Agile Documentation with uScrum (full paper)

Joaquim Baptista
Altitude Software
Alameda Fernão Lopes, 16 – 4 andar
1495-136 Algés, Portugal
+351-21-412-9800
px@acm.org

ABSTRACT

uScrum (uncertainty Scrum) is an agile process developed by a small team at Altitude Software to manage the process of gathering information and writing documentation.

uScrum proposes different practices to estimate tasks, depending on their difficulty as defined by an order of ignorance. Understanding the difficulty of tasks has allowed the team to effectively prioritize regular work together with difficult creative work.

uScrum overbooks writers on iterative cycles called sprints, then lets the writers micro-manage their tasks to overcome obstacles. After each sprint the team decides what to publish and whether to proceed with unfinished work. uScrum manages uncertainty and the unknown, allowing writers to quickly react to changing conditions.

Categories and Subject Descriptors

K.6.1 [**Management of computing and Information Systems**]: Project and People Management – *Life cycle, Management techniques.* H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Training, help, and documentation.*

General Terms

Documentation, Management, Measurement, Theory.

Keywords

Agile documentation, Agile management, Orders of ignorance, Complexity measures, uScrum, Writing, Overbook, Wiki, Risk, Uncertainty.

1. INTRODUCTION

uScrum originates from the unique combination of the writing conditions at Altitude Software, the history of the company, and the self-improvement efforts of the documentation team.

1.1 Writing at Altitude Software

Altitude software products have been evolving since 1995 as a result of market needs, customer needs, and bug fixes. Specific issues can quickly gain importance as customers start to use new or existing features in innovative ways. Therefore, technical writing at Altitude Software is open-ended and never finished. Work done today may need to evolve or improve tomorrow.

There are no documentation projects at Altitude Software in the sense described by JoAnn Hackos[9], for example. However, the documentation team has almost complete autonomy to negotiate the documentation deliverables, and can often phase the delivery of larger work or delay the delivery of less important work.

Different developers follow different practices to communicate and share information, ranging from open sharing to "need to know." Typically, there are no detailed design specifications for new features, meaning that writers spend most of their time gathering and organizing information.

Variants of the same information are explained to very different audiences, including end users, system administrators, and developers.

1.2 Company and Team History

Altitude Software started as Easyphone in 1995. The documentation team started in 1997 and, by 1999, had grown to three writers. As the company grew and merged with Intervento to become Altitude Software, the documentation team grew to nine in 2000, with plans to hire five more in 2001.

The global economic downturn of 2001 forced the company to downsize. Between 2002 and 2006, a team of three writers struggled to maintain and evolve the documentation set as the Altitude Software products evolved. One of the writers served as part-time manager, while another served as part-time editor.

A major issue was the question "what do I do next?" that was often asked by writers hitting a roadblock. Answering that question a few times each week hindered the concentration required by the part-time manager for harder writing tasks.

By 2003, writers already worked at more than one task at the same time, being asked to juggle between tasks while waiting for developers. In 2004, the team devised an iterative management process, hereby named uScrum, influenced by the unorthodox solutions advocated by Russell Ackoff[2, 3] and Tom Peters[13, 13], among others.

As the team grew to five writers in 2008, the team evolved uScrum by adapting some practices from Scrum[19, 10] and improving the publishing decisions.

2. THE ART OF ESTIMATING TASKS

The documentation team introduced metrics in 2000 to counteract a specific development team that kept redesigning the graphical interface of a product. The metrics succeeded in exposing the large effort required by the documentation team to keep pace with irrelevant changes.

After collecting metrics for several months, the team understood documentation tasks well enough to accurately estimate the time required for the simpler tasks. However, many tasks took much longer than their estimated times.

For some tasks, although the estimated effort was close to the actual effort, the effort did not translate into the same number of calendar days. For example, an effort of three days might still take two weeks to complete. The difference was due to the time spent waiting for developers to provide information or for reviewers to finish their reviews.

Other tasks just failed in spectacular ways. For example, a task estimated to take two weeks of effort (not calendar days) might not be anywhere near completion after four or six weeks of dutiful work.

The seminal article by Philip Armour[4, 5] provided the crucial insight to understand the discrepancies. As a result, the team started to classify tasks into orders of ignorance, named as *0oi* to *3oi*.

2.1 Tasks at *0oi*

Writers know everything needed to complete the task. This happens when writers update a document, for example to improve or expand the description of known concepts. Typically writers can estimate these tasks accurately in terms of both calendar days and effort required.

2.2 Tasks at *1oi*

Writers know what they need to write, but the information must still be gathered from developers. Writers may also discover something unexpected that requires extra effort. For example, a concept may be especially hard to explain, or the writer may discover extra configuration parameters not initially known.

Typically the writers can estimate the effort required as a range of days, but not the number of calendar days. For example, a task may be estimated to take three to five days, or even one to five days (one day if the task is trivial as expected, or up to five days if extra work arises).

Tasks at *1oi* form the majority of maintenance work. Knowing the different teams of developers, writers learned when to expect extra work associated with apparently simple maintenance tasks.

2.3 Tasks at *2oi*

Writers know the problem to solve, but the team has never solved a similar problem. This means that writers don't quite know what to ask, or don't know how to organize the information gathered.

Sometimes writers find a shortcut that enables a quick solution for an apparently hard problem. For example, a task with many difficult variants may suddenly become simple if a writer realizes that only a specific variant applies, thus killing the source of uncertainty. In such cases, ten days of estimated effort may actually cost a single day, typically spread over several calendar days.

At other times the hard problem may be abandoned after investing some effort without any reasonable outcome. Tasks at *2oi* tend to resist any kind of estimating.

2.4 Tasks at *3oi*

Writers try to solve a hard problem but are not sure how to proceed.

This kind of problem is not common but has occurred several times in practice, as a result of unexpected consequences when many documents were being rewritten according to a different set of principles, in a way similar to refactoring code.

2.5 How to Estimate Tasks

Estimating the effort required by a task requires the team manager to understand the issue. However, fully understanding an issue in order to get an accurate estimate may take as long as half the total effort, by which time the team manager might as well finish the work. In other words, getting an accurate estimate for a task is just too expensive to be practical.

Classifying tasks by their order of ignorance allows the team manager to apply different estimating strategies to tasks of each order.

For small tasks expected to last days or at most a few weeks, the team manager just guesses based on past experiences. However, *2oi* tasks guessed to take more than a week often surprise the writers assigned to the task.

For large tasks at *0oi* or *1oi*, the team manager finds some item to count (for example, screens, parameters, errors, or even pages), and estimates based on a guessed average effort to complete each item.

For harder tasks, estimates are often pointless, either because the objective is poorly defined or because the steps to reach the objective are not known. However, it may be possible to estimate the effort to complete some initial work.

3. AGILE DOCUMENTATION

uScrum prioritizes tasks into monthly iterations called sprints, based on their importance, urgency, and timeliness. To juggle longer and harder tasks with small important tasks, the team must develop a shared long-term vision of major work as well as a feeling for the importance and urgency of upcoming short-term work.

Sprints are based on the idea of overbooking writers with more work than they should reasonably be expected to complete, and then letting writers micro-manage the exact work that gets done. In a month with 22 working days, for example, a writer may have work estimated to take 18 to 30 days.

The monthly iterations allow the team to react quickly to the changing needs of internal and external customers. Requiring new “urgent” work to be prioritized at the end of the month has kept most “urgent but not important” work at bay, preventing interruptions caused by work that turns out to be useless. Given the right resources and information, writers usually start important tasks before other teams ask for them.

Over the years the team has used different sets of meetings to coordinate sprints, to decide what to publish, and to decide whether to proceed with unfinished work.

3.1 Collecting Tasks

The team manager collects tasks by attending coordination meetings and monitoring internal sources of information such as mailing lists. Tasks can be features to document, issues to solve, or objectives to achieve. Writers also suggest tasks, for example after noticing interface changes or as a result of interacting with developers.

The current set of tasks to do is called the product backlog or just the backlog.

Over time, document sets also require major improvements to the depth, breadth, or organization of information. The team conducts yearly strategy meetings to provide overall vision and direction, motivating the team to pursue new major goals. For example, to address new audiences or new needs such as training. These improvements typically require large efforts.

3.2 Dates and Commitments

Since managers in coordination meetings are often expected to provide dates (or a least a date for having a date), the team manager must estimate tasks and make some commitments.

The team manager can often promise small and important tasks for the next sprint or even the current sprint if the task is small enough.

The team manager can commit to perform larger tasks, but avoids providing any firm dates because more important tasks can appear in the meantime. The team manager only commits to harder *2oi* tasks as a best effort because, by its own nature, these tasks are impossible to estimate.

Refusing to promise dates for what appears to be urgent work is quite unpopular with managers that want to predict the future. However, it is often enough to promise to tackle the issue, especially to managers that learned to trust the judgment of the team manager.

The lack of dates is now tolerated because the team tends to tackle important issues and reacts quickly to urgent needs. The answer to the question "When will this be ready?" is often "It is already done" or "Next month."

3.3 Task Selection for Sprints

The team chooses what to do based on the skills required by each task, how useful the output of the task will be for customers, and the timeliness of the task.

Writers must judge the importance of each task to decide what gets done immediately and what can be safely delayed. For example, some maintenance tasks are of general use while others matter to a single customer.

Some tasks have an ideal timing, normally during or after a feature is tested. Documenting those features near their best timing optimizes the time of writers, testers, and developers.

Hard tasks may require that the team delays short-term maintenance work in some sprints.

Tasks may linger in the backlog for years without ever being started. These are either nice ideas that are too hard to put into practice, small tasks that are just not important enough, or work that cannot be done for lack of resources.

3.4 Writers in Sprints

Since estimates vary with the writer assigned to the task, writers renegotiate estimates at the beginning of the sprint. The understanding is that the writer should not spend significantly longer at a task than initially estimated without calling the manager's attention to that fact.

The writer that takes a task will often be the first person to try to understand the required effort in detail, which means that the initial estimate is often inaccurate.

Writers are free to help each other during sprints, as long as the team metrics remains accurate and the total effort does not exceed the initial estimate.

3.5 Handling Unfinished Tasks

At the end of the sprint, each task may be finished, not finished, or not started. In the next sprint, the team may decide to continue or stop the unfinished work.

The team may decide to stop the work abruptly if progress is not as expected, or if there is no progress at all. This happens if writers try to document new features too soon, when the interface or the behavior are not stable.

The team may also decide to finish some of the work done and stop further work. This captures some benefit from the effort done, but allows the writer to start working at other tasks. In effect the writer performed only part of a larger task.

Unfinished tasks at *0oi* and *1oi* are typically waiting for input from a developer, either basic information or the results of a document review. Also, the writer may have started the task just before the end of the sprint, or the task may have turned out to be larger or harder than expected.

Tasks at *2oi* often reach the end of the month unfinished, since they are both harder to do and harder to estimate. Typically, there is some partially completed work to show for the effort.

3.6 Meetings and Publishing 2004–2007

At the end of each year (and sometimes more often) the team holds a *Strategy meeting* to evaluate the work done in the previous year and to identify a few major goals for the next year. These goals guide the selection of hard tasks for sprints.

In 2004, the team had meetings at the beginning and at the end of each month. The first meeting defined what to do in the month, and the end meeting evaluated the tasks done during the month. These meetings initially lasted about two hours each, but gradually folded into a single end-of-month meeting, lasting two or more hours. During the sprint, the team used the coffee-breaks to discuss small details and scheduled *ad-hoc meetings* as needed.

Between 2005 and 2007, the team held *Weekly meetings* on Tuesdays to share issues and develop a better understanding of changing priorities. The meetings lasted between one and two hours. Gradually the first Tuesday of the month took over as end-of-month meeting.

The team experimented with *Stand-up meetings* during 2007, but these were abandoned because they added little value to the weekly meetings.

Although the team typically used monthly sprints, sometimes it was useful to have 15-day sprints when planning a month in advance was impossible or inconvenient.

Urgent and important changes affecting a single document could often be published immediately. Larger changes required further coordination to publish the documents in a consistent state, typically one or two weeks later. Sometimes the team also published unfinished work internally, especially if the work was deemed to be immediately useful within the company.

In 2002 and 2004 the team also complemented the published documentation with a series of light workshops[6] mostly aimed at sales, service, and support staff.

Between 2002 and 2006, the time required to manage the team of three writers was typically in the range of three to five days each month, including team meetings, product meetings, team evaluations, and backlog maintenance.

3.7 Meetings and Publishing in 2008

In 2008 the team grew to five writers (requiring extra coaching from the team manager) and the weekly Tuesday meetings were no longer an effective way to coordinate the team. Inspired by the practical advice of Kniberg[10], the team separated different concerns into different meetings and formalized the sprint deliveries.

Table 1. Change-of-sprint schedule in 2008

Monday	Thursday	Friday	Monday
What to publish	Publish	Sprint retrospective	Sprint kick-off
	Sprint review		

Sprints are now aligned on week boundaries and typically take 4 weeks (a lunar month), although bank holidays and company events have forced different durations. Table 1 shows the schedule of the meetings and publishing events.

The sprint starts with a two hour *Sprint kick-off meeting* that assigns tasks to writers. During the sprint, the team holds 15-minute *Stand-up meetings* two or three times a week, replacing the weekly meetings used previously. Stand-up meetings are followed by extra *ad-hoc meetings* as needed to address identified issues.

The last week of the sprint focuses on publishing and process. The week starts with a *What to publish meeting* where writers negotiate what work to publish, especially unfinished work. Tasks can be finished and integrated into the documentation set until Thursday morning, when the documentation set is republished.

On Thursday afternoon, writers present significant work to each other in the *Sprint review meeting*. These presentations foster a sense of accomplishment and spread knowledge through the team, which is especially important for new writers. Some presentations may later be offered to other audiences as light workshops[6]. Each presentation takes at most one hour.

On Friday, the *Sprint retrospective meeting* reviews the metrics of the sprint, determines what went well and what went wrong, and proposes improvements. The next sprint starts on the following week.

4. SPRINT PATTERNS

Writers perceive sprints differently, depending on the size and difficulty of the tasks selected for a sprint. In general, the team needs to create *bubbles* in the regular maintenance work in order to tackle harder problems.

4.1 Get Things Done

All writers have tasks at *0oi* or *1oi*. At the end of the sprint, many tasks are finished or partially finished. For example, documents are under review.

4.2 Try With Fallback

A writer has one task at *2oi* complemented with several less important tasks at *0oi* or *1oi*. The writer advances the hard *2oi* problem as possible, while filling the slack with more predictable work.

Although at the end of the sprint the writer may have solved the hard problem, it is more common for the writer to understand the problem better, enabling the completion of partial work and more informed decisions to proceed, delay, or drop the effort.

4.3 Just Try

One or several writers have nothing but a *2oi* problem to solve. The writers should just advance as much as possible during the month. The problem typically requires the full attention of the writers and does not depend from the availability of the developers.

4.4 Cannot Go Back

Most or even the whole team works at a single *2oi* problem (or a set of related *2oi* problems) that makes the documentation set unpublishable for a while. The team must reach "the other side" of the problem before regular maintenance work can resume again.

This kind of disruptive work appeared when the team migrated the documents to new tools, developed new document sets from scratch, or reorganized the documents according to different principles.

4.5 Uphill Battle

The implications of disruptive work are hard to understand. Sometimes sprints of disruptive work keep getting harder and harder, suggesting that the initial problem was *3oi* instead of *2oi*. The team feels lost, without a clear notion of where it is in the process, and how long it will take to finish the disruptive work.

Table 2. Assumptions of Scrum and uScrum

	Scrum assumptions	**uScrum assumptions**
Product backlog	The top of the product backlog is fully prioritized and estimated.	*2oi* tasks cannot be reasonably estimated, although the team can estimate the effort that it is willing to invest at solving a task in each sprint.
Sprint goal	Team demonstrates an objective to the product owner in the sprint review, based on a realistic sprint backlog.	Writers show results for the effort invested. Overbooked writers select what work gets done during the sprint.
Team	Work must be coordinated among team members.	Work is mostly independent of other writers.
Sprint tasks	The sprint backlog can be broken down into 4 to 16 hour tasks in the first day of the sprint.	Finding out what to do or how to do it is a large part of the work that cannot be fully anticipated.
Sprint progress	A burndown chart can track the progress through the sprint.	There is nothing interesting to count down during a sprint.
Obstacles	The team can complete its tasks because it has the necessary skills and resources from the beginning. The Scrum Master can quickly remove any obstacle.	Some obstacles simply cannot be removed or anticipated by the team, such as a developer going on a two week vacation immediately after finishing a project.

The team just keeps working in what appears to be the right direction, typically trying to break hard problems before performing significant work.

For example, in 2005 the team migrated the documents from proprietary XML to DITA[15, 16]. After converting contents for a few sprints, the team realized the need to break the contents into much smaller topics than initially expected, which required reorganizing and rewriting most documents. It took a few extra sprints to reach the expected results. In effect the team had to skip *Level 1* and move to *Level 2* in the recently proposed DITA Maturity Model[17].

5. RELATED WORK

JoAnn Hackos[9] remains the classical reference for managing well-defined documentation projects, namely for consultancy work. Much less seems to be written for teams that maintain and evolve documentation sets over large periods of time, especially if hiring extra writers to absorb peaks of work is not an option.

There are reports of writers participating in agile development teams, but not of documentation teams applying agile practices to the documentation team itself. Still, uScrum can be compared to two agile development methods that focus more on project management than on software development practices.

5.1 Scrum

The documentation team learned about Scrum[19] in April 2005 after a former co-worker attended a presentation at ISCTE[7] and noted the similarities with the practices of the team.

Over time uScrum adopted some terminology from Scrum, but the two approaches are fundamentally different even if some meetings and artifacts are similar. Table 2 shows how the two methods assume different things about the work that is performed in each sprint.

Still, the team adopted the name uScrum to emphasize the philosophical similarities of the two methodologies, where tight teams fight together against large problems broken into pieces.

5.2 Lean Software Development

Mary and Tom Poppendieck stated seven principles of lean software development[14], based on the Toyota Production System[11]. These principles resonate with the experience of uScrum.

Principle 1: Eliminate waste. uScrum started as a way to minimize the cost of managing a small team. uScrum avoids the cost of detailed estimates for work that may never be done, or that will be done nevertheless.

Principle 2: Build quality in. Deliver and validate incremental improvements instead of reviewing complete documents.

Principle 3: Create knowledge. The whole job of the writers is to find, organize, and encode knowledge.

Principle 4: Defer commitment. Sprints delay the commitment to perform tasks until it is time to start them. Overbooking allows writers to delay tasks that are not ready to be started.

Principle 5: Deliver fast. Deliver improvements at the end of each sprint, or sooner if possible.

Principle 6: Respect people. Writers appreciate the freedom of managing their daily workloads, letting the work and the metrics support their choices.

Principle 7: Optimize the whole. uScrum keeps writers working at important tasks and prevents distractions from urgent but not important tasks.

6. CONCLUSIONS

uScrum describes the daily practice of a small team of writers, so generalizations should be made with some caution.

However, the team uses practices that are not reported elsewhere and that may be of more general use even if, in the tradition of agile processes, each team should adapt the process to their unique needs. At the very least, writers like the autonomy that uScrum gives them in managing their daily work.

7. ACKNOWLEDGMENTS

The author would like to thank all the present and past employees of Altitude Software for enduring and helping to shape uScrum, with special thanks to Pedro Pimenta, Derek Watson, and Célia Cerdeira.

8. REFERENCES

[1] TWiki – the open source wiki for the enterprise. http://twiki.org. Accessed on 2008-04-25.

[2] R. L. Ackoff. The Democratic Corporation. Oxford University Press, New York, 1994.

[3] R. L. Ackoff. A Empresa Democrática. Difusão Cultural, Lisboa, 1998. Portuguese translation.

[4] P. G. Armour. The laws of software process. Communications of the ACM, 44(1):15–17, January 2001.

[5] P. G. Armour. The Laws of Software Process: A New Model for the Production and Management of Software. Auerbach Publications, Florida, 2004.

[6] J. Baptista. Just-in-time training workshops. STC Management SIG News, 8(2):14–16, October 2004.

[7] B. Câmara. Desenvolvimento ágil usando Scrum. Presentation in Portuguese at the ISCTE workshop "Metodologias Ágeis de Desenvolvimento de Software", April 2005.

[8] N. Drakos. Altitude's user-centered wiki becomes a mission-critical tool. Toolkit Case Study G00150519, Gartner, August 2007.

[9] J. T. Hackos. Managing Your Documentation Projects. Wiley, 1994.

[10] H. Kniberg. Scrum and XP from the Trenches: How we do Scrum. Lulu.com, Canada, 2007.

[11] T. Ohno. Toyota Production System: Beyond Large Scale Production. Productivity Press, 1988.

[12] T. Peters. O Seminário de Tom Peters: Tempos Loucos Pedem Organizações Loucas. Bertrand, Lisboa, 1994. Portuguese translation.

[13] T. Peters. The Tom Peters Seminar: Crazy Times Call for Crazy Organizations. Vintage, 1994.

[14] M. Poppendieck and T. Poppendieck. Lean Software Development: An Agile Toolkit. Addison-Wesley Professional, 2003.

[15] M. Priestley. DITA XML: A reuse by reference architecture for technical documentation. In Proceedings of the 19th annual international conference on Computer documentation, pages 152–156. ACM SIGDOC, October 2001. Santa Fe, New Mexico, USA.

[16] M. Priestley, G. Hargis, and S. Carpenter. XML-based technical documentation authoring and publishing architecture. Technical Communication, 48(3):352–367, August 2001.

[17] M. Priestley and A. Swope. DITA maturity model. Whitepaper, JustSystems, 2008.

[18] R. E. Raygan and D. G. Green. Internet collaboration: TWiki. In Proceedings IEEE SoutheastCon, pages 137–141, April 2002. Columbia, USA.

[19] K. Schwaber. Agile Project Management with Scrum. Microsoft Press, Redmond, Washington, 2004.

9. APPENDIX: ARTIFACTS

The documentation team pioneered the use of a wiki[18, 1] in 2002 at Altitude Software both to complement the published documentation with unpublishable knowledge[8] and to capture mundane details such as task lists and meeting minutes. Therefore, uScrum captures backlogs and sprints in wiki pages. Metrics predate uScrum and are still based on spreadsheets.

9.1 Team Improvement Meetings

Since 2002, the team meets at the end of the year to discuss how the team works and what can be improved. Team improvement meetings originated breakthrough ideas and have shaped the team processes over the years.

Each team member answers the following four questions privately, and then the team openly discusses and considers each issue raised.

—What keeps you from being as effective as you would like to be in your position?

—What keeps the unit from functioning as an effective team?

—What do you like about this unit that you want to maintain?

—What suggestions do you have for improving the quality of our working relationships and the functioning of our unit?

9.2 Backlog (ToDo) Pages

The backlog or *ToDo* pages (such as *DocsUci62ToDo*) hold task entries that range from simple bug fixes to large or vague undertakings.

Task entries are bits of text with a short code (used for metrics), a classification, an optional effort estimate, and a description. For example, the sample task entry below describes a task with an estimated effort of 5 to 10 days at *2oi* (second order of ignorance) with the short code *skill.sigdoc.*

5-10d 2oi skill.sigdoc *Write a paper for SIGDOC'08 that explains the management practices of the team.*

Task descriptions can be as terse or as detailed as needed, often using bulleted lists for detail, as well as code snippets and bugs numbers for reference.

Having the backlog as wiki pages allows the team to cooperate in updating the backlog while recording whom changed what. Any new candidate task discovered, either formally (for example by email) or informally (for example by someone noticing a new parameter in the interface), is immediately added to the appropriate backlog page, even if it must later be estimated or classified.

Task entries are grouped, split, or organized as needed. To simplify editing and navigation, the team keeps several backlog pages further organized with headings and subheadings.

Although the backlog page for tool-related tasks exists since 2002, backlog pages for product tasks evolved over time as needed. At different times there were pages for different product versions, for different kinds of changes (to separate new product features from general improvements), and for special projects such as the migration to DITA[15, 16].

9.3 Sprint (Do) Pages

The sprint or *Do* pages (such as *DocsDo2004a*) record the decisions and events of a sprint.

Sprint pages record the period of the sprint, the rationale behind the choice of tasks for the sprint, and the tasks assigned to each writer. If several writers are to work on the same task, the tasks are duplicated for each writer.

During the sprint, writers will take notes of progress or impediments below each task. Therefore, the sprint pages serve as a team status report where updates are immediately visible to the whole team. Significant tasks completed are also recorded in a *team achievements* section.

At the end of the sprint, each writer creates a table that compares the estimated effort with the actual effort taken, and notes the state of each task as completed, rollover, or not started. If possible, writers also estimate the effort required to complete unfinished tasks.

Over time, the sprint pages grow into a useful team memory of past efforts. For example, the page *DocsDo2005d* recorded the first contact of the team with Scrum[7].

9.4 Metrics Spreadsheets

Writers collect metrics in a spreadsheet with rows for days and columns for document or activity. The minimum trackable time is a half hour. Writers add columns as needed, although at different times there were different guidelines for what should be tracked. Table 3 shows a small part of a metrics spreadsheet. For example, in April 1 the activity *skill.sigdoc* took 5 hours. Writers fill the metrics daily and summarize their metrics at the end of each month.

The metrics were designed to be easy for writers to fill in, even if they later require more work to consolidate. Metrics are consolidated at the end of the year or when the need arises. The team manager takes about a day to consolidate the metrics of the team, the harder part being the need to consolidate the column titles among writers and throughout the year.

Table 3. Sample metrics spreadsheet

Date	Meetings	skill.sigdoc	Total	Notes
1-Apr	3	5	8	R&D meeting
2-Apr	1.5	6.5	8	Team meeting
Total	4.5	11.5	16	