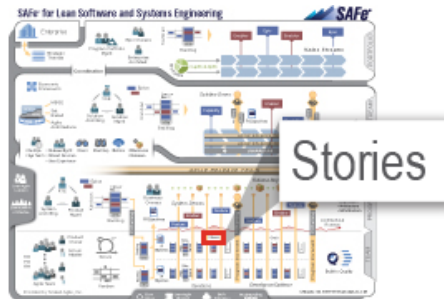


<https://twitter.com/ScaledAgile>
<https://www.linkedin.com/grps/Scaled-Agile-Framework-4189072?>

<https://www.youtube.com/user/scaledagile>
<https://www.slideshare.net/ScaledAgile>



(/)

II *Stories act as a “pidgin language,” where both sides (users and developers) can agree enough to work together effectively.*

—Bill Wake, co-inventor of Extreme Programming

Stories Abstract

Stories are the primary artifact used to define system behavior in Agile development. Stories are not requirements; they are short, simple descriptions of a small piece of desired functionality, usually told from the user’s perspective and written in the user’s language. Each is intended to support implementation of a small, vertical slice of system functionality, supporting highly incremental development. In Agile development, stories largely replace the traditional requirements specifications, or they’re used later to compile any mandated, traditional requirements documentation.

Stories provide just enough information for the intent to be understood by both business and technical people. They are a “promise for a conversation,” intended to serve as a focal point for a more thorough discussion of the intended behavior and impact. Details are deferred until the story is ready to be implemented. Through acceptance criteria, stories get more specific as they are implemented, helping to ensure system quality. Acceptance criteria can be captured and automated in acceptance tests. These tests confirm that the functionality has been implemented properly, both when the story is written and later, as the Solution evolves. This is a critical element of SAFe Built-in-Quality practices.

Enabler stories are another type of story. They do not describe system functionality; rather, they are used by the teams to bring visibility to the work items needed in support of exploration, architecture, and infrastructure.

Details

SAFe describes a four-tier hierarchy of artifacts that describe functional system behavior: Epic (/epic/) > Capability (/features-and-capabilities/) > Feature (/features-and-capabilities/) > *Story*. These, along with Nonfunctional Requirements (NFRs) (/nonfunctional-requirements/), are the Agile requirements (system behavioral) artifacts that are used to define system and Solution Intent (/solution-intent/), model system behavior, and build up the Architectural Runway (/architectural-runway/).

Epics, capabilities, features, and Enablers (/enablers/) are used to describe the larger intended behavior, but the detailed implementation work is described via *stories*, which constitute the Team Backlog (/team-backlog/). Most stories arise from business and enabler features in the Program Backlog (/program-and-value-stream-backlogs/), but others emerge from the team's local context.

Each story is a small, independent behavior that can be implemented incrementally and that provides some value to the user or the Solution (/solution/); it is a vertical slice of functionality to help ensure that every Iteration (/iterations/) delivers new value. To accomplish this, stories are split (see below) as necessary so they can be completed in a single iteration.

Initially, stories are typically written on an index card or sticky note. The physical nature of the card creates a tangible relationship between the team, the story, and the user and helps engage the entire team in story writing. They have a kinesthetic element as well; they help visualize work (/visualize-and-limit-wip-reduce-batch-sizes-and-manage-queue-lengths/) and can be readily placed on a wall or table, rearranged in sequence, passed around, and even handed off when necessary. They help teams better understand scope ("Wow, look at all these stories I'm about to sign up for") and progress ("Look at all the stories we accomplished in this iteration").

While anyone can write stories, approving stories into the team backlog and accepting them into the system baseline is the responsibility of the Product Owner. Of course, stickies don't scale well across the Enterprise (/enterprise/), so stories often move quickly into Agile project management tooling.

There are two types of stories in SAFe, user stories and enabler stories, as described below.

Sources of Stories

In SAFe, stories are generally driven by splitting business features and enabler features, as Figure 1 illustrates.

Business Feature:

Sound simulation

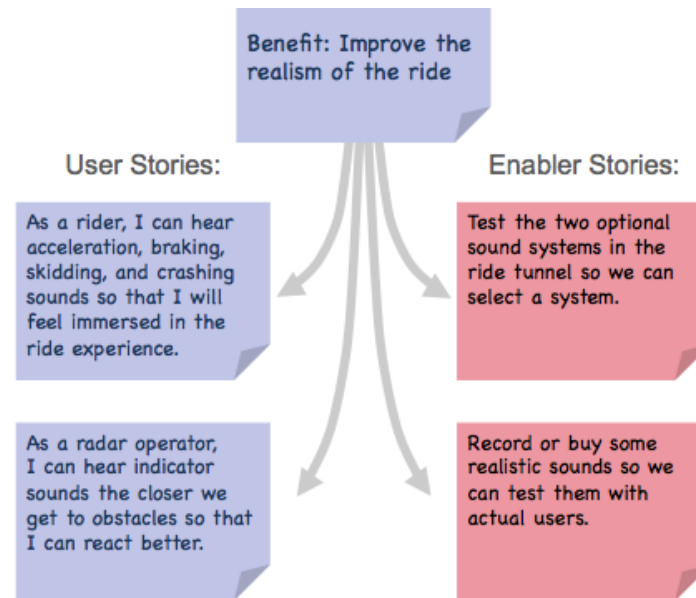


Figure 1. Example of a business feature split into stories

User Stories

User stories are the primary means of expressing needed functionality. They largely replace the traditional requirements specification. (In some cases, however, they serve to understand and develop functionality that is later recorded in such a document in support of compliance, traceability, or other needs.)

User stories are *value centric* in that they focus on the user, not the system, as the subject of interest. In support of this, the recommended form of expression is the “user voice form,” as follows:

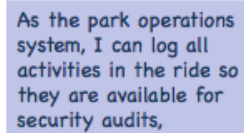
As a <user role> I can <activity> so that <business value>

By using this format, the teams are constantly guided to understand *who* is using the system, *what* specifically they are doing with it, and *why* they are doing it. Applying the user voice routinely tends to increase the team’s domain competence; they come to better understand the real business needs of their user. Figure 2 provides an example:

As a rider, I can hear acceleration, braking, skidding and crashing sounds so that I will feel immersed in the ride experience.

Figure 2. Example user story in user voice form

While the user story voice is the common case, not every system interacts with an end user. Sometimes the “user” is a *device* (example: printer) or other *system* (example: transaction server). In this case, the story can take on the form illustrated in Figure 3.

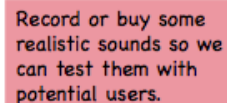


As the park operations system, I can log all activities in the ride so they are available for security audits,

Figure 3. Example of a user story with a system as a user

Enabler Stories

Teams also need to develop technical functionality that is needed to implement a number of different user stories, or support other components of the system. In this case, the story may not directly touch any end user. These are *enabler stories*, and they can support *exploration*, *architecture*, or *infrastructure*, just like all other enablers. In these cases, the story can be expressed in technical rather than user-centric language, as Figure 4 illustrates.



Record or buy some realistic sounds so we can test them with potential users.

Figure 4. Example enabler story

Enabler stories may include any of the following:

- Refactoring (/refactoring/) and Spikes (/spikes/) (as traditionally defined in XP)
- Building or improving development/deployment infrastructure
- Running jobs that require human interaction (example: *Index 1 million web pages*)
- Creating required product or component configurations for different purposes
- Performing special types of system qualities verification (vulnerability testing, etc.)

And, of course, enabler stories are demonstrated just like user stories, typically via showing the artifacts produced or via UI, stub, or mock.

Writing Good Stories

The 3Cs: Card Conversation Confirmation

THE 3Cs: Card, Conversation, Confirmation

Ron Jeffries, one of the inventors of XP, is credited with describing the “3Cs” of a story:

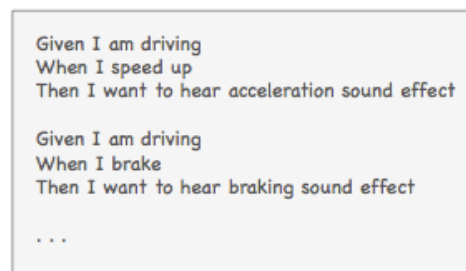
Card represents the capture of the *statement of intent* of the user story on an index card, sticky note, or tool. The use of index cards provides a physical relationship between the team and the story. The card size physically limits the length of the story and, thereby, too-early specificity of system behavior. Cards also help the team “feel” upcoming scope, as there is something materially different about holding 10 cards in one’s hand versus looking at 10 lines on a spreadsheet.

Conversation represents a “promise for a conversation” between the team, Customer (/customer/)/user, the Product Owner (/product-owner/), and other stakeholders. This is the discussion necessary to determine the more detailed behavior required to implement the intent. The conversation may spawn additional specificity in the form of attachments to the user story (mock-up, prototype, spreadsheet, algorithm, timing diagram, etc). The *conversation* spans all steps in the story life cycle:

- Backlog refinement
- Planning
- Implementation
- Demonstration

Conversations provide shared context that cannot be achieved via formal documentation. It drives away requirements ambiguity via concrete examples of functionality. The conversation helps uncover gaps in scenarios and nonfunctional requirements. Some teams also use the confirmation section of the story card to write down what they will demo for the story.

Confirmation the *acceptance criteria* and provides the precision necessary to ensure that the story is implemented correctly and covers the relevant functional and nonfunctional requirements. Figure 5 provides an example.



```
Given I am driving
When I speed up
Then I want to hear acceleration sound effect

Given I am driving
When I brake
Then I want to hear braking sound effect

...
```

Figure 5. Story acceptance criteria

Agile Teams (/agile-teams/) automate acceptance tests wherever possible, often in a business-readable, domain-specific language, thereby creating the “automatically executable specification and test” of the code. Automation also provides the ability to quickly regression-test the system, which enhances Continuous Integration (/continuous-integration/), refactoring, and maintenance.

Invest in Good Stories

People often use the mnemonic INVEST, (<http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/>) developed by Bill Wake (<https://www.industriallogic.com/people/Bill>), to provide a reminder of the things that make a good story:

- **I** – Independent (of all other stories)
- **N** – Negotiable (a flexible statement of intent, not a contract)
- **V** – Valuable (providing a valuable vertical slice to the Customer)
- **E** – Estimable (small and negotiable)
- **S** – Small (fits within an iteration)
- **T** – Testable (understood enough to know how to test it)

Refer to [1] and [2] for more information.

Estimating Stories

SAFe ScrumXP (/scrumxp/) Agile Teams use story points and estimating poker [2 and 3] to estimate their work. A story point is a *singular* number that represents a combination of things:

- Volume – How much is there?
- Complexity – How hard is it?
- Knowledge – What's known?
- Uncertainty – What's not known?

Story points are relative; they are *not* connected to any specific unit of measure. The size (effort) of each story is estimated relative to the smallest story, which is arbitrarily assigned a size of 1. SAFe applies the modified Fibonacci sequence (1, 2, 3, 5, 8, 13, 20, 40, 100) to reflect the inherent uncertainty in estimating, especially large numbers (e.g. 20, 40, 100, etc.) [2].

Estimating Poker

Agile Teams often use “estimating poker,” which combines expert opinion, analogy, and disaggregation for quick but reliable estimates (note that there are a number of other methods used as well). The rules of estimating poker are:

- Participants include all team members
- Each estimator is given a deck of cards with 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞, and ?
- The Product Owner participates but does not estimate
- The Scrum Master (/scrum-master/) participates but does not estimate; an exception is if he or she is doing actual development work
- For each backlog item to be estimated, the Product Owner reads the description of the story
- Questions are asked and answered
- Each estimator privately selects an estimating card representing his or her estimate
- All cards are simultaneously turned over so that all participants can see each estimate
- High and low estimators explain their estimates

...and team estimators explain their estimates.

- After discussion, each estimator reestimates by selecting a card
- The estimates will likely converge; if not, repeat the process

Some amount of preliminary design discussion is appropriate. However, spending too much time on design discussions is often wasted effort. The real value of estimation poker is to come to a common agreement on the scope of a story. And it's also fun!

Velocity

The team's *velocity* for an iteration is equal to the sum of the points for all the stories completed (that have met their Definition of Done) in the iteration. Knowing velocity assists with planning and is a key factor in limiting WIP, as teams don't take on more stories than their prior velocity would allow. Velocity is also used to estimate how long it takes to deliver larger epics, features, capabilities, and enablers, which are also estimated in story points.

Common Starting Baseline for Estimation

In standard Scrum, each team's story point estimating—and the resultant velocity—is a local and independent concern; the fact that a small team might estimate in such a way that they have a velocity of 50 while a larger team has a velocity of 12 is of no concern to anyone.

In SAFe, however, story point velocity must have a common starting baseline, so that estimates for features or epics that require the support of many teams are based on rational economics. In order to achieve this, SAFe teams start down a path on which a story point for one team is roughly the same as a story point for another, so that, with adjustments for economics of location (U.S., Europe, India, China, etc.), work can be estimated and prioritized based on converting story points to cost. After all, there is no way to determine the return on potential investment if there is no comparable "currency."

The method for getting to a common starting baseline for *stories* and *velocity* is as follows:

- For every developer-tester on the team, give the team eight points (adjust for part-timers).
- Subtract one point for every team member vacation day and holiday.
- Find a small story that would take about a half-day to code and a half-day to test and validate. Call it a one (1).
- Estimate every other story relative to that one (1).

Example: Assuming a six (6)-person team composed of three (3) developers, two (2) testers, and one PO, with no vacations or holidays, the estimated initial velocity = $5 * 8$ points = 40 points/iteration. (Note: Adjusting a bit lower may be necessary if one of the developers and testers is also the Scrum Master.)

In this way, story points are somewhat comparable to an ideal developer day, and all teams estimate size of work in a common fashion, so management can thereby fairly quickly estimate the cost for a story point for teams in a specific region. Then they have a meaningful way to figure out the cost estimate for an upcoming feature or epic.

Note: There is no need to recalibrate team estimation or velocity after that point. It is just a common starting baseline.

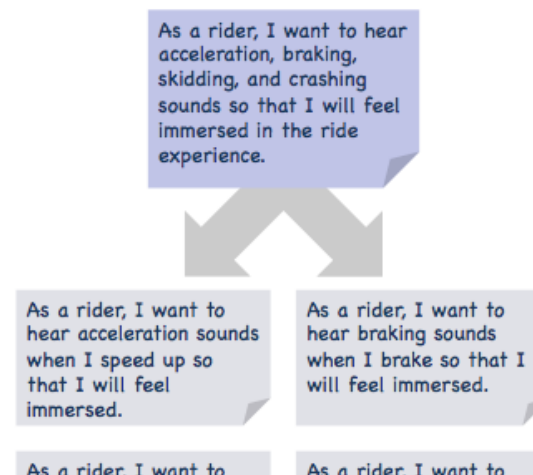
While teams will tend to increase their velocity over time—and that is a good thing—in fact the number tends to remain fairly stable. A team's velocity is far more affected by changing team size and technical context than by productivity changes. And if necessary, financial planners can adjust the cost per story point a bit. Experience shows that this is a minor concern, compared to the wildly differing velocities that teams of comparable size may have if they don't set a common starting baseline. That simply doesn't work at enterprise scale, because decisions can't be based on economics that way.

Splitting Stories

Smaller stories allow for faster, more reliable implementation, since small things go through a system faster, reducing variability and managing risk. Splitting bigger stories into smaller ones is thus a mandatory survival skill for every Agile Team, and it is both the art and the science of incremental development. Ten ways to split stories are highlighted in [1]. A summary of these techniques is included below.

1. Work flow steps
2. Business rule variations
3. Major effort
4. Simple/complex
5. Variations in data
6. Data entry methods
7. Deferred system qualities
8. Operations (example: Create Read Update Delete, or CRUD)
9. Use-case scenarios
10. Break-out spike

Figure 6 illustrates an example of #9, splitting by use-case scenarios.



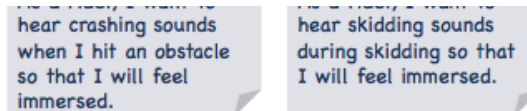


Figure 6. An example of splitting a large story into smaller stories

Stories in the SAFe Requirements Model

As described in the SAFe Requirements Model (/safe-requirements-model/), the framework applies an extensive set of artifacts and relationships to manage the definition and testing of complex systems in a Lean and Agile fashion. Figure 7 illustrates the role of stories in this larger picture.

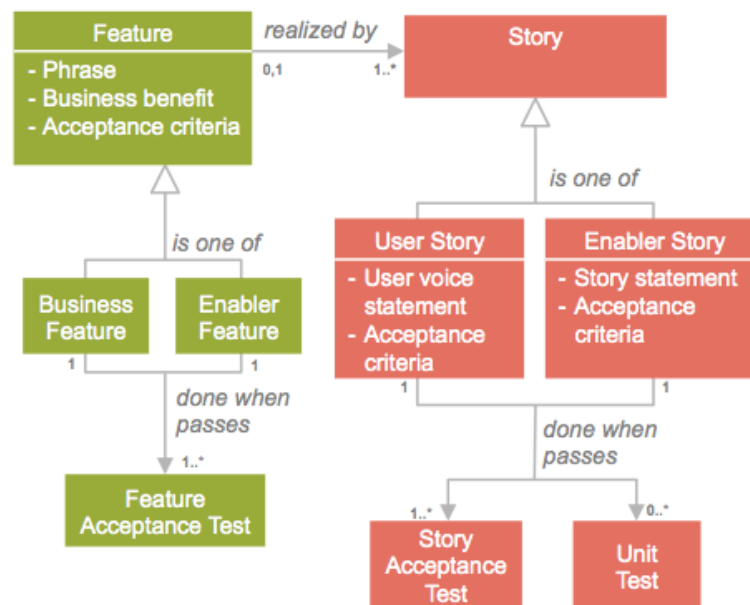


Figure 7. Stories in the SAFe Requirements Model

Figure 7 illustrates how stories are often, but not always, spawned by new features, and how each has an associated story acceptance test. Further, in XP and SAFe ScrumXP, each story should have a unit test associated with it. The unit tests serves primarily to ensure that implementation of the story is correct. In addition, as unit tests are readily able to be automated, this is a critical starting point for test automation, as described in Test-First (/test-first/).

Note: Figure 7. uses UML notation to represent the relationships between the objects: zero to many (0,1), one to many (1..*), one to one (1) and so on.

Learn More

[1] Leffingwell, Dean. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*. Addison-Wesley, 2011, chapter 6.

[2] Cohn, Mike. *User Stories Applied: For Agile Software Development*. Addison-Wesley, 2004.

Last update: 1 November 2015

The information on this page is © 2010-2016 Scaled Agile, Inc. and is protected by US and International copyright laws. Neither images nor text can be copied from this site without the express written permission of the copyright holder. Scaled Agile Framework and SAFe are registered trademarks of Scaled Agile, Inc. Please visit Permissions FAQs (<http://scaledagile.com/permission-faq/>) and contact us (<http://www.scaledagile.com/permissions-form/>) for permissions.

FRAMEWORK

[Downloads \(/posters/\)](#)

[Latest Updates \(/blog/\)](#)

[SAFe 3.0 \(http://v3.scaledagileframework.com\)](http://v3.scaledagileframework.com)

TRAINING

[Course Calendar \(http://www.scaledagileacademy.com/events/event_list.asp\)](http://www.scaledagileacademy.com/events/event_list.asp)

[About Certification \(http://www.scaledagileacademy.com/?page=WhichCertification\)](http://www.scaledagileacademy.com/?page=WhichCertification)

[Become a Trainer \(http://www.scaledagileacademy.com/?page=becomeatrainer\)](http://www.scaledagileacademy.com/?page=becomeatrainer)

PERMISSIONS

[Permission FAQ \(http://www.scaledagile.com/permission-faq/\)](http://www.scaledagile.com/permission-faq/)

[Permissions Form \(http://www.scaledagile.com/permissions-form/\)](http://www.scaledagile.com/permissions-form/)

Usage and Permissions (/usage-and-permissions/)

PARTNER

Becoming a Partner (<http://www.scaledagile.com/become-a-partner/>)

Partner Directory (<http://www.scaledagile.com/listingcategory/directory/>)

Partner Event Calendar (<http://www.scaledagile.com/event-list/>)

GET SOCIAL

Twitter (<https://twitter.com/ScaledAgile>)

Linkedin (<https://www.linkedin.com/grps/Scaled-Agile-Framework-4189072?>)

YouTube (<https://www.youtube.com/user/scaledagile>)

SlideShare (<http://www.slideshare.net/ScaledAgile>)

RECENT POSTS

New Essential SAFe guidance article (/new-essential-safe-guidance-article/)

Nov, 30th 2016

SAFe Website Updates and new Glossary (/safe-website-updates-and-new-glossary/)

Nov, 29th 2016

End of life for SAFe 3.0 website and courseware 12/30/16 (/eol-safe3/)

Nov, 28th 2016

SCALED AGILE, INC

CONTACT US

5480 Valmont Rd., Suite 100

Boulder, CO 80301 USA

Email: support@scaledagile.com (<mailto:support@scaledagile.com>)

Phone: +1.303.554-4367

BUSINESS HOURS