were big and small, with Sprint lengths from 1 to 4 weeks long.

To get an aggregated velocity, all Scrum teams would have to calibrate and unify story point 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 our 25 ScrumMasters be trained to use the same method. None of this seemed 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 ESPs were roughly equivalent to "estimated team months times 100." We did not use rollups, because they would take too long. Furthermore, we decided to create a first quarter-long "enterprise sprint" to coincide with our company's normal planning period. We decided to run weekly stand-ups at the company level. When things stabilized, our thinking went, we would meet less frequently.

It turned out that all of these temporary approaches have become 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. In fact, you can look at the meeting notes we email broadly, and the blockers are highlighted.

## 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. It requires rejecting process complexity.

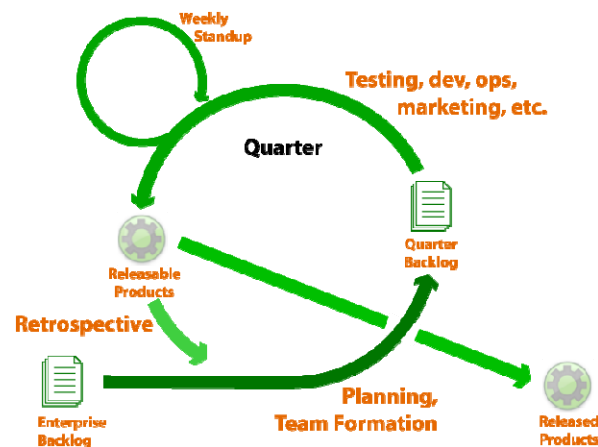


Figure 1. Enterprise Scrum

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 Product Managers, ScrumMasters and Team Leads 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 Director of Product Management coordinates and reconciles value research and prioritization, and is essentially the Enterprise Product Owner. The VP of 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 ScrumMasters each hold responsibility for enforcing the Scrum process within one or more projects. Tech Leads from different development groups, the User Experience Manager and the Operations Director can thoughtfully discuss technical limitations, opportunities and blockers. These people comprise the Enterprise Scrum Team.

## 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 Board*

A Product Board, including our Product Managers, Chief Architect, and Director of Business Intelligence, prioritize all engineering work at our company. The Product Managers each represent one or more product lines (sometimes our Product Managers act as Product Owners for individual Scrum teams, sometimes they delegate Product Owner responsibility to others). The Chief Architect is responsible for infrastructure components, scalability, fault-tolerance, modularity and maintainability. The Director of Business Intelligence manages financial reporting and sales tracking.

## 2.4 *Quarterly Planning*

Our planning process has shifted quarter-to-quarter. We started with uninterrupted quarters, and several stakeholders rebelled. We then attempted a full “lean” approach (no Sprint time-box) with three-month forward looking plans adjusted monthly, but this was an overly demanding non-starter.

We tried various estimation techniques. When our process allowed people to “game the system” to staff their projects with the maximum engineers at the expense of overall department productivity, they typically did so.

Our current Quarterly Planning process involves four iterative steps and starts 6 weeks before the start of a quarter. Each iteration introduces greater care in

resource allocation: first Product Board members estimate, then Team Leads estimate, then Teams estimate, then Engineering assigns teams. By iterating, we postpone the high cost of full team estimation and resource assignment until priorities are well considered and established.

In step 1, Product Board members propose projects for the coming quarter, as Enterprise Backlog Items, with clear acceptance criteria that (hopefully) have independent business value. The Product Board members themselves estimate effort in Enterprise Story Points. Enterprise Backlog Items can vary between 50 and 900 Enterprise Story Points. A team of 3-4 developers (including QA and user-experience staff) is 50 ESPs per month. A team of 5-7 developers is 100 ESPs per month. We ban over-size Scrum teams. We count each “part-time” specialist as a full developer, because the dependency and blocker problems they introduce typically slow the team down.

This ESP metric slightly favors smaller teams, which are known to be more productive per engineer, and improves our ability to staff thoughtfully. When we didn’t distinguish big teams from small, Product Board members sought large teams almost exclusively. Large teams tend to be more productive *per team* but less productive *per engineer*.

To finish step 1, Product Board members prioritize based on their own effort estimates.

In step 2 of Quarterly Planning, we first save Product Board member estimates and then erase them from public view, leaving only the EBIs and their rank ordering. For each EBI, we find Team Leads in the organization that can responsibly estimate it. We remind every Team Lead to estimate the size of the entire EBI based solely on the acceptance criteria, even if it would extend beyond a quarter with teams they think will work on it. We caution Product Board members not to reveal their step 1 estimates, to avoid biasing the result.

We then meet with Product Board members and compare Team Lead estimates to their original estimates. Where Product Board member and Team Lead estimates differ, the responsibility falls on the Product Board member to reduce the EBI acceptance tests, or accept the Team Lead estimate. Product Board members can reprioritize in this meeting.

We meet with executives and show them our current proposed Quarter Backlog, using past velocity as a guide to the new quarter’s capacity. We

typically draw a line at that capacity, but remind execs that items close to the line are unlikely to be completed, even if they are above the line. This is their first opportunity to argue for reprioritizing EBIs they think provide more long-term value.

In stage 3, we first save Team Lead estimates and erase them from public view. For each EBI, we identify teams likely to do the work and ask the teams to estimate Enterprise Story Points in multiples of 50 using Planning Poker [6]. We make a special effort to avoid biasing the teams' estimates. We try to shield them from Product Board member and Team Lead estimates. We also ask teams to free themselves of any preconception that effort should fit into a single quarter.

We meet again with the Product Board. We compare Team Lead estimates with team estimates. Product Board members then have a last chance to reprioritize the backlog. If there are significant changes, we share this with executives and obtain their approval or advice.

In stage 4, engineering attempts to staff the Enterprise Backlog, starting at the first EBI. At the top of the list, there are usually reasonable teams to staff EBIs. As we proceed through the list, teams become less and less ideally suited for the EBIs remaining. Engineering can choose to create teams with reassigned staff members to suit an EBI, or can decide not to staff an EBI due to lack of appropriate people.

**Self Organization** We have tried several staffing processes. In Q2, we staffed using a "self organization" meeting. This process has not ultimately been retained, but discussion of it may provoke thought.

A representative of every existing Scrum team attended the self-organization meeting. Such representatives were team leads, not Product Owners or ScrumMasters, and were empowered to reallocate the people they represent to different teams.

Representatives came armed with sticky-notes containing the names of people they represented. In the meeting room, each EBI was written on a large paper sheet. Representatives then milled around, discussing options and assigning people to an EBI by placing sticky-notes on the appropriate sheet. We asked that representatives focus on making assignments to the top-priority EBIs first.

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

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

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

**Non-Self Organization** In Q3, we stopped using a formal Self Organization meeting, because the overhead was too great and the outcome was mostly predictable. Instead, we rely on ScrumMasters and Managers working independently to identify appropriate teams and gain mutual approval. This is a work-in-progress, with the ultimate arbiter being the VP of Engineering.

Self organized or not, stage 4 results in a prioritized commitment from the Engineering Department and a 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." All participants 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 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 at least partial Scrum. Most use pure Scrum.)

Individual teams can choose the sprint length they prefer. Larger projects can choose the Scrum-of-Scrums form if 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 and planning. Our use of Enterprise Scrum makes synchronized Sprints largely unnecessary.

Engineering support services, such as Operations, User Experience, Marketing and Customer Support, are now starting to use the rank ordering of EBIs to guide their own priorities.

For example, if Operations 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 rapid deployment. However, in consultation with the VP of Engineering, we can change these priorities.

**2.6 Enterprise Scrum Meeting**

The Enterprise Scrum Team meets every week, in an Enterprise Scrum Meeting. It has two primary agenda items: the Standup and the Solver Session.

Time	Item
9:00am	Pick note-taker and bailiff
9:02am	Standup. Late fines enforced
9:27am	Create Solver Session 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 Standup portion (I admit it: we don't all stand up) is 30 minutes long, and Enterprise Scrum 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 its Product Board member: What did your teams

work on last week? What will your teams work on this week? What blockers impede your productivity? People can ask questions, but long-discussions are deferred for the Solver Session.

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.

**2.7 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.

**2.8 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.

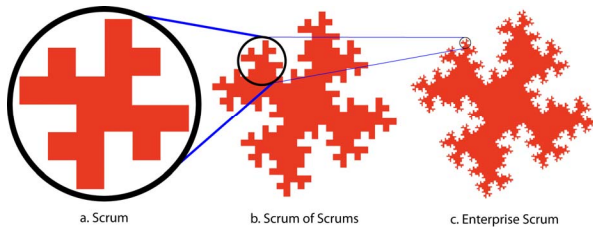
**2.9 Quarterly Retrospective**

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

**3 A Fractal View of Scrum**

I assert that normal Scrum exhibits *fractal self-similarity*, a property I recognized and used to scale normal Scrum to Enterprise Scrum. Self-similarity is a property of many edge-of-chaos systems [4]. Scrum pioneers applied chaos theory to software engineering problems [11]. It isn't surprising that Scrum has this property.

### 3.1 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 (ignore the subcaptions for now). 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 completely separated 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*. The estimating person is typically 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. There are many self-similarities in Scrum.

### 3.2 Enterprise Scrum as a Fractal

We've analyzed what existed before, 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 and aggregated teams,
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? Like normal Scrum, Enterprise Scrum is mechanically simple, but initiating it is not. Executives will take time to adopt Enterprise Scrum. Enterprise Scrum increases their accountability, always a little scary. Enterprise Scrum exposes their 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 President and relevant Vice Presidents support you. We had executive support.

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; the value of each EBI in our Enterprise Scrum typically exceeds US\$1M.

## 4 Challenges

As with all continuous improvement processes, Enterprise Scrum exposes problems we didn't realize we had. Here I outline several "works in progress" we are gradually addressing.

### 4.1 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 can now produce a feasibly releasable product in 4 weeks or less, and Product Managers want to release those features to users.

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 Operations 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.

### 4.2 Net Present Value Estimation

In each stage of planning, prioritization depends on both effort and value estimation. The normal Scrum process focuses heavily on effort estimation, but largely ignores value estimation. Scrum makes value the sole purview of the Product Owner; the Product Owner articulates value by prioritizing the Backlog after a Scrum team estimates effort. The Product Owner's difficult value estimation work, in Scrum, is an "exercise left to the reader."

In contrast, Scrum provides a sophisticated process to get bias-free *actual* effort estimates, though Scrum training and rituals hide this in team velocity (actually, we depend on this hiding to remove psychological bias). Burndown charts help construct a linear mapping from estimated effort (in Story Points) to actual effort (in team-time), while handily avoiding the huge psychological and training overhead of trying to correct the difference (attempted by the Personal Software Process promoted by SEI) [12]. Scrum seems to work relatively better than previous software engineering processes to estimate effort. Engineers seem to more readily adopt it.

Enterprise Story Points, which are 100 points per estimated team month, can be converted to estimated cost straightforwardly. A single team-month is about

US\$100,000 in loaded staff cost. Until inflation erodes this convenience away, to get the estimated cost of an EBI, we simply multiply its ESPs by US\$1000.

But *effort* and its cost are only half of the prioritization problem; the other half is *value*. Normal Scrum demands only that the Product Owner rank-order groomed Product Backlog Items to articulate value, and this is reasonable. It is impractical to try to assess the monetary value of a Scrum team's Product Backlog Items [13]. But it is almost a moral obligation with projects as large as an EBI, which costs a minimum of \$50,000 and as much as \$900,000.

Rank-ordering EBIs seems to require apples-to-oranges comparisons, but we have started to rely on Net Present Value (NPV) estimates as a common metric. This allows us to compare infrastructure projects, which often make more efficient use of hardware or employees, against product features.

Our NPV calculations assume a 10% annualized discount to cash, with a maximum forward-looking window of 3 years. Net Present Value can compare a cost-savings effort to one that generates revenue, or compare something that generates one big pile of cash (such as selling off a division) to something that generates revenue over time (such as keeping that same division and receiving its revenues yearly).

What does "annualized discount to cash" mean? It means that money earned today is worth more than money earned next year, based on an interest rate. Furthermore, costs incurred today are more expensive than costs paid next year, based on the same interest rate. For example, if finishing EBI A would generate \$1M exactly 1 year from now, it is only worth \$909,091 in Net Present Value. This is because \$909,091 placed in a bank earning 10% annually will be worth \$1,000,000 a year from now.

What does "maximum forward-looking window of 3 years" mean? We assert software businesses have many near-term risks, therefore we can't see further than 3 years ahead, and don't count earnings beyond that. For example, if finishing EBI B would earn \$1M on one day per year including today, its Net Present Value is  $\$2,735,537 = \$1,000,000$  (this year) +  $\$909,091$  (next year) +  $\$826,446$  (two years from now).

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 that profit margin.

For example, in our first planning period for Enterprise Scrum, a server-team added a 600 ESP project to our Enterprise Backlog and requested prioritization. But the description was indecipherable to everyone outside the team: it was a refactoring project to reduce hardware requirements. In an early Product Board meeting, members dropped it off the Quarter Backlog. The server 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 Product Board meeting, its priority was raised back into 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 Net Present Value for the information it reveals, such as through Real Options [8]; we are open to this, but have not yet found a practical way to do it. 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.

NPV generates controversy in our company for these reasons:

1. Not everyone is mathematically inclined. Our answer is to suggest they find colleagues to help.
2. Conservative interpretations of NPV generate ridiculously low numbers; our answer is to interpret NPV to include "the whole profit picture" (i.e. ongoing revenues and costs, including cost-reductions from partnering, cross-product leverage, sales-training costs, marketing channel expansion costs, etc.).
3. NPV estimators worry that others will later compare their estimates to actual results, and "throw them under the bus" if they were wrong. Engineering departments using waterfall

processes suffered this problem, on the effort side, when we would forecast release dates far into the future. We partly mitigate the value problem by keeping project-size small: 3 months or less, and trying to make the project releasable to customers. We actively discourage revisiting old NPV estimates to judge the quality of a product management decision. In short, NPV is useful as a decision-making tool; for this purpose, it is not an auditing tool.

4. Some projects have value, but two people making reasonable assumptions could obtain wildly different NPV. For example, security projects can have huge dollar risks with very low likelihood. An option in this case is to explicitly state assumptions and provide a sample calculation (in the EBI description), thus helping people assess the value themselves. In these cases, we sometimes leave NPV unstated.

We continue to promote and respect use of NPV to assess value in all Enterprise Backlog Items until someone nominates a reasonable alternative. That said, if all we get is a prioritized list of EBIs from Product Management and no NPV, Engineering has enough information to deploy resources.

### 4.3 *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 to create theoretically optimal teams for a new quarter, 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.

Because we use Scrum within teams, and because we have made radical whole-team reassignments, we can now estimate the learning curve cost of new work.

### 4.4 *Agile Programming Environments*

Development environment, database, source control and dependency management setup for an unfamiliar project 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.

### 4.5 *Measuring Improvement*

We define company productivity as NPV/effort (for a familiar analogy, national productivity is measured as Gross Domestic Product, i.e., dollars per person per year). However, we are often uncertain how to measure productivity in a relatively short time frame, so we can compare and select from alternative approaches.

As we have deployed and improved our use of both normal Scrum and Enterprise Scrum, we realize that *how* we do things can make large productivity differences. Sprint length, office layout, Scrum training, how EBIs articulate acceptance criteria, encouraging NPV analysis, etc. all make a difference. But measuring improvement is not easy.

Scrum productivity metrics in peer-reviewed articles often use value surrogates for productivity’s numerator. Value surrogates can include function-points or lines of code [19].

We find these surrogates inadequate to measure Enterprise Scrum. One example illustrates the problem: in many companies, the urgency to release a new product can require “forking” the code from an older product. The company then must maintain both forks as long as the two products remain viable. To pay for a short-term productivity gain, the company pays in future productivity year-after-year. Forked code is not just theoretical: I’ve seen this problem in every multi-product company that has employed me.

Merging two code forks could increase NPV for the company, due to increased future engineering productivity, but it will likely *decrease* the number of function-points. What simple metric can show this productivity gain?

### 4.6 *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, 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.

## 5 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 1-to-4 week Sprints. 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 (September 2009) we are finishing the third Quarter of Enterprise Scrum. We required diplomacy and our President’s buy-in to start it. Organizational resistance may be the main barrier to other organizations trying it, because top executives and engineers must be willing to give it a serious try. It was disruptive to our organization, exposing many previously hidden conflicts and revealing upstream root-causes once attributed to engineering.

Our current results are 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, User Experience, Security team, IT, etc.) use the Quarter 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.

Enterprise Scrum seems to work well, but we don’t yet have clear metrics. We are producing more frequent releases that better target user needs. Our engineers are becoming more flexible and better aligned with company success. We are working to assess improvement numerically. Profitability may

ultimately be the only reasonable measure, but it takes time to emerge.

*Dan Greening is Citrix Online’s Director of Engineering Productivity and User Experience, and acts as Enterprise ScrumMaster. He largely designed the Enterprise Scrum process. He was the founder of several startups, some successful, some not. He has been Principal Investigator on three National Science Foundation SBIR grants. He holds a Ph.D. in computer science from UCLA.*

## 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/eA1v1>, 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>.
12. Personal Software Process, Wikipedia, 15 Sept 2009, [http://en.wikipedia.org/wiki/Personal\\_Software\\_Process](http://en.wikipedia.org/wiki/Personal_Software_Process).
13. Luke Hohmann, Why Prioritizing your Product Backlog for ROI Doesn’t Work, <http://www.enthiosys.com/insights-tools/prioritizeforprofit1of3/>.
14. Jeff Sutherland, Scott Downey and Bjorn Granvik, Shock Therapy: A Bootstrap for Hyperproductive Scrum, Proceedings of Agile 2009, Chicago IL (August 28, 2009).