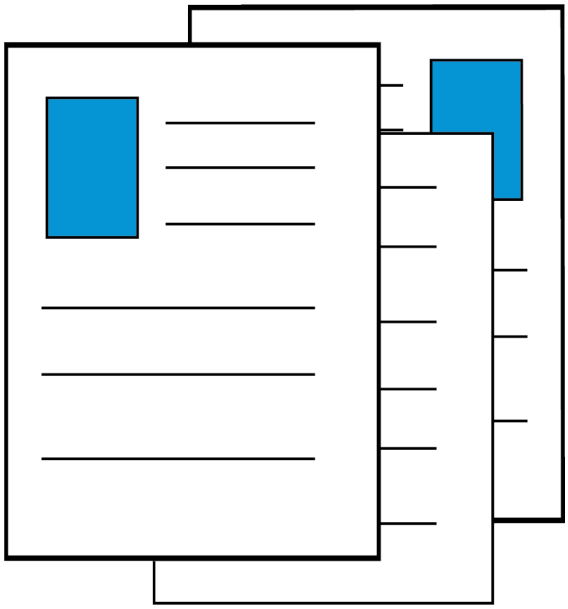
Scrum in the Enterprise: Common Issues

Are you running into these 11 issues? You are not alone!



cPRIME RESOURCE CENTER

www.cprime.com | learn@cprime.com | (877) 753-2760

Contents

1 Hybrid Projects: Scrum and Waterfall, Together

The Scrum process focuses on the tactical execution of requirements generation and implementation, for environments where scope is fluid and schedule cannot be predicted reliably. It does not attempt to define how the enterprise or business unit operates, and it will not usually be the only type of development process present in a large organization.

As a result, “Scrum projects” must interface to the rest of the enterprise in a graceful fashion, or failure is likely. The question of how these interfaces work comes up frequently. The answer is that there are two types of interfaces: Stakeholder and Project interfaces.

1.1 Stakeholder Interfaces

“Stakeholder Interfaces” are the touch points between the Scrum team that plans and executes work with a Scrum process, and external stakeholders outside the team who participate in the process.

Examples of external stakeholders include

- Customer Support personnel, who provide bug reports and feature requests from customers, and information to customers about when their requests are likely to be addressed
- Professional Services personnel, whose needs are similar to Customer Support
- Sales and Marketing personnel, who need information about planned features when soliciting new business

When an organization implements a Scrum process, it must do so in a way that involves external stakeholders productively. cPrime provides Agile consulting services to assist in developing a Scrum process that works smoothly for the Scrum team and all external stakeholders.

1.2 Project Interfaces

Large projects, or programs, may be composed of smaller pieces, each of which is managed by a particular type of process. For example, a project to build a Data Warehouse (DWH) and supply Business Intelligence (BI) capabilities might contain the following sub-projects:

1. Environment: Set up hardware and software environments
    a. Set up development environments
    b. Set up testing environment
    c. Set up staging environment
    d. Set up production environment
2. Development: Develop DWH and BI capabilities
    a. Implement user accounts
    b. Develop Report #1
    c. Develop Report #2

The project as a whole contains two major sub-projects.

The Environment project is a project in the classic sense of the word: “A temporary endeavor undertaken to create a unique product, service or result.” It has a start date, an end date, a schedule, and a fixed scope. The work is familiar, repetitive, and predictable. It is managed well by a *predictive*, or waterfall, process (often referred to as SDLC, or Systems Development Life Cycle).

The Development project is very different. It represents an ongoing development process, with no fixed concept of scope or end date, which provides increasing value over time in the form of new deliverables. (The outline above shows only the first three of these deliverables.) The work involves constant invention, is not repetitive, cannot be estimated well, is subject to frequent scope changes, and requires continuing adaptation to unpredictable circumstances. It is managed well by an *adaptive*, or agile, process such as Scrum.

1.3 How to Manage Hybrid Projects

The first step to managing a hybrid project is to recognize that it decomposes well into (say) waterfall and Scrum sub-projects, and to plan each sub-project with the appropriate process.

The second step is to identify the dependencies between these component projects. For hybrid projects, the predictive projects typically generate milestones that act as constraints for the adaptive projects. The *Constraint-Based Planning* method for hybrid projects starts by identifying the critical milestones in one project on which the other project depends, and planning the latter around the former.

For example, production deployment cannot occur before the production environment is ready, even if the software is ready for release well in advance. In this case, the development team would continue to develop new features (e.g., reports) until the production environment is ready, and then deploy the new capabilities to production.

While the project as a whole is hybrid in nature, from the “top” or containing framework it would most likely be viewed as a waterfall or SDLC project. This is because the waterfall nature of some of the sub-projects will be reflected in the organization of the overall project. In contrast, the Scrum sub-projects, which are designed to be more adaptive than predictive, will then adapt to the constraints and predicted schedules that define the rest of the project.

1.4 How to Manage Complex Hybrid Projects

Large, complex projects, which contain multiple communicating subsystems that have strong dependencies on each other, present more challenges than the example above, but the challenges are of the same kind. While a pure-Scrum project developed by one team has great freedom to develop features when desired, development in complex multiple-subsystem environments is strongly constrained by dependencies.

In these cases, planning often crosses subsystem boundaries, and requires defining a careful choreography of development steps to be performed by different teams at specific times. This kind of work is a common exercise in Program Management, and the basic concepts are not much affected by the use of Scrum in some subset of the

projects. Scrum enters the picture when scheduling work to be done in a coordinated fashion across projects, as this work is assigned to specific Sprints in the Scrum projects.

2 Scrum as a Project Phase

The standard definition of Scrum assumes a steady stream of work, not a project of fixed duration. While each release can be considered a project in the latter sense, the steady sequence of releases is similar to the concept of "operations," rather than "project." As a result, confusion often arises when Project Managers try to figure out where Scrum fits within the context of a fixed-duration project.

Sometimes the answer is that the project is ongoing, rather than of fixed duration, in exactly the sense that Scrum assumes. Feature requests come in over time, priorities change frequently, and the Scrum team puts out a steady flow of releases in a way that best addresses the changing customer priorities.

Sometimes the project is a fixed-duration project, and terminates on completion. In this case, Scrum may fit into the picture in one of two ways:

1) The project is conducted along standard Scrum lines, and simply ends when the desired scope is judged to have been completed.
2) The project has a requirements phase, which may take weeks or months to complete, before any development work is done. In this case, the Scrum part of the process kicks off after the requirements work is complete.

Both scenarios occur, although the latter is often where a Project Manager's mind turns when first thinking about Scrum. Whether the second approach is more or less appropriate than the first should be decided based on the constraints of the project, and the merits of the approach for that project.

3 Budgeting

Budgeting is typically done on an annual basis. Portfolio managers review a list of projects, and decide how or whether to fund each. In order to make these decisions, portfolio managers require information about each project's expected cost, resource requirements (human or otherwise), expected timeline and schedule of investment, magnitude and timing of benefits to be realized (ROI), and relationships or inter-dependencies with other projects in the portfolio.

The introduction of a Scrum process in some parts of the enterprise does not change the budgeting process dramatically. Scrum projects can provide better visibility into project cost and resource usage of past projects, but the way this information is used in the budgeting process is essentially unchanged.

For example, a Project Manager who produces estimates for a project might go through these steps:

1) Get a Preliminary Scope Statement and Benefit Analysis from business analyst
2) Work with implementers to estimate resource and effort requirements
3) Create high-level estimates for the
    a. Project timeline
    b. Implementation personnel
    c. Other resources
    d. Total cost and schedule of investment
    e. Expected benefits (revenues) versus time
4) Provide the estimates to the portfolio managers

Depending on the nature of the project, the Scrum process might be used for all or a part of the project. We will assume that the existing process is used for part of the project, and Scrum is used for the remainder, below, and that budgeting for the existing process is well understood.

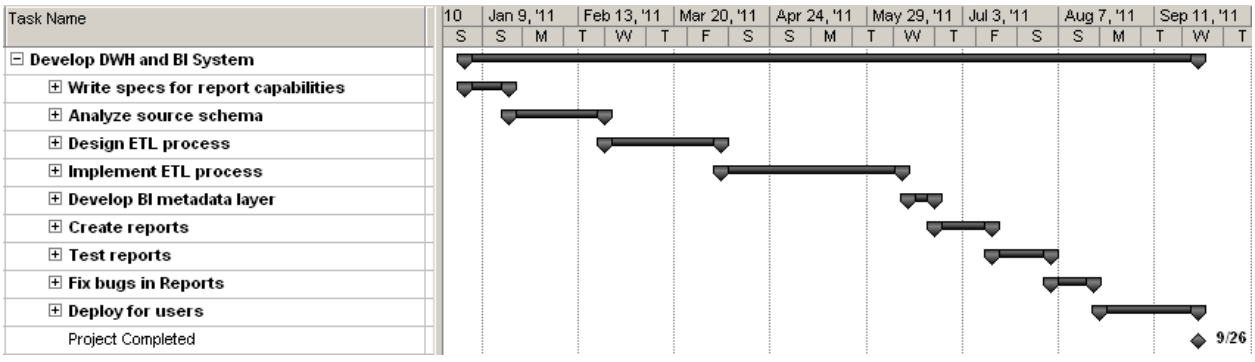
Given these assumptions, migrating from an existing development process to a Scrum process affects the Project Manager's estimation work in relatively minor ways.

1) The subset of the project that uses a Scrum process (part or all, depending on details) will estimate costs on a per-team basis, rather than a per-department basis, since teams often cross departmental boundaries
2) The estimation process may involve more people (up to all members of all teams) than previous approaches
3) The accumulation of detailed actuals and estimates made possible by Scrum projects may improve the reliability of estimates over time

Once these estimates have been obtained, they are supplied for use in the usual budgeting process.

4 Tracking Progress of Projects

Large IT organizations are accustomed to tracking progress against predictive schedules and Gantt charts (at a detailed level) and milestones (at a coarse level), as in the example below.



The key thing to note about the example is that milestones represent project phases. Each project phase has a specific set of activities, planned for completion by specified dates. The Project Manager tracks actual progress,

notes deviations from the plan, and responds to the deviations as needed (the response being either to work to get the project back on schedule, or modify the schedule.)

Scrum projects do not have the same kind of milestones. The reason is that the project is not divided into major phases. Instead, a long-term project is divided up into short-term development cycles (Sprints). Within each Sprint, the Scrum Team typically implements several small-scale features (Stories), all the way from design through testing and bug-fixing. The elapsed time from start to finish per individual feature is usually 2—5 days.

Thus in Scrum projects, we do not track progress against a predicted schedule of major phases. Instead, we track task or feature completion over time.

4.1 Progress Charts

These two charts track progress over time, usually on a daily basis.

- The **Burndown chart** is a type of Estimate-to-Complete chart, and shows the amount of work that remains to be completed in the period of interest (usually a Sprint).

- The **Burnup chart** is a type of Earned-Value chart, and shows the amount of work that has been completed from a particular point in time (such as the start of a project) to the present.

It is necessary to understand how work is defined, and related to requirements specifications, to make sense of the charts.

At the start of a Sprint, the Scrum Team select a set of Stories (features) for implementation, based on priority and size. They first estimate both the amount of work they can accomplish in the Sprint, and the amount of work required to implement the top-priority Stories, and then select enough of the latter to fill their capacity.

Next, the Scrum Team members define the set of tasks required to implement each Story, and estimate the effort for each task. The sum of the task estimates for all Stories selected for the Sprint defines the total effort projected for this Sprint.

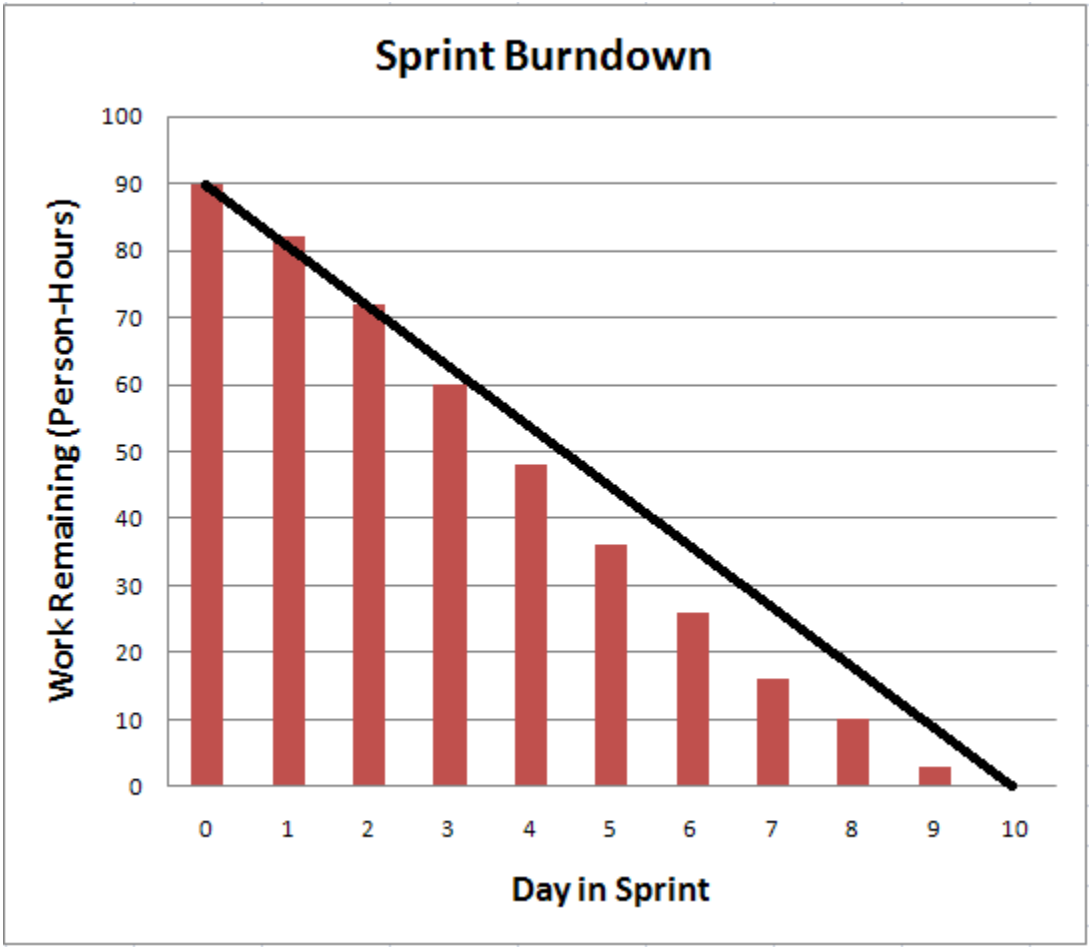


Figure 1 Burndown chart for two-week Sprint

The sample Burndown chart in Figure 1 shows the progress for a Team that is working in a two-week Sprint, which contains ten working days. The total effort, estimated at the beginning of the Sprint, is 90 person-hours.

The ideal rate of progress is indicated by the diagonal line, which trends down from 90 at the start of the Sprint, to zero at the end. The actual amount of work remaining at the end of each day, in the form of uncompleted tasks, is denoted by the red bars. The project is ahead of schedule if the bars fall below the ideal progress line, and behind if they fall above it.

The Burnup chart is similar to the Burndown chart, but shows amount of work accomplished, rather than work remaining. The Burnup chart tracks progress at the Story level, not the task level, and so shows the work associated with completed Stories.

While both Burndown and Burnup charts can be used for any span of time, Burndown charts are most often used for Sprints, while Burnup charts are particularly useful for tracking implementation of large projects across many Sprints. The example in Figure 2 shows a release-level Burnup[1] chart:

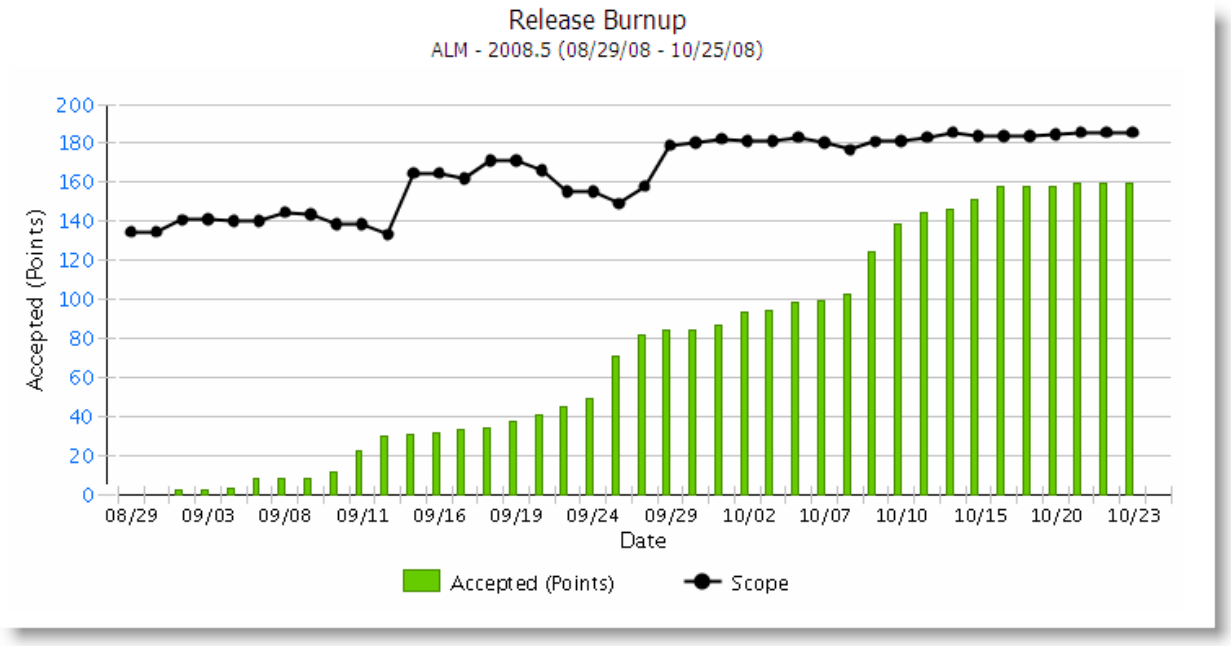


Figure 2 Burnup chart for two-month release cycle

The black curve shows the planned scope, and the variation in this curve over time shows that the project scope itself changes during the release cycle, as is common in Scrum projects. The green bars show the amount of work completed to date, again with granularity at the Story level.

5 Management of Distributed Scrum Projects

The Scrum process makes a conscious trade-off towards communication and collaboration, and away from project documentation. Two of the observations that drive this trade off are the knowledge that

1) Requirements documentation is rarely complete and correct at a detailed level, and requirements evolve during the course of the project anyway
2) Every hour spent writing project documentation that isn't required for the project to succeed takes one hour away from the work of producing deliverables

[1] The vertical axis is in units of Points, which are larger than person-hours (e.g., 1 Point may be defined as 8 person-hours).

For these reasons,

1) Scrum requirements are written very simply, with the intent that the Scrum team members will work out the details with the Product Owner during implementation
2) Design documentation is kept to a minimum, with the intent that design decisions will be worked out through collaboration among team members, with preference given to recording decisions in the code (where future developers will look), rather than in formal design documents (which may never be seen again)

These tradeoffs work well when all team members, the Product Owner, and the ScrumMaster all work in the same room. They work less well as these people spread out, and become difficult when they have no common working hours (e.g., teams with members in the US and India).

5.1 Team Organization

The two possibilities for team organization in geographically distributed projects are

1) Co-located teams in different locations. Each team has all members in close proximity, but different teams are in different locations.
2) Distributed teams. A team has members in different locations.

The case is preferable, as it eliminates the problem of long-distance collaboration. The second case is much more difficult to manage, so we will address it below.

5.2 Managing a Distributed Scrum Team

In trying to make Scrum work with a distributed team, we encounter two general problems, which impact the development process in various ways. Reducing the impact to a tolerable level requires adjustments to the process, and effective use of communication and collaboration tools.

5.2.1 The Problems of Distributed Scrum Teams

The major issues that impair the ability of the team to function include

1) Distribution makes face-to-face collaboration impossible
2) Large time-zone differences reduce or eliminate common working hours
3) Language differences impair communication

5.2.2 Where the Impacts are Felt

The impacts are felt in two general areas

1) Standard team meetings (Sprint Planning, Daily Scrum, Sprint Review, Retrospective)
2) Informal collaboration

5.2.3 Tools to Mitigate Impact

Many software and telecommunication tools are available, which are useful for distributed teams. Some important categories are listed below.

1) Agile Project-Management applications: These are usually externally-hosted (“Cloud” or “SaaS”) applications, designed for agile processes such as Scrum. They store requirements (Product Backlog, i.e. Stories and bug reports), act as bug-tracking systems, define work at the task and Story level, define the

Sprint schedule and backlog, and track work via Burndown and Burnup charts. Examples include Jira, VersionOne, and Rally.

2) Document repositories: There is usually a need for a commonly-accessible location to store documents in addition to any PM application. Many solutions exist. Some of these strictly store documents, while others provide collaboration capabilities as well.
    a. Pure document storage: Wikis, Microsoft Sharepoint, Google Docs
    b. Document storage and collaboration: Confluence, Basecamp
3) Real-Time Communications: Phone, chat, teleconference, Web conference
4) Real-Time Collaboration: Sococo
5) Asynchronous Communication: Email

5.2.4 Collaborating in a Distributed Team

Collaboration in a distributed team is a mix of interactive (real-time) and email-style communication. Requirements and other important documents are written, stored, and accessed with appropriate tools, as above. The challenges are around real-time interaction, when time-zone and language-barriers are present.

5.2.4.1 Team Meetings

There are no obvious best solutions. Various strategies can be used, and some are listed below.

- If not everyone shares a common language, make sure someone who can translate is present
- If working hours overlap, hold meetings at overlapping times
- If working hours don't overlap, pick a meeting time that inconveniences the smallest number of people. (Keep the time standard. Altering times from Sprint to Sprint to move the inconvenience around causes scheduling nightmares.)
- For Sprint Planning meetings, the whole team must be present, even if it is very inconvenient for some people
- For Daily Scrum meetings, if it is not practical to hold them every day with the whole team, consider fallback methods
    - Hold them 2—3 times per week, and have a local representative responsible for identifying and resolving issues quickly between meetings
    - Split into local meetings, and have representatives from each meet or communicate daily
- For Sprint Demo meetings, have the Product Owner review each Story's implementation when it is complete, instead of holding a meeting to demonstrate all of them. (This is often a good idea, anyway.)
- For Retrospective meetings, hold smaller local versions if it is not practical to get the whole team together, and combine the results

5.2.4.2 Informal Collaboration

- Try to arrange for some overlapping work hours. This is usually arranged to inconvenience the smallest number of people. If this isn't possible for all team members, designate a subset to do this (e.g., a senior developer at location A spends one evening hour online to work with team B).
- Use text (chat) communication when people write English better than they speak it

6 Tools for Agile Project Management

These tools contain the standard Scrum artifacts (Stories, bug reports, Product and Sprint Backlogs, Task breakdowns), generate standard tracking metrics (Burndown and Burnup charts), and are used to assign work to Sprints (define a Sprint schedule and backlog). Many tools exist, most of which address the above items.

Agile PM tools are not necessary if a project contains only one or two co-located teams (everyone in the same room). The usual tools for this environment are whiteboards or corkboards to display status, and sticky notes or index cards on which Stories and tasks are written. Team members move cards from “Not Started” to “In Process” to “Done” columns as they work, and the Sprint status is visible by inspection. The ScrumMaster will usually create a daily Burndown chart with a spreadsheet program, based on the status just before the Daily Scrum meeting.

Agile PM tools become a necessity for large (>2 team) or distributed projects. These projects require a common source of project status that is visible to all stakeholders in all locations, is always up to date, and can generate charts and reports on demand.

Unlike many classical PM tools, agile PM tools are designed for use by team members who do the hands-on work of the project. Because these tools useful to team members on a daily basis, they make the acquisition of status information relatively easy. (This is in contract to tools such as Microsoft Project, which provide relatively little value to most team members, and are not used by them.)

Many tools are available. Jira, VersionOne, and Rally, for example, are well-known products. When choosing a tool, the following capabilities should be considered.

6.1 Must-Haves

- Repository for Stories and Bug reports
    - Supports task breakdowns for Stories and Bug reports
- Sprint Definitions
    - Start and end dates
    - Stories and bugs assigned to Sprint
    - Team membership
    - Velocity / capacity definition (for planning)
- Release Definitions (mapping to Sprints)
- Tracks Task, Story, Sprint, Release status
    - Must track all work!
    - Generates burndown, burnup charts from status information
- Scalability
    - Roll up, drill down Sprint and Release information, for hierarchical team structures
- Standard and custom queries
    - Get what you want, when you want it

6.2 Nice-to-Haves

- Test-case support
    - Useful for manual test cases
        - But better to have test cases automated
- Workflow features
    - Guidance for next step (canned or customized)
    - Triggers for email notifications
- Integrations with other applications you use
    - SalesForce, Jira, Bugzilla, QualityCenter, …
- Opportunity to eliminate redundant tools
    - Why have separate bug tracker if Agile PM tool tracks bugs?

7 Dedicated and Shared Team Members

A typical Scrum team has 3—9 members. Over time, these team members develop increasing domain expertise, and an increased ability to collaborate effectively. To get the best results from development teams, Scrum projects require reasonably stable team membership.

"Reasonably stable" does not mean no change. People change jobs, and new people are hired over time, so some evolution in team membership is normal. However, frequent major changes should be avoided. A team of six people can adapt well to changing one or two members per quarter. That same team will function poorly if half of the people come and go with each Sprint. Finally, it may not function at all in a situation where team membership changes arbitrarily with each Sprint.

Questions about "dedicated teams" often arise. A dedicated team is one in which all members work full-time on that team's work. This is the ideal, and an organization should try to approach this ideal as closely as possible, for best performance. However, it is often necessary for some people to be shared across two or more teams. This situation can arise when someone has specialized expertise that is needed for multiple teams, and when no one team has enough work to keep that person occupied full time. (UI designers and Database Architects are two common examples.)

Nothing in Scrum requires a dedicated team. The process will work even if members are shared. The impact of sharing one person across teams is to reduce his efficiency (something that is true for any process, not just Scrum). If sharing is necessary, then one should simply be aware of the impact of sharing on the shared person's effectiveness.

8 Documentation Issues

One of the common preconceptions of agile processes, such as Scrum, is that they eliminate documentation. This belief is incorrect, but it does arise naturally from one of the basic concepts of agile development, which values "working software over comprehensive documentation." Scrum projects do have some documentation. The question is not whether to have documentation, but what documentation to have. Answering this question requires understanding that there are two categories of documentation.

8.1 Deliverable Documentation

Scrum is a type of process intended to produce valuable deliverables to customers as quickly as possible, in a flexible and efficient fashion. If some of those deliverables are documents (such as manuals or online help systems), then there is no question about whether to produce them. The Scrum team will produce the deliverable documentation along with other deliverable items.

8.2 Non-Deliverable Documentation

Not all of a project's documentation is intended for the project's customers. Documentation may also be intended for

- The process itself (e.g., requirements specifications)
- Stakeholders in the company (e.g. deployment instructions for Operations groups)
- Stakeholders external to the company (e.g., regulatory agencies)

Scrum does have something to say about non-deliverable documentation, and it is this: Produce no more of it than is truly necessary for success. The "lean" thinking behind this directive is that every hour spent writing documentation that isn't truly necessary is an hour not spent producing something of real value.

The concept seems simple, but can be challenging to practice. The difficulty is that many processes require documentation that is used very little, or not at all. This practice is often driven by past failures, for which the perceived solution is to "tighten the process" and "document more carefully." Unfortunately, the result is often to expend valuable time creating artifacts solely to support the process.

These documents often

- Have little or no value for any other purpose
- Are often "faked" through an exercise in copying-and-pasting
- Fail to achieve the desired result (improving the chances of success)

If projects keep failing, they can enter a "death spiral" of increased overhead until they grind to a halt.

The solution to project failure isn't just adding more process and more documentation, but devising a process that works better. Scrum is an effective process partly because it focuses on doing things that provide value, and avoiding things that do not. This philosophy should be applied throughout the project. Every suggestion to create an artifact that isn't delivered to the customer should be examined to see if it is truly necessary. If it is, then thought should be given as to the most efficient, least-effort solution that meets the need, rather than blindly filling out a template.

Finally, when documentation is produced, it is a good idea to write it once, and leverage it many times. Thus the requirements specifications and bug-fix requests that a Scrum team turns into working products can also provide the definition of release content used by Customer Service personnel when investigating customer queries. (The motto: "Write once, read many times.")

9 Requirements Specification in Scrum

Contrary to some beliefs, Scrum projects do use written specifications for product requirements. These come in three forms: User Stories, Technical Stories, and Defects.

1) User Stories are short narrative descriptions of a user-facing feature
2) Technical Stories are short descriptions of infrastructure requirements
3) Defects are descriptions of product failures (bugs)

User Stories are short descriptions of a capability than can be implemented in a few person-days of work. (Larger capabilities should be broken up into smaller stories.) Each consists of a title, a description, references to external documents (such as tax tables or user interface prototypes), and a description of how the implementation should be tested.

Technical Stories are short description of infrastructure work that must be done in order to support other requirements, such as performance or specific user-facing features. Their format is similar to User Stories, except that the description need not be in narrative form.

Defects are reports of product failures (bugs). They are requirements specifications because they are prioritized, scheduled, and worked on in exactly the same fashion as Story-type requirements.

Stories (User or Technical) are kept brief for two reasons.

1) The work required should be under 20 person-days, and so large descriptions shouldn't be necessary
2) The amount of detail in the Story should be enough for the Scrum team to use in making rough estimations. No more is needed for the purpose of estimation, and further details emerge during implementation work as Scrum team members and the Product Owner collaborate to complete the implementation.

10 Rolling out a Scrum Process to Multiple Teams and Projects

Training multiple teams to plan and execute Scrum projects is not much more difficult than training a single team. While the number of people involved is larger, the training process is essentially the same.

The options range from migrating one team at a time to Scrum, to migrating all teams in a company at the same time. The decision depends on how tight the coupling is between teams. If a project contains multiple teams, the coupling between teams in the project is usually tight. Coupling between different projects can range from nonexistent to tight, but is usually not as tight as between teams within a project.

The common practice is to migrate teams on a project basis. A project whose people are (or will be) organized into multiple teams normally requires a high degree of coordination among the teams. Close coordination is necessary because the product under development incorporates work from all teams. The need to coordinate work means that all teams on a project should migrate to Scrum at the same time, and work according to the same Sprint schedule.

Migration of projects follows common-sense guidelines

- Projects that are independent can migrate on a schedule that is convenient for them.

- Projects that are tightly-coupled should migrate at the same time.
- Whether to migrate projects with moderate coupling together or separately is a judgment call, which weighs the benefits of simultaneous migration against other issues. This decision has to be made on a case-by-case basis.

11 Organizing Cross-Team and Cross-Project Implementation

A Program in a large enterprise or business unit will contain many projects, and each project may contain many teams. The hierarchy of teams and projects is shown in Figure 3.

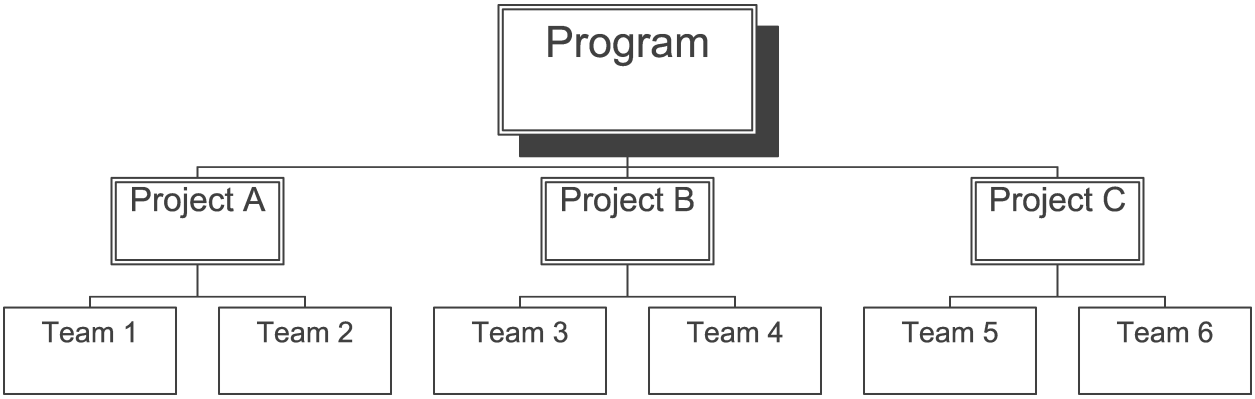


Figure 3 Project and Team Hierarchy

Teams within a project usually work in close coordination with each other. Teams in separate projects may be closely coordinated or completely independent, but most often work together on an occasional basis, as needs dictate. We will look at how different teams collaborate below, and summarize the most general case for requirements management in Figure 4.

11.1 Collaboration within One Team

In Scrum projects, requirements for each team are defined by the team's Product Owner, and work to implement the requirements is managed by the ScrumMaster. This type of collaboration is clearly defined by Scrum.

The Daily Scrum meeting provides the opportunity to produce a common awareness of progress, and to identify any issues encountered by anyone on the team that require help from someone else to resolve. The resolution of issues occurs after the meeting, when the relevant people work together on them. The team also views the current Burndown chart to see how team progress as a whole is tracking against the plan.

11.2 Cross-Team Collaboration within a Project

Scrum does not specify how teams within a project collaborate, but there are common practices. The practices focus on requirements definition, scheduling, and execution. Collaboration between teams is necessary when a product

specification requires synchronized work across the teams. This collaboration is described below, and the requirements-management portion is illustrated in Figure 4.

11.2.1 Planning

A project that contains multiple teams will have anywhere from one Product Owner, to as many as there are teams. (Often, a single Product Owner will provide requirements for two or three teams.) If there is only one Product Owner, synchronization is relatively easy, as he is familiar with all of the project work. He will work with ScrumMaster(s) and team members to identify dependency and schedule issues, and decide when to schedule the necessary work (i.e., in which Sprints to implement the relevant Stories).

In a project that contains two Product Owners, the Team-level Product Owners may be able to collaborate informally to achieve the necessary synchronization. Informal collaboration becomes impractical as the number of “Team Product Owners” grows, at which point it is necessary to have a single person who is responsible for identifying and planning cross-team work. This person might have a variety of job titles (Senior Product Manager, Senior Business Analyst, etc.), but Scrum does not have a term for the role. We will call him simply the “Product Owner,” and define this role as being responsible for the product as a whole, while the Team Product Owners are responsible for requirements to be implemented by their Teams.

The Product Owner will work with the other Product Owners to identify incoming requirements that have cross-team impact, and break the requirements (“Epics,” in Scrum terms) down into per-team requirements (“Stories”). Individual Product Owners will work with the teams to validate and revise the team-level Stories, and understand their impact and dependencies, then work with the Lead PO to finalize the schedule (allocation of Stories to Sprints).

A typical organizational structure for multiple teams and products is illustrated in Figure 4.

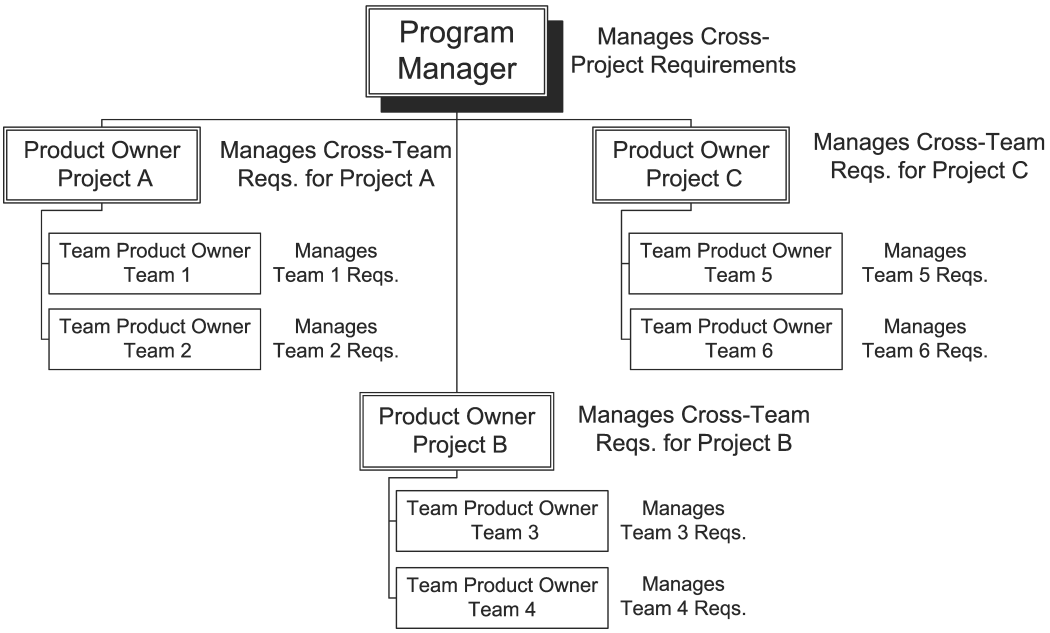


Figure 4 Requirements Management Hierarchy

11.2.2 Execution and Tracking

During the Sprint, teams work to implement requirements. Members of different teams, who need to collaborate on interface issues, will work together informally, as needed, just as members within the same team do.

Tracking and issue identification and resolution occur in the usual ways at the team level. At the cross-team level, one representative from each team comes to a daily “Scrum of Scrums” meeting (SoS). The SoS meeting is similar to a Daily Scrum meeting, but focuses on issues that have cross-team impact. These issues include delays that impact another team, questions or decisions that need quick action, and so forth.

The cross-team tracking metric is again the Burndown chart, but now viewed at a Project level, rather than a team level. (It is convenient to have an application that can generate these charts at different levels, on demand.)

11.3 Cross-Team Collaboration across Projects

Collaboration across Projects is very similar to collaboration between different teams in the same project. The difference is mostly one of scale. If the number of Projects is large enough, one Lead Product Owner may not be able to manage cross-Project requirements, in which case it is necessary to create a cross-project Product Owner hierarchy that contains a number of Lead Product Owners. Scrum does not have a title for the person who leads this group, but “Program Manager” is often used, and we’ll adopt the term here.

The Program Manager is responsible for working with the Lead Product Owner group to identify cross-project requirements, and ensure that they are broken down into team-level Stories, and the process described in Section 11.2.1 is followed successfully.

The Program Manager may also be the person to conduct “Cross-Project Scrum” meetings, attended by one representative from each SoS meeting, and whose purpose is the same as the SoS meeting.

The tracking metric for cross-project work is the Enterprise-level Burndown chart.