User Stories in Agile Software Development

Kent Beck
Three Rivers Institute
kent@threeriversinstitute.org

David West
New Mexico Highlands University
davidwest@nmhu.edu

Keywords
XP, user story, developer story, story card, discrete story, composite story, collaborative story, dialogue, Agile, system requirements

Summary
A story is just a story, but a user story is key for extreme programming (and agile software development). A user story is distinct from similar techniques (scenarios and use cases) because it is always told from the point of view of the user, not the developer. A user story begins as text-on-a-card but its life extends throughout development as the on-site customer and the developers engage in dialog that confirms, evaluates, expands, and modifies the original story premise. Users can become more proficient at telling stories, and developers in listening to stories, by recognizing some basic story categories.

Applicability
User stories are appropriate, applicable and essential to every type of software application – from embedded safety critical systems to prosaic back office accounting systems. Despite the doubts advanced by traditional software engineers, user stories (and XP and Agile methods in general) are appropriate for projects at any level of scale.

While it may appear that the assertions in the previous paragraph are a bit extreme or extravagant, a bit of reflection brings the recognition: if a system has users, those users must communicate their expectations and needs and they will do so by telling you stories. Even if a user handed you a non-story based document like a flow chart, data flow diagram, or entity relation diagram and said, “build this,” it is unlikely that such a document would be sufficient specification. They would also proceed to tell you stories about what the document really means to them and how they want you to interpret the document.

All software development is a human activity (as yet) and is therefore dependent on individuals and small groups. The organization of those groups and techniques to facilitate communication among members of those groups is paramount to the success of the project. Agile methods, and XP in particular, provide tools and techniques for
maximizing individual and small group success – and user stories are an integral part of those tools and techniques. The fact that very large projects, with geographically dispersed teams, might need additional support mechanisms is a red herring – it does not obviate the need for agility and for communication tools like user stories.

If the term ‘applicability’ is substituting for the term ‘sufficiency’ - i.e., for what types of applications are user stories sufficient? – a slightly different response is required. User stories are not sufficient in any organization where management style and culture mandates more formal documentation or any organization that believes itself subject to externally imposed documentation standards.

Note that it is not the application type that is at issue; it is the culture and belief system of the organization in which development takes place that questions the sufficiency of user stories. If you can develop a system that has no users or where user input is considered irrelevant, you have no need of user stories. This is not a facetious statement. There was a time when our profession developed systems for customers (those paying the bills) and actual users (seldom the customer) were not consulted, simply told, “do it - this way.”

Applicability II – Why User Stories?

Once upon a time (actually just a couple of decades ago) computer were wondrous and wondrously expensive things. The individuals charged with operating these behemoths were akin to priests in a glass walled temple housing “the mainframe.” Business people were but humble petitioners, thankful for anything the technicians deigned to give them. But resentment set in – the technicians “knew nothing of business” and since business was paying for systems the business people should tell the technicians what to do instead of the other way round. The revolt of the users was successful, but only for a time because it seemed that they did not know enough about technical matters to make their desires known. An uneasy truce ensued characterized by a strong emphasis on contracts – rigid and precisely defined requirements strictly adhered to that essentially provide little more than a means for pointing fingers when the development project inevitably failed. This unhappy state persisted “unhappily ever after.”

Power – especially the imbalance of power is one of the root causes of development failure. If business folk have the upper hand, hey demand unreasonable timelines and they don’t eliminate scope, even less important scope, that would enable the whole team to reach those deadlines with quality. If the technical folk get the upper hand, they insist on lengthy processed focused on technical ingenuity instead of business value. What is necessary is a common currency that tends to keep both parties honest and on their toes. Enter the user story.

User stories provide a format that is approachable for business folk while simultaneously containing enough of a reality check to keep client aspirations from spiraling off into the ether.
By starting with a low-tech format (words and sketches on index cards), we lower the barriers for business ownership of the stories. By rapidly estimating the stories and continually refining those estimates, we allow technical feedback to inform and enrich the envisioning process. Physical manipulation and public display of the stories to track progress keeps everyone on the team engaged with the stories.

User stories result in a simple, but not simplistic, meeting ground of “desire and possibility” in Ward Cunningham’s notable phrase.

**Position in Life-Cycle**

User stories are employed throughout the lifecycle. That this might not be the case is due to a misunderstanding of user stories as nothing more than the index card containing a short prose paragraph. (We will dispel this notion in the main body of the chapter.) It must also be remembered that XP/Agile are highly iterative, short cycle, practices that obscure (or eliminate) the kind of “discrete phases” implied for typical life-cycles. However, in some cases the user story takes the form of a metaphor (applied to issues of system specification and system design) or a user test (applied to integration and testing activities). Because metaphors and tests merit their own distinct label in XP/Agile, and only for this reason, we can allow that user stories are more critical in the lifecycle segments indicated in boldface in the following table.

<table>
<thead>
<tr>
<th>Requirements Discovery</th>
<th>Requirements Validation</th>
<th>System Specification</th>
<th>System Design</th>
<th>Coding, First of Class</th>
<th>Integration &amp; Testing</th>
<th>Operations and Maintenance</th>
</tr>
</thead>
</table>

**Key Features**

The key feature of user stories is the *User*.

Unlike use-cases and scenarios, user stories are intended to capture the voice and perspective of the user of the application or system.

User stories are expressed and documented using natural language prose: on the story card, in response to questions, and as part of the ongoing dialog with developers. Use cases and scenarios are expressed using a constrained (semi-formal) syntax. User stories are descriptive and expressive of human desires. Scenarios and use-cases are specifications of object interactions. User stories contain “what” and “why;” use-cases and scenarios contain “how;”

The key feature of user stories is the *User*.

Typical development techniques involve the user but twice – once at project inception and again at project delivery. Project development is based only what could be extracted
from the user during the brief “requirements gathering” window – a window severely restricted in terms of time and tools available. Whatever “story” the user might have told at that point in time is all you have.

XP/Agile require an on-site customer, a user, functioning as an active storyteller throughout project development. Stories are elaborated and extended and modified based on the increasing sophistication and understanding of the user. Oversights and omissions are caught and corrected and the quality of requirements increases dramatically.

**Strengths**

User stories, and the context (XP/Agile) in which they are deployed, eliminate the typical, adversarial, relationship between users and developers. The stories are a source of constant feedback and guidance from the people that will pass ultimate judgment on the success or failure of the project.

Making storytelling “official” allows both the teller and the listener to focus on developing skills in the use of stories. This is important because stories are the primary (perhaps only) means of communicating needs and desires, and providing critical feedback, to developers.

Increasing your story skills makes you a better developer – the real strength of the user story practice is the improvement in the people using the technique rather than something intrinsic to a practice, model, or tool.

**Weaknesses**

“Tell me a story.” It seems too simple to be the basis for software development – especially since we have been telling ourselves, the past thirty years, how hard, complicated, and formalized software development “must” be.

If the two greatest strengths gained from the practice of user stories derive from giving stories official status and allowing developers to develop their human skills; the two most prominent weaknesses derive from negation of the strengths.

Stories are perceived to be too simple and too vague (they use imprecise user vernacular and ambiguous natural language). So even when they are used, there is a tendency to turn them into something they were not intended to be – a “requirement specification” – something more “software engineering-like.”

Storytelling is an art. Software development (traditionally) has no room for art. Storytelling relies on people. Software engineering, as a philosophy and method, seems determined to eliminate both art and people in favor of formalized (ideally automated) process and tools.
In short, the greatest weakness of user stories as a practice is its requirement for managers, users, and developers to exhibit faith in simple things, art, and people to the degree necessary for the practice to prove its effectiveness.

**Technique**

**Story:** “one thing that the customer wants the system to do.”

This definition by Beck [2000] is but a starting point. Once imagined, a story must be recorded, clarified, sometimes factored, developed and extended, and utilized in a series of other activities.

The user story technique is straightforward:

- The customer (user) thinks of something she wants the system to do.
- The desire is written down on an index card; given a name and a number.
- An estimate is made of how long it will take to realize the story in fully functional and releasable form.
- Stories are factored – split into smaller stories – if it appears they will take too long to implement as written.
- Stories are factored if one aspect of the story is more important than others.
- Stories are factored if they are long and rambling or overly general.
- Stories are prioritized.
- Stories are aggregated into collections, each collection defining the scope of work to be undertaken by the team this period.
- Work products are validated. Their ability to satisfy the original story is confirmed.
- The customer uses the developed system and new stories are conceived.
- Iterate.

Even the simplest technique has nuances that must be recognized, if one is to master that technique. By exploring the activities in the preceding bullet list, some heuristics can be provided that will enhance everyone’s use of stories

**Users: Tellers of Stories.**

Users tell ALL stories. Developers may ask for a story if they think they have an idea about something the system might do or if they need clarification of some aspect of the system – but a user must actually write the story.

Stories are told (written) using natural language. The only vernacular or specialized vocabulary allowed in a story is “business” language. Almost every domain of human activity develops its own specialized linguistic extensions: special terms, acronyms, even colloquialisms and redefinitions of common words; and all of these may be employed in a story. Software developers have their own specialized jargon and, unfortunately, we have coerced users into using some of that jargon in their communications with us. But
no more! Developer jargon is not to be used in stories. This means that a story, “the system will adjust COLA rates for all exempt employees,” is OK; but, a story, “the UI will include a check box for COLA Y/N and a widget showing new COLA value if checked,” is not OK.

A story card – a simple index card – is used to record the story. Using an index card is a heuristic that leads to good story telling. There are fairly obvious limits to the amount of prose that can fit on an index card, which encourages brevity and clarity of expression. Story cards typically include a title for the story and a number. The title provides a handy reference label for the story and the numbers can be used to aggregate stories, e.g. stories factored from the same general story.

Stories begin, but do not end with, the story card. A story card “… is nothing more than an agreement that the customer and the developers will talk together about a feature.” [Beck and Fowler 2001] The XP practice – on site customer – might very well be renamed on-site storyteller. Developers engage in a continuing dialog with the customer; asking questions, restating their understanding to gain verification, and listening as the customer elaborates the original story. Elaboration may take the form of added detail or it may reveal additional stories that must be captured on a story card of their own.

Developers are not totally passive. Although they are not allowed to write stories they can suggest new stories to the customer who may find the idea worthwhile and write an appropriate story. Developers are also providing constant feedback to the customer. Feedback definitely involves estimates – the developer telling the customer that this story will take X amount of time to complete – which the customer can use to judge whether or not his stories are appropriately sized. Developers also provide feedback about the “quality” of the stories, how easy or difficult they are to listen to and understand. This kind of feedback improves the customers ability as a story teller.

“Don’t be too impressed with your stories. Writing the stories is not the point. Communicating is the point.” [Beck and Fowler 2001] A story is the entire conversation that takes place among customers, developers, and even managers.

A story includes the contributions of all parties – not just the customer and not just that which is written down on the story card. Keith Basso is an anthropologist who studied Native American stories. He noted that published transcriptions of stories included only the narrative of the “storyteller,” ignoring the responses and comments made by the audience. This omission inevitably distorted the story, sometimes inverting the actual point and meaning of the story.

If you want to be successful with user stories you must constantly remind yourself – until it is unspoken second nature – that the story card is not the story. The story is the entire conversation, only parts of which are recorded. And the recorded parts might be distributed in code, in comments, on story cards, in diagrams on a white board, or in informal notes. In fact, the only real persistent “documentation” of a story (the cards themselves can be discarded when completed) is in the code and the acceptance tests.
Story Characteristics and Typography.

Telling stories is an art. People can develop their art and become better storytellers. It is also possible to become a better listener or participant in the conversation that is the true story. It is not possible to write a cookbook, “Great User Stories in Just 30 Minutes.” [Beck and Fowler 2001] Instead, feedback and practice are required to develop your story telling (and story writing) skills. It is also possible to provide some heuristic guidance about stories that help get you started.

All stories must be:

- **Discrete.** They describe a single bit of functionality, a single feature, or an expectation the customer has of the system. Discrete does not mean precise, a story can be general in nature (at least for a while), e.g. “calculate the Payroll.” The more general a story is, the more likely it will be factored into numerous stories with narrower scope. But do not hesitate to start with a general – yet discrete – story, they make excellent starting points.

- **Estimable.** The development team must be able to read the story and estimate the amount of time required to realize that story in production quality software. How the developers make those estimates is not of major concern here. That they provide feedback to the customer (storyteller) as to what those estimates are and how uncertainties in the story or the scope of the story contributed to the estimate’s value (duration) is critical.

- **Testable.** How do you know that, “they lived happily ever after?” How long is ‘ever after?” What constitutes happiness? Where they happy 24/7 or just most of the time? Defining tests is a joint responsibility, shared by the customer and the developers. Developers must define the more technical tests and can provide feedback to the customer that will enable her to define acceptance and usability tests.

- **Prioritized.** The customer, and only the customer, must be able to determine which stories are more important, more likely to generate an immediate business benefit, or otherwise more valuable to the customer (business). Story cards record the priority and developers work on the pending stories with the highest priority.

“Tell me story about what you want the system to do,” asks the developer, totally intimidating the customer. There are few things as frightening in life as a blank sheet of paper and the request to fill that paper with some meaningful communication. We can help our customer get started by suggesting some categories or types of story and providing some examples. We cannot write the story, he must do that, but we can suggest story topics or types. Examples of such types include: interrogatory, delegation, composite, collaborative, and fuzzy.
“Are there questions you would like the system to answer?” A lot of stories reflect the result of an implicitly asked question. The heuristic to the customer is: “think of a question (an interrogatory) you would like the system to answer and then describe that answer.” Examples might include: [Story in Arial font, implicit question in italics.]

Sound an alert if a “near miss” is detected. (Are any planes in imminent danger of collision?)

Disable any bin containing product that has exceeded its expiration date. (Is that candy bar edible?)

Provide a count of all employees in each salary category. (What happens if we give everyone a sliding percentage raise, greatest percent to the lowest salary categories?)

Sometimes a customer just wants the system to perform some specific piece of work that is too tedious, complicated, or slow if assigned to a human being. Delegation stories tend to be fairly terse but they also tend to hide some interesting complication. Examples might include:

Assign the customer a credit limit.

Calculate and print an amortization schedule

Composite stories are general stories, “calculate this week’s payroll,” that we are fairly certain will be broken up into smaller stories but we record as starting points for our thinking about the system. An example:

Fill in and validate the form. (The “and” clause is a dead giveaway that this story is composite in nature – at least one story to fill in the form and another for performing the validation.)

In some cases the customer can visualize an interaction with the system focused on accomplishing a specific task. (Filling in a form might be require an collaborative effort, the system asking for content, the user providing same, and the system providing a dialog for correction if the content is unsatisfactory.) An example of this type of collaborative story might be:

Allow a reservation to be made from a list of available rooms based on the selection criteria entered. (Story requires a dialog where options are presented, selections are made, information is sorted and sifted and then presented, another selection made, and reservation completed.)

Both composite and collaborative stories are subject to factoring – being split into simpler stories. The original composite and collaborative story card might actually be
discarded once the team (customer and developers) is satisfied that they have identified all the subsidiary stories required to realize the original.

A final type, fuzzy, of story captures a customer desire that will not be fully explicable without a lot of dialog and probably a lot of development and feedback. It is important to capture this type of story as a reminder of important functionality even though everyone concerned knows the full parameters of the story will not be worked out for several development iteration cycles. An example of a fuzzy story:

Provide context sensitive help. (But not that &*$%#$ paperclip!)

Story characteristics and story types help the customer get started. Skill in story telling must be acquired through experience. Developers provide the feedback necessary for the customer to improve her story art. Stories should therefore be developed in small batches, perhaps focused on a single aspect of the system, so that effort is not wasted creating poorly conceived story cards.

Splitting Stories

There are numerous reasons for breaking a story up into smaller and more focused stories. Sometimes the story was intentionally general or intentionally composite in nature, just to get a starting point. Sometimes a story takes too long to implement (at least it is estimated that it will take too long). Sometimes the telling of a story – the dialog that occurs between developers and customers as tests and code are being written – reveal an unforeseen complexity that needs to be “spun off” into a separate story. Talking about a story might reveal an exception and exception handling requires a different story.

Stories might be split as a result of the assignment of all or part of the story to specific objects for implementation. Splitting stories for this reason requires a bit of background – XP/Agile assumes you are going to be doing object-oriented design and object-oriented programming. Stories that are specific enough in nature will become methods assigned to a particular class. Other stories might be implemented as a set of methods given to a single class. More likely, the individual tasks implied by a single story will be assigned to several classes, instances of which will collectively implement the story. This kind of story splitting may not be documented with a story card – merely as a test or a bit of code or a comment attached to code.

Ordering and Completing Stories

Stories are used as a unit-of-work, i.e., the team plans their work activity on the basis of completing a story and completing a set of stories in a specified calendar interval. The
Planning Game (XP) is focused on deciding which stories the development team will complete during this time interval. If the Planning Game is to succeed, the time and resources required to complete a story must be estimated and attached to the story. The developers provide this estimate. The customer(s) must provide a priority for each story. The customer supplied priority must reflect business priority – it must be based on business value added and nothing else. Deciding on a schedule and a set of deliverables is as simple as arranging the cards in priority order and, using the estimates, select story cards until the estimates fill the time interval available.

Of course, it is not quite this simple – there will be ties in priority, actual time may not equal estimate time, there may be more #1 priorities than can be accommodated in the given interval, etc. Negotiations take place among the team and the customers and conflicts resolved. Like everything else in XP/Agile, planning is an iterative process that improves with experience.

Stories continue to be told during planning. When a customer explains why she is rating this story card number one she will be elaborating the text that appears on the story card, elaborations that might merit a quick Post-It note attached to the card to further guide the developers. A story that has borderline priority might be split into parts – one with higher priority – to assure that some aspect of the story is developed this interval. A meta-story (a story about how the stories on cards are related or interact) might be told as a foundation for aggregating stories that are not, strictly speaking, highest priority but nevertheless need to be developed in concert.

The plan itself is a story. A story that captures the collective understanding of what we are trying to accomplish as a team (the team is inclusive of customers, managers, and developers). It probably includes our understanding of how we will react when things do not go exactly as planned. It is a story that captures our sense of self and of community. Planning documents are evocative symbols of this story – not documentation or representations of it.

Confirming Stories

A test is a story recorded in a different format. Both novels and poems tell stories, but the format is quite different – so to with story cards and tests. Story cards and tests share a sequential relationship – the story card is the first chapter and the test is the final chapter. The story card says, “this is what I want,” and the test says, “this is how I know I got what I wanted.”

Tests can confirm the original story or they can reveal its shortcomings. King Midas wanted everything he touched turned to gold. The test cases - bricks, apples, and his beloved daughter – revealed a flaw in the story, not in the implementation of the story. Similar things will happen with software development stories and their test cases.

Humans cannot seem to tell stories without embellishment. Anyone that has played the game where a short story is passed from person to person around a circle knows how
changed the story becomes by the time it returns to its origin. Development necessarily
involves passing a story from person to person within the development team. Tests help
assure story consistency by providing an objective (it is hard to think of automated test
suites as being anything other than objective since they are implemented on machines)
check on story consistency.

Hierarchical organization of test suites (unit tests done by programmers to acceptance and
usability tests created by customers) also provides a means for keeping independent story
lines in concert.

Communication – The Never Ending Story

Descriptions of the user story technique are inevitably misleading. The easiest things to
talk about are the least important aspects of the technique. Story cards, tests, priorities,
and estimates are but artifacts that arise from story telling.

An archeologist investigating a prehistoric culture finds artifacts (potsherds, arrowheads,
bones, and other items) that are evidence of human existence. The artifacts themselves
are trivial. The archeologist and other anthropologists can use those artifacts to
reconstruct what life and culture were like at the time the artifacts were created and used.
It is the culture, the living interaction of human beings in a community, that is critical –
both for those living it at that time, and for our understanding of our own culture.

User stories, story cards, estimates, priorities, tests are byproducts of a culture of
communication. They provide some evidence that communication is taking place but do
not in and of themselves assure that communication (or enough communication) is taking
place. It is possible to generate every artifact discussed in this chapter and still fail to
successfully implement the user story technique.

There is but one story – it begins the moment someone with authority says, “lets do X,”
and ends only when the project has been abandoned and everyone involved has moved
on to other things. Everyone contributes to the story. There is a narrator, the customer,
that outlines the plot and limns the action scenes but she is not the sole speaker. The
story exists in the memories of those involved in its telling. The artifacts generated by
the occasional reduction to paper (or binary integers) of parts of the story serve as
reminders, evoking the memories, refreshing the story.

Worked Example

Arriving at work one morning you are called to a meeting. Margaret, the CEO of the
company, tells you that Bob, the other person in the meeting, is heading up a new
division charged with designing, building, and deploying a new line of superior vending
machines – the Extreme Performance Vending Machine or XPVM. You will be expected
to write the software for this new line of vending machines and Bob will be your on-site
customer.
You sit down with Bob and give him a quick briefing on user stories and the XP/Agile process and ask him to write the first set of stories that will get the team thinking about the project.

“I guess we already have our first story, build a better vending machine,” you suggest to Bob who likes the idea and writes on the first index card: 0.0 – Mission - *XPVM will outperform all other vending machines.* You have your first story.

“Can you give me a story that tells me why the XPVM is better,” you ask? Bob quickly writes down some additional stories. (Bob has done this before so he is not a total novice at story telling.)

1.0 – Payment. *XPVM will accept any kind of payment including cell phone transactions via the Web.*

2.0 – Freshness. *XPVM will sell no merchandise that is stale or outdated.*

3.0 – Restocking. *XPVM will automatically request restocking with the best selling items in its area.*

4.0 – Communication. *XPVM will communicate with the customer to prevent transaction errors.*

You have five stories, enough to get started. Unfortunately, you cannot see any way to estimate the time required to realize any of these stories. Splitting is required and probably of only one of these stories. So you ask, “Which of these stories is most important to you Bob?”

“Well, Margaret just came back for Scandinavia and was very impressed when her friends bought sodas from a vending machine using their cell phone. She thinks, and I agree, that the single biggest problem with vending machines is having the right change to make your purchase. So … the accept all forms of payment story is most important.”

“OK, so how would you split up that story?”

Bob grabs some more cards and records the following:

1.1 - Accept coins.

1.2 - Accept currency.

1.3 - Accept debit card.

1.4 - Accept credit card.
1.5 - Accept debit or credit card via a Web transaction.

1.6 - Accept foreign coins and currency, at least Euros since we want to sell the XPVM in Europe.

1.7 - Convert currencies appropriately.

1.8 - Make sure the payment meets or exceeds the cost of the product selected.

1.9 - Make change.

All of these stories are discrete, estimable and testable – except, the ones about accepting and converting foreign money, those you are not sure about. Stories 1.6, 1.7, and 1.8 are estimated to take a lot of time, perhaps exceeding the 4 weeks cap on story implementation. (4 weeks is an arbitrary (mostly) shop standard used for work planning purposes.) 1.8 will take longer because it involves an interface with other parts of the vending machine system and you are not sure what those parts look like as yet.

The developers suggest that Stories 1.1 and 1.2 might be combined as they seem to be essentially identical. The customer accepts their suggestion and throws out 1.2 and changes 1.1 to, Accept U.S. legal tender.

A planning session convenes and the customer gives 1.5 and 1.6 the highest priority followed by 1.8 and 1.7, the other stories have the same priority. Discussion ensues. Bob indicates his highest priority selections reflect the CEO’s excitement when she was able to use her cell phone to buy products from vending machines in Sweden. He wants to keep her happy and feed on her enthusiasm. The developers point out that the highest priority stories presume the existence of the lowest priorities – you can’t do any kind of transaction unless you can accept money and make change. The developers also have a vague sense of unease about the possible complexities involved in debit and credit card transactions.

It is decided to take one day to “spike” stories 1.3, 1.4, and 1.5; to rough out some tests and some code to see if they understand the problem and understand the dependencies on the stories about accepting payment and making change. This way they will be addressing the customer’s highest priorities and their own concerns simultaneously.

The next morning the planning session resumes. Developers report:

- It appears that some debit cards can be read directly and others require a query to the issuing bank to ascertain funds available.
- Credit cards usually, but not always, depending on the amount, require an authorization transaction.
- For debit cards and credit cards, identity of the user needs to be confirmed.
• Web transactions are really just a sequence of the other transactions but the sequence needs to be maintained.
• For Web transactions it is probably a good idea to confirm that the user is talking with the vending machine in front of him and not one across town.
• Talking about debit and credit cards made them realize that currency transaction require an accumulator of some sort – some way to get total amount of money available instead of coin by coin.
• Based on their improved understanding of the stories the developers have some tests that they can offer programmers working on the accept money and make change stories the reflect assumptions they are making as they develop their own stories. Those tests can be appended to the story card as annotations for teams working on them.

Bob, listening to the developers quickly writes some additional stories: 1.10 – Confirm Customer Identity, 1.11 – Report Available Money, 1.3.1 – Authorize Debit Transaction, 1.4.1 – Authorize Credit Transaction, 1.5.1 – Confirm Web Conversation, and 1.5.2 Execute Web Transaction.

Programming teams select cards and development commences. Iteration follows.

Comparisons

Most readers will discern a strong similarity between XP story cards and the CRC (Class-Responsibility-Collaborator) cards invented by Ward Cunningham and Kent Beck. Both are simple index cards with simple entries. Most importantly, on both types of card, the entries are intended to describe expected behavior, something(s) that is expected of the software artifact to be constructed. Behavior is defined in terms of the domain and of the inhabitants of a domain – the users.

This emphasis on behavior – domain-centric behavior – is a primary difference between user stories and use cases or scenarios as discussed by Achour, Rolland, Maiden, and Holtzblatt earlier in this section of the book. [[Editor – please insert cross-reference in your desired format, degree of specificity.]]

Although nothing mandates this to be so, both use cases and scenarios tend to be used to describe the solution space (implementation space, i.e. the software) rather than the problem space (user domain). Use cases were invented by Ivar Jacobson and reflect his experience and understanding of software engineering. Scenarios have a more varied history but they too reflect their invention by software professionals. Both tools reflect the software professional’s desire to have user requirements stated in a more formal manner. To participate in the construction of use cases and scenarios, users must adopt, or adapt to, the mindset of the developer, to think in terms of formal interactions, messages, and objects – all artifacts of the technical implementation.

User stories are deliberately intended to provide more of a metaphor than of a specification. Use-cases and scenarios are generally intended to provide specification
and therefore must be more formal in nature than a story. A specification is something to be satisfied – it is a contract. A metaphor is a framework for exploration, discovery, and conversation. XP in general is more concerned with iterative exploration and the use of metaphor as a framework for that exploration – a reflection of the values underpinning XP. User stories are valued as metaphors. Specifications of the sort arising from use cases and scenarios are suspect, not because they are “bad” or “wrong” but simply because they are subject to abuse. They lead too easily towards an assumption of need for upfront design or overemphasis of the intrinsic value of documentation.

As noted earlier in this chapter (Applicabiity II – Why User Stories) there is a delicate balance of power that needs to be maintained between customers and developers. Both use cases and scenarios tend to tip this balance of power towards the developers. (Although I must stress again this is not intrinsic to either tool, just to the way they are generally used.) User stories allow the customer to articulate their concerns and desires in their language – natural language, business language.

As markers or checkpoints used for common and consistent reference as the conversation between customer and developer continues, user stories allow both sides to do a better job of translating from “what” is wanted to “how” it will be delivered.

References

The entire XP Series published by Addison-Wesley with Kent Beck as series editor provide valuable background on what users stories are and how they are used. Of special note in this series:


Recommended Reading

Lakoff, George and Mark Johnson. *Philosophy in the Flesh: The Embodied Mind and Its Challenge to Western Thought*, Basic Books. 1998. (For ideas on the power of metaphor and story telling to guide thinking.)

Weinberg, Gerald. *The Psychology of Computer Programming*. Dorsett House. 1998. (For ideas on how stories are used in human interactions endemic to software development.)
Wood, Jane and Denise Silver. Joint Application Development, 2nd ed. John Wiley and Sons. 1995. (For ideas on story telling in a group context and the use of stories when development and design responsibilities are shared with all stakeholders.)

1 Each of the XP/Agile conferences attended by the authors the past two years has featured at least one session dealing with mandated process and documentation issues (e.g. military or FDA requirements). In each case the experts from the mandating agencies indicated that the process and documentation generated in an XP/Agile process can satisfy their agency’s requirements. An yet, compliance officers in companies continue to express contrary beliefs.

2 One of the best tools available for requirements gathering (both direct and indirect) is an ethnography – a period of participant observation. This tool is seldom used because it takes “too much time” and because most requirements gatherers lack adequate observation skills.

3 Roger C. Schank wrote a wonderful book with this title – *Tell Me a Story: Narrative and Intelligence*, Northwestern University Press, 1995. The topic of that book was artificial intelligence and modeling that intelligence with hierarchically nested stories and their variants. The power of stories to capture arbitrarily complex specifications is evident in his treatment.

4 When writing about objects, Cunningham and Beck used the term “behavior” to describe a discrete unit of object functionality. Stories and behaviors are almost synonymous, and anyone wanting to master story writing would be well served by investigating what Beck and Cunningham had to say about objects.

5 There is an epilogue, a chapter after the final one, which covers the dialog that ensues once the new system is deployed. The epilogue is highly reflective and focused on lessons learned, lessons that will become the initial stories for the next system to be developed.

6 XP/Agile practitioners believe that a project continues as long as someone is still using the system and someone is still “maintaining” it. Deployment and maintenance are not outside the project, merely a different aspect of the project.

7 The term spike comes from XP and merely means a short intense period of exploration using test and code tools. The objective is to ask better questions, confirm understanding, and outline a framework for future story implementation.