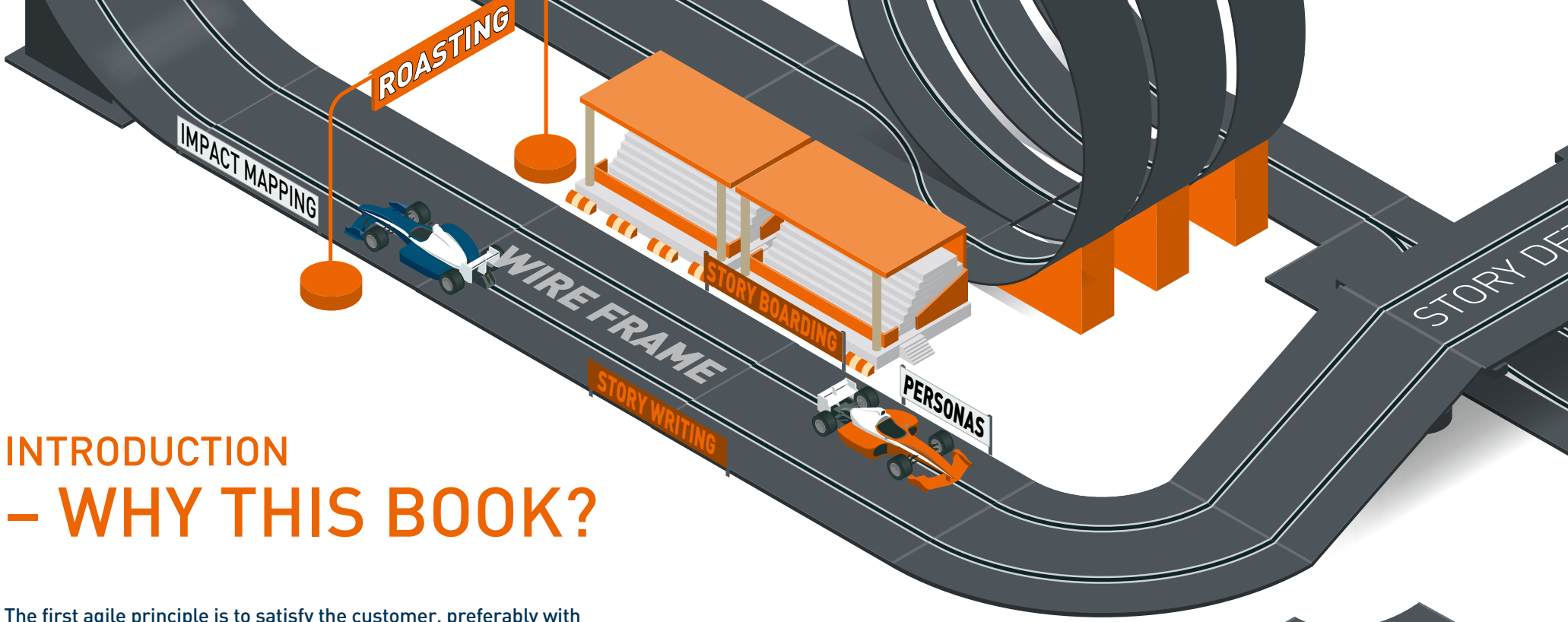
SOFTHOUSE

AGILE REQUIREMENTS

in 60 minutes

Bea Düring &
Håkan Kleijn

EUREKA ITEA 3 SCALARE



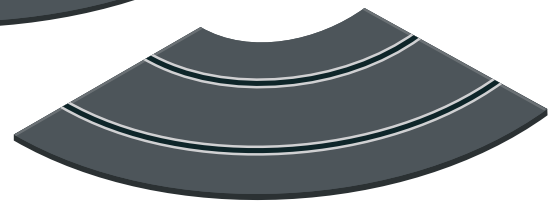
INTRODUCTION – WHY THIS BOOK?

The first agile principle is to satisfy the customer, preferably with frequent deliveries. The shorter the time box, the faster the feedback loop, and, the more opportunities for adjustments.

The software development process starts with exploring needs through collaborative requirements gathering. We are not done refining needs until we validate our exploration with examples and acceptance tests. Only then do we have actionable user stories or product backlog items that can be pulled into actual sprints or iterations.

As Agile coaches we have seen the bottlenecks that occur when businesses fail to understand the need of accelerating requirement work in parallel with the accelerated development flow. Unproductive sprint planning sessions, starting and stopping story development during the sprint, not being done during the iteration and the "boomerang" effect of returning stories over and over again. These are some examples of the penalties inflicted by not paying the requirement work its due.

We have battled these bottlenecks by mixing various agile requirement related methods and experienced the acceleration of the overall value chain of delivery. We have also noticed how collaboratively analyzing a demand from different perspectives intensifies learning and moves unnecessary work to the last responsible moment.

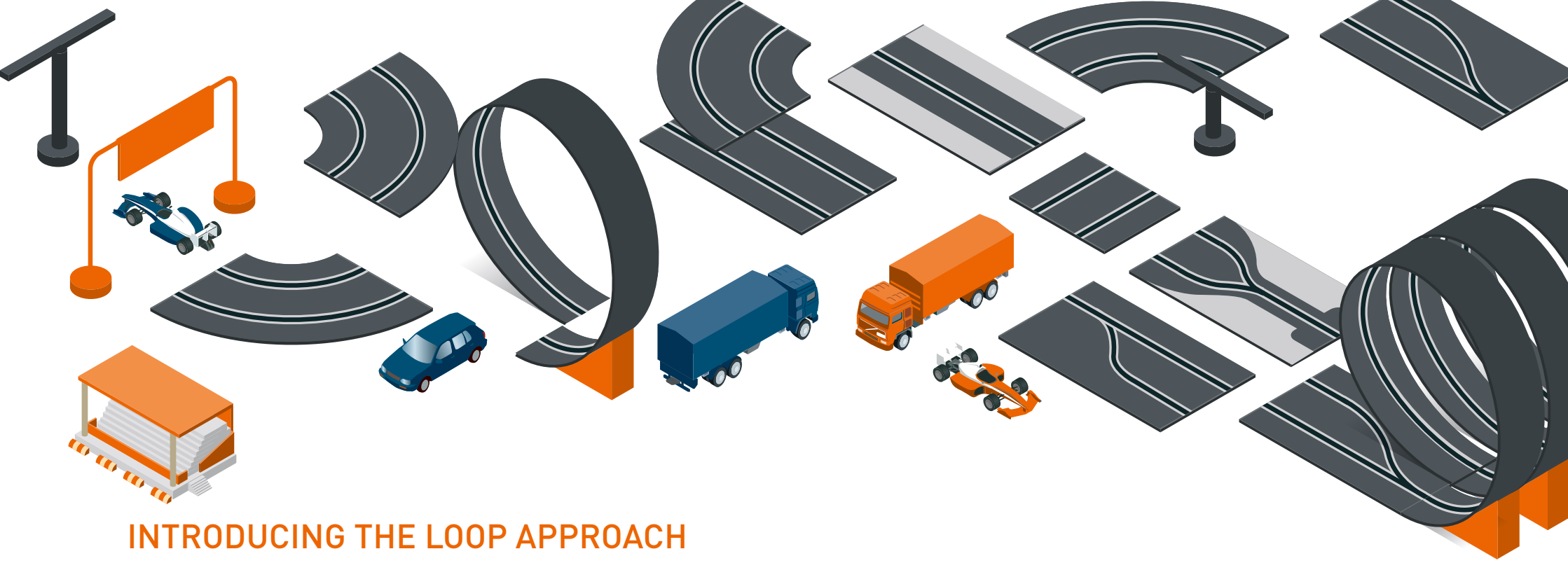


Adopting cross-functional collaboration and just-in-time requirements is the key to becoming agile. A lot of businesses going agile still live under the misconception that requirement work is part of a waterfall model they are trying to move away from. Agile requirement and its extensive toolbox is the answer. The intent of the agile requirement molecule, the user story, is to foster collaboration; the perspective is on delivering the right customer value with as much precision and discipline as possible in a just-in-time manner.

This book is the second in a series and is a more detailed continuation of our first book, Agile Inception, which describes different method combinations to nail initiatives. In 'Agile Requirements in 60 minutes' we focus more in-depth on the requirement work and content rather than on the business scoping side of the techniques which we explored in 'Agile Inception in 60 minutes'.

We really hope this book will make you FAIL*, since failing is what it takes to learn new stuff. *(First Attempt In Learning)

We welcome questions and feedback.



INTRODUCING THE LOOP APPROACH

This book is about release planning, which is briefly covered in Scrum but an essential part of eXtreme Programming and other ancestors. Release planning means identifying features, splitting them into tangible user stories, mapping stories into a look-ahead plan, and finally, from that story map, slicing out the most valuable stories to become the first release.

Please note that we are not promoting the full up front breakdown of all stories to the same level of detail, rather the opposite. We also want to be clear that the requirement related work of release planning needs to be executed in a cross-functional and collaborative work mode.

Discussions around requirements (and testing) tend to be a very complicated tangle of perspectives, methods and buzzwords; business, user and software requirements. Acceptance test driven development, behavior driven development, specification by example and (user) acceptance testing – you name it. And more to the point, how is this done in an agile context?

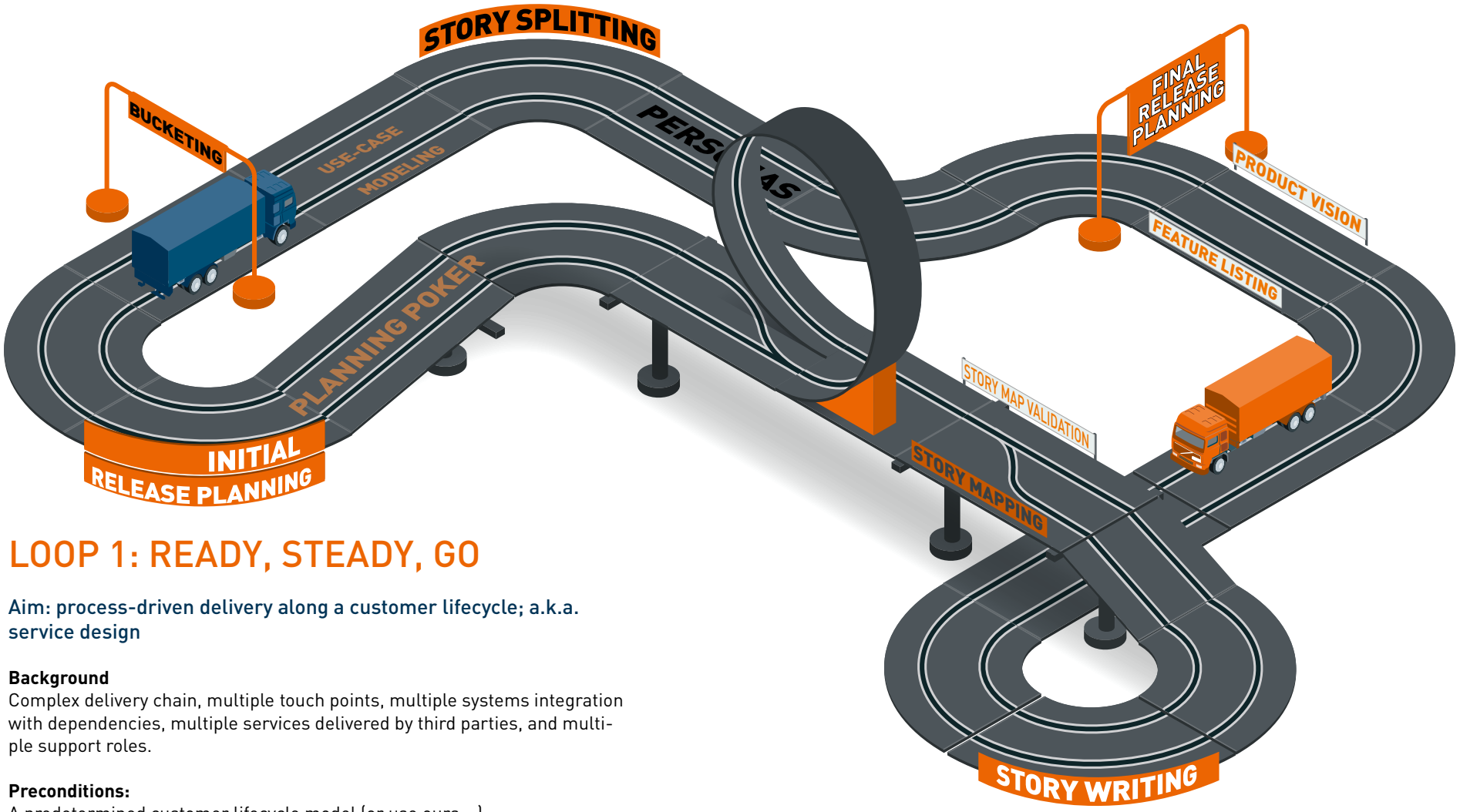
When introducing the Loop Approach in the following pages, we share our experiences of how successful methods blend when used under different conditions, depending on how much exploration is needed up front as input to the release planning.

All three agile requirement loops represent strategies or actual recipes that can be used in a large scale, multi-team agile environment. In fact, that is the primary context in which we, as Agile coaches, have been exploring the techniques and combinations described in the loops. They can also be used in single team environments of course.

As always, you have to do your homework and contextualize these ideas to fit your specific business domain, your mode of development and your requirement stakeholders. We offer three different modes of execution to make it simpler for you to find one that fits your needs. We would also encourage businesses that have not yet gone agile to read the book and see how you can accelerate and hopefully improve your requirement work.

In the end we all have to pay the requirement piper and the 3 Agile Requirement Loops boil down to this:

Ready-Steady-Go GO! Go & Do!



LOOP 1: READY, STEADY, GO

Aim: process-driven delivery along a customer lifecycle; a.k.a. service design

Background
Complex delivery chain, multiple touch points, multiple systems integration with dependencies, multiple services delivered by third parties, and multiple support roles.

Preconditions:
A predetermined customer lifecycle model (or use ours ;-)

Example of fit:
As a large organization delivering products/services via customer lifecycle processes
We want a faster but disciplined release planning approach
So that we make quicker investment decisions regarding delivering the right customer value

Acceptance Criteria:
- Verify that you can execute the loop with cross-functional collaboration between business stakeholders, users and development teams
- Verify that you have a validated product vision
- Verify that the customer lifecycle is the foundation for feature identification
- Verify that features are decomposed into use case diagrams
- Verify that your customer segment has been validated with personas
- Verify that your high-level stories have been prioritized for a 1st minimum release
- Verify that your final release plan covers both smaller, testable user stories for initial iteration and high-level stories for upcoming iterations
- Verify that the release plan can be used for initial cost and time estimates

Time box:
10 working days

PRODUCT VISION

This first loop starts with product visioning, which is a good starting point from a requirements perspective. From an agile requirement perspective we need a good product-centric starting point that clearly states the key capabilities and how they fit the target customer group needs. We need to clear away "business and project debris" and achieve consensus on the "what". Product visioning techniques are very useful to create precision on the business level of the overall requirement.

Usually a larger effort of planned work stays in a "concept state" while product management and/or chief product owners sort out delivery prioritization between this specific effort and all other efforts described in, for example, a product roadmap. "Concept state" at this point usually means 1–5 powerpoint slides that describe some overall need from a business perspective and the justifications for this specific need and effort.

The product vision step is part of the planning and collaborative practices of XP, eXtreme Programming, to initiate product effort identification. Typical for XP is the playful, collaborative and accelerated nature of the vision activities. Here are two favorites:

THE ELEVATOR PITCH

Imagine a time slot of 30 seconds to sell and/or inform someone about your product. Which information is essential to deliver a clear message? The elevator pitch is a good way to take the "concept state" and turn it into a distinct summary of the product effort you are about to start work on. If the "concept state" is not able to answer the key questions enforced in the elevator pitch format, or if different business stakeholders have distinctly different interpretations, then we fail quickly and business stakeholders are shipped back to explore and answer the questions raised in the elevator pitch session. It is a very useful validation step and a good starting point for the requirement work.

We collaborate in smaller teams, 3–4 individuals, to ease cooperation and to get multiple alternatives before narrowing them down and merging into one end-result.

1. The elevator pitch sentence structure is written on a whiteboard, or flip chart, with breaks for the keywords, e.g., target customer.

2. The key words are written on post-it notes to ease duplication and replacement.

3. The exercise runs in time boxes of 15–20 minutes, each ending with a demo. We show each other to generate new ideas.

4. The last time box focuses on narrowing down alternatives into one end-result. We use dot-voting or fist of five to facilitate collaborative decision making.

Elevator Pitch sentence structure:

FOR (target customer) , WHO HAS

(customer need) , (product name)

IS A (market category) THAT

(one key benefit) UNLIKE (competition)

THE PRODUCT (unique differentiator)

Elevator pitch example:

FOR *a multi-site vpn customer,*
WHO HAS *a need for a movable high-speed access point,*
MovIT ***IS A*** *virtual private network service*
THAT *provides 4G access with quick launch and backup capabilities*
UNLIKE *competing vpn services*
MovIT *has one contract, one bill and one help desk*

A brief word of advice – if your product effort is an enhancement of an already existing product and you find it difficult to differentiate from competitors, you can compare the benefits against the previous version of your product. Sometimes you may also need to refer to a well-known quality product with LIKE product x, our product etc.

THE PIXAR PITCH

The Pixar pitch is about using storytelling with a fixed format to nail down the key narrative of a film and thus answer why it will be a compelling experience to watch. When used for product centric purposes as a product vision technique the "event" is used to identify the trigger blocker that a user encounters that prompts the need for change. These problem areas are highlighted in the pitch and can be countered in the "because of" sections where key new capabilities (high level features) can be stated and how the target group problems are eliminated. A great technique to identify elements of "Who" as well as "Why" plus key "What" aspects in a manner that supports cross-functional collaboration. You can organize the work on the Pixar pitch in a similar way to the Elevator Pitch with break out groups and similar time boxes.

The story spine	Structure	Function
Once upon a time ...	Beginning	The world of the story is introduced and the main character's routine is established.
Every day ...		
But, one day ...	The event	The main character breaks the routine.
Because of that ...	Middle	There are dire consequences for having broken the routine. It is unclear if the main character will come out alright in the end.
Because of that ...		
Because of that ...		
Until finally ...	The climax	The main character embarks upon success or failure.
And, ever since then ...	End	The main character succeeds or fails, and a new routine is established.

Published at aerogrammestudio.com, © Kenn Adams.

The product vision work, for example the elevator pitch, gives us an initial understanding; what we should do for whom and why.

By now we have a value and product-centric starting point for our requirement work. We can take our understanding of the statement of need and start exploring features needed from a broader product planning perspective. Let us continue to look for features along our customer lifecycle model that can be tied back to the statement of need in our product vision.

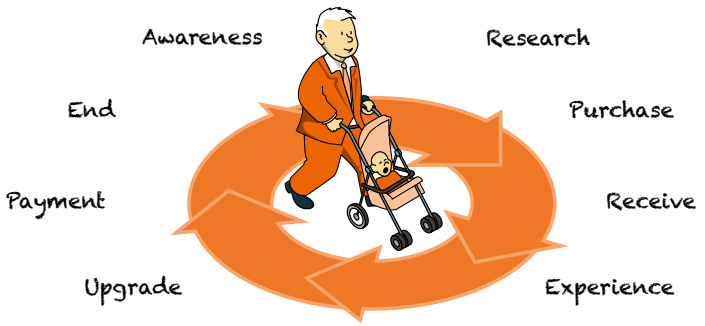
Source	Product Vision	Feature Listing	Use-case diagram	User Persona	User story
Who	Customer segment		User role	Persona	User role
What	Product	Feature	User story title		User story description
Why	Key benefit & Unique differentiator			Goal	Goal

CUSTOMER LIFECYCLE

A customer lifecycle model describes the high level customer and product interactions. It is used as a foundation to ensure customers are satisfied every single time they are in touch with the product.

Customer Lifecycle is a model coming from the Customer Relationship Management domain. Sterne and Cutlers original model is based on five basic steps: reach, acquisition, conversion, retention, and loyalty.

We usually use this model to go from product vision and key product capabilities to actual high level features. We use it to support product exploration in order to find our feature candidates. Our hybrid model below is more of a business requirement related tool and we have used it in various customer projects, mostly within the IT and telecom domain. We usually talk about them as buckets to trawl for requirements during collaborative requirement sessions.

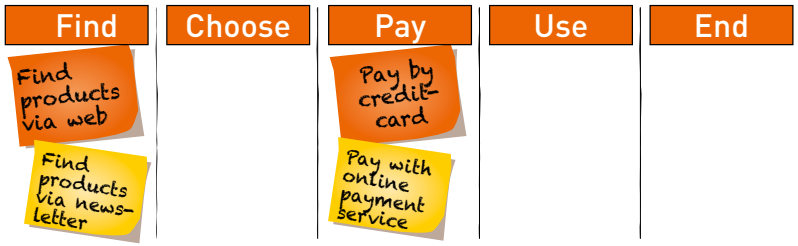


A Customer Life Cycle model also facilitates value prioritization when having to make trade-off decisions between adding new product features or adding capabilities to the process/systems enabling the customer experience, a decision that is related to where your product is in its Product Lifecycle. Your product might be brilliant but your sign up or payment support is a painful experience where customers give up, drop out of the cycle and never come back again.

Customer lifecycles can be, and are often, further divided into (communication) channels, e.g. retail store, web shop etc. Again, depending on where your product is in its lifecycle, the main focus of your upcoming product effort might be more customer lifecycle related rather than purely product related. It is important to keep a customer-centric view because the customer does not differentiate between the actual product capabilities and the supporting processes.

Here is an example on how to organize your customer lifecycle session when trawling for features related to the product effort we are targeting.

1. Sketch lifecycle stages on a whiteboard, or flip chart, in their order of appearance.
2. Identify wanted features and write them on post-it notes, e.g. "Find product via web", just stating the "verb + object" as a feature title is enough at this point.
3. Use color-coded post-it notes to distinguish between channels e.g., "Find product via web" and "Find product via newsletter"
4. Stick the post-it notes to their related lifecycle stage.



Still on a high level we have expanded our what, only briefly described in the elevator pitch, to a set of features needed for a complete customer experience. We now have a feature list of between 15–30 high level features which cover both product features and possibly also customer experience supporting features that live in one or several supporting systems.

Source	Product Vision	Feature Listing	Use-case diagram	User Persona	User story
Who	Customer segment		User role	Persona	User role
What	Product	Feature	User story title		User story description
Why	Key benefit & Unique differentiator			Goal	Goal

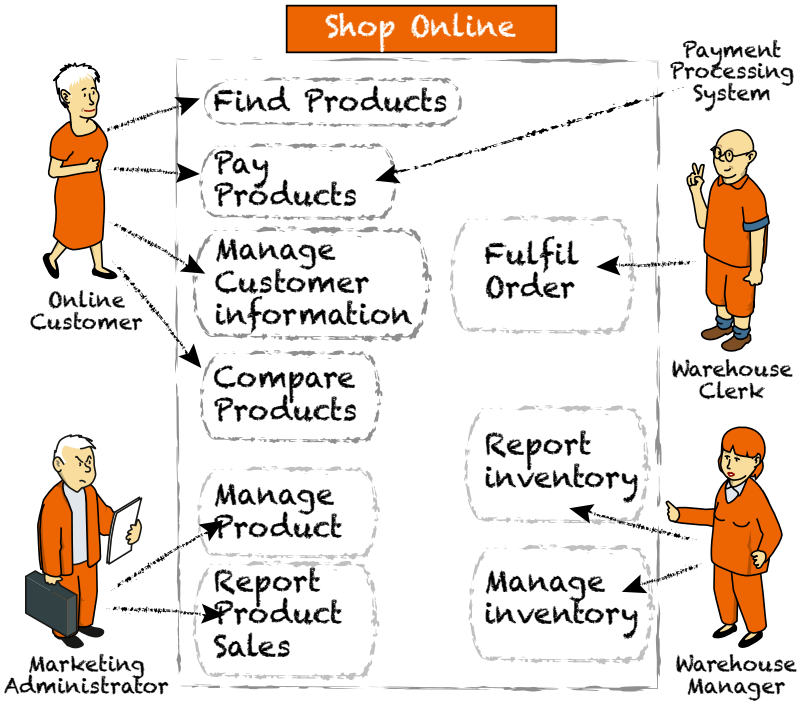
Let us move on and refine these high-level features. We still go for breadth-before-depth and steer clear of the details. However we need to reach a state where we can validate our exploration with user story writing and examples/acceptance tests for the prioritized features.

USE-CASE MODELING

We believe you can still model while working in agile mode. In fact we insist on doing it when working with agile requirements. The trick is to model light weight in collaborative and cross-functional sessions as part of refining our understanding of the features.

What are use case diagrams? Use case diagrams provide a simplified and graphical representation of an actor's interaction with a product, i.e. what the product must actually do. If the product vision represents the overall goal of a product, each use case (oval with a title, we recommend writing them in the user story format later) in the diagram represents a goal of an actor (straw man).

We are goal-centric (we can trace our use case goal back to the CLC features back to the Product Vision), we are user-centric (our actors/roles are connected back to the CLC and the product vision target group) and we get context that will help us understand how a single smaller feature (possibly written in user story format) can co-exist with other features to achieve a goal. What is not to like?

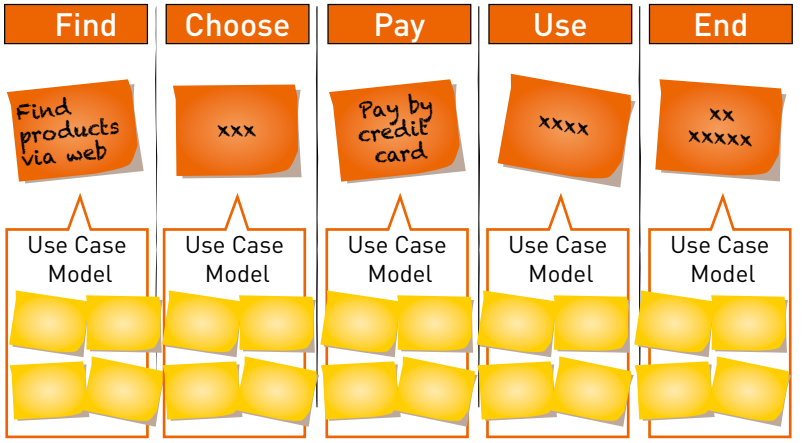


The actual users of the product are on the left hand side, whereas systems integration and support roles are shown on the right hand side.

The primary user role is placed in the top-left corner of the diagram. We increase the readability of the use case diagrams by arranging use-cases to imply timing.

Service design is complex; it combines parallel development of both product and process, i.e., systems integration and supporting roles. To cope with complexity, we break down each customer lifecycle stage into its own use case diagram. This is why we recommend use case modeling specifically in this loop.

1. Put identified features at the top of the flip chart. State the goal for the specific feature your use case diagram is exploring, e.g. Pay by credit card. Make sure you get a first quick prioritization of the high level features which need refining attention now, and which of them can stay on a high level for now and be dealt with during the first 3 iterations or sprints of the first release. After this is done, only work on the ones requiring immediate attention.



2. Start identifying user roles.
   - Which customers are on this customer lifecycle stage?
   - Which customers could be?
   - What systems, services or support roles do we need integration with or get help from?

3. Name user roles with singular domain-relevant nouns.
   - Write them on post-it notes and stick them to the flip chart.
   - Straw man as a symbol is optional.

4. We continue by identifying the use-case needed by the user roles.
   - Why does a "customer" use the product?
   - Why would the "payment processor system" have an interface with us?

5. Begin use case titles with a strong user-centric verb from the domain terminology.
   - Write down identified user story titles on post-it notes, e.g. Search for items
   - Stick them to the flip chart and draw line connections to user roles
   - The oval symbol is optional.

Remember to make one use case diagram each per identified channel, e.g. "Find product via newsletter" and "Pay by online payment service".

Each high-level feature is refined into 5–10 lower level features, using roles and verb + object titles. This makes it easy for us to write user stories should we want to. Our understanding of what and whom has increased quite a bit and we have a context for each lower level feature that we can use to zoom out and connect back to our CLC exploration. Please note that not all high level features have to be decomposed and refined. Even after the immediate attention prioritization in step 1 we might discover, while discussing the goal or while modeling, that this is definitely not a candidate for a first release effort for various reasons. Again, by modeling we are validating our understanding of the high level need a customer or user role has. If we can't answer questions while refining and decomposing we should immediately stop working, send exploratory questions back to the business stakeholders and proceed with modeling another high level feature.

Thanks to use case modeling, we might have a refined feature list of between 80–150 features at this point. We strongly encourage you to avoid decomposing all high level features up front, it is neither valuable nor feasible to assume they will all be part of a first release. They will only create an inventory, which is an unnecessary waste. 100–150 items is a good limit to strive for, it fits the so called Dunbar number. Note the emphasis on limit. We don't want you to end up in "story card hell" (Jim Shore).

Source	Product Vision	Feature Listing	Use-case diagram	User Persona	User story
Who	Customer segment		User role	Persona	User role
What	Product	Feature	User story title		User story description
Why	Key benefit & Unique differentiator			Goal	Goal

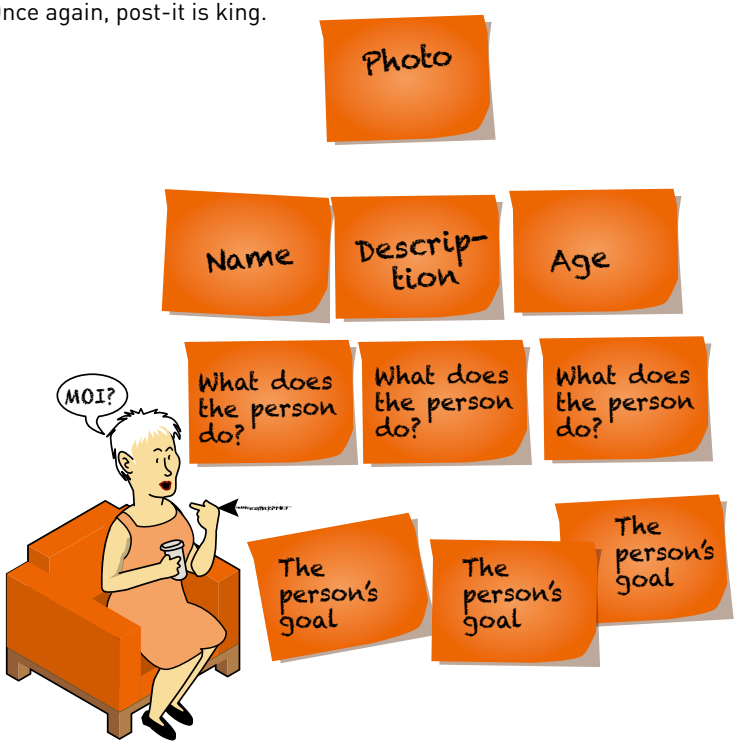
It is usually very creative and fun work exploring and refining features, but we need to revisit the statement of need from the product vision to make sure that our explorations fit the needs of the target group. That it is the correct work to invest in. We need to find out more about who we are dealing with. Let us zoom in on our customer segment. Another benefit of doing this is it validates that we did not miss out crucial areas, or equally bad, ran away and got stuck in gold plating land.

USER PERSONAS

A persona is an archetypal description of a fictional person's characteristics, demographics and goals.

We make 2–4 draft user personas illustrating the top three characteristics, demographics and goals of our primary customers.

Once again, post-it is king.



The user personas add customer understanding. We are especially interested in their goals, which we use to validate our current user stories.

- Did we learn anything new from the user personas?
- Do we need to adjust some of our lower level features/user story candidates?

A word of advice, do not mix up user personas with the less informative actor description, which represents a specific user role of a product, such as an online customer.

Tip – at this point you could say that we are creating user persona mockups that could be detailed out later as part of UX work during the first and second sprint/iteration of the release. As such, the draft or full user personas can be used for UX trade-offs decisions as well as global conditions and constraints prioritization during the actual release work – well worth investing in.

Our first complete refining cycle has come to an end, we have achieved a foundation for why we are planning to do what and for whom. At this point we have probably spent 5–6 days of our time box and have explored as well as validated quite a lot of requirement content.

Source	Product Vision	Feature Listing	Use-case diagram	User Persona	User story
Who	Customer segment		User role	Persona	User role
What	Product	Feature	User story title		User story description
Why	Key benefit & Unique differentiator			Goal	Goal

Let us make some estimation, prioritization, calculation and planning before we continue with further decomposition and detailing. The reason for this is to make sure we spend time on what has most value to the organization and the customer. Please note that we have not yet written any stories. We are staying in lower feature/user story title land for as long as possible.

BUCKETING ESTIMATION

Estimation is based on a combination of experience and guessing. It is never correct and from a lean perspective a waste. Still, we need to estimate expected effort in order to be able to prioritize and plan upcoming work. We use bucketing in combination with T-shirt sizes to be quick and not waste too much time.

We still have not written any user story descriptions, we are boldly trying to survive with strong verb + object titles to support the conversations during the collaboration sessions taking place during the loop. Also note that this is doable because the idea is to colocate and run the collaborative sessions during less than 10 days, preferably consecutively.

Bucketing estimation will support the validation centric work mode. If we have difficulties in doing a first bucket estimation of a strong verb + object user story title (with supporting use case models and customer life cycle visualizations) we can either write the story as a way to probe deeper on role and goal to see where the ambiguity lies, or catch key assumptions or exploration questions on the back of the user story title post-it or index card.

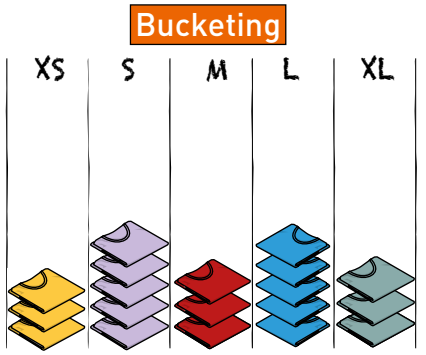
1. Make a swim lane wall with T-shirt sizes at the top, starting from XS, S, M, L, XL.

2. Choose a "reference story" to compare everyone else against. A common strategy is to choose the smallest one and declare it to be size XS.

3. Stick user story titles to the corresponding swim lane.

4. Mark each user story title with its size estimate, e.g. S.

With rough initial bucket estimation we can now do a small, skinny first version of a release candidate separation which is our next step.



RELEASE PLANNING (INITIAL)

We prioritize stories for our first minimal release by weighing their business value against estimated implementation effort, basically setting up a release candidate queue.

The primary purpose of this is to identify which requirement content needs further immediate attention of analysis and refinement and which content can stay on a user story title level until the first sprints of the first release.

1. Make a swim lane wall with 1st, 2nd and Later at the top of the lanes.

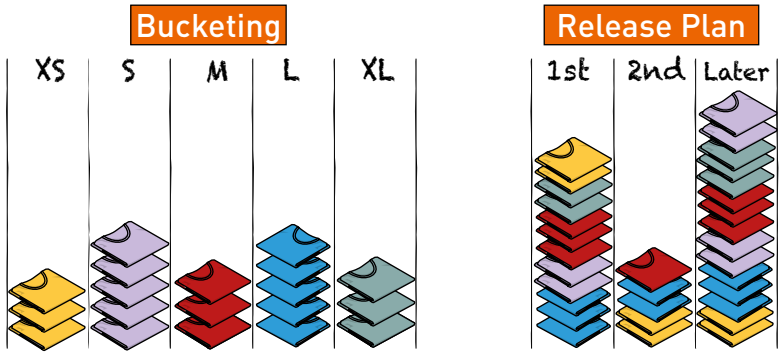
2. Stick the most important user stories to the 1st release lane, the minimal amount of stories needed to provide value, early learning and reducing technical risk.

3. Stick next priority stories to the 2nd release lane.

4. Stick the rest of the stories to the later lane (= backlog)

5. Mark stories prioritized for 1st release.

Initial release planning has given us an overview of what is required to deliver a complete customer experience or some version of a walking skeleton of one.



How do you make sure that not all stories end up in the first column? Conversation is the most honest and simplest answer. You can use constraints based on the total amount of requirement content gathered until now and divide their total to an even distribution between the three lanes.

We are still not making assumptions on capacity yet. We are trying to identify the most critical use cases (high level features) and their lower level features, trying to drill down a release strategy that makes sense from a business value as well as from a technical delivery feasibility perspective.

However, most user stories are still vague and too big for iterative delivery. They must be split and detailed with descriptions, acceptance criteria and additional information. Now we know which ones to focus this work on, namely the 1st release candidates.

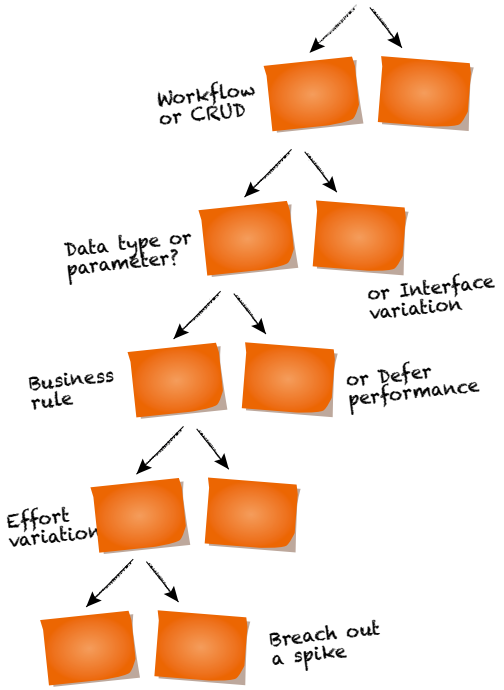
STORY SPLITTING

The initial bucketing estimation makes a good starting point for further story splitting. As a rule of thumb, all stories above size XS need further decomposing.

The general guidance favors vertical slices before horizontal slices, e.g. splitting stories through all architectural layers. In the image below you can see the most common splitting strategies for user stories.

In the end we want to have refined the user stories so that they can be pulled into the iterative delivery flow, we want them to be actionable. The INVEST model serves as a high-level definition of ready, when a user story is actionable:

- Independent. Reduced dependencies = easier to plan.
- Negotiable. Details added via collaboration.
- Valuable. Provides value to the customer.
- Estimable. Too big or too vague = not estimable.
- Small. Can be done in less than a week by the team.
- Testable. Good acceptance criteria and test example.



In our experience you can gain a lot by focusing on splitting stories based on operations (too large verbs), based on data (too large objects) first because they are directly related to the requirement work – the quality of the conversation, finding the functional gaps, or the too large holes in the Swiss cheese.

Splitting based on deferring performance and even by business rules is more about finding value and/or technical implementation strategies. The challenge here is not to get stuck in decomposition, slipping into inventory creation and story splitting gold plating.

Remember the following:

– Every user story has to provide value, we are not shipping crippleware just because it was a more comfortable way to deal with implementation difficulties. By crippleware we mean both delivering the wrong thing in the wrong way as opposed to delivering the right thing, right and fast

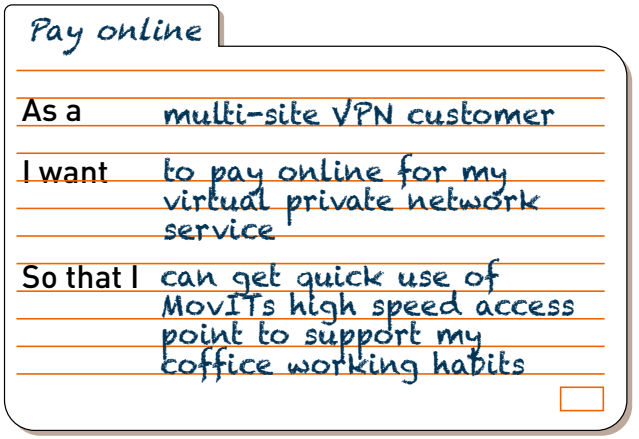
– Decomposition will only take you so far. Do not get fooled into thinking that breaking down a problem into its component parts and then aggregating those parts together again will yield the original problem (or feature). It still has to make sense from both a business and user perspective and it still needs to be worth it from a technical implementation perspective

Here is an example of a user story in need of splitting because the operation (verb) is too big:

Remember our Product Vision:

FOR a multi-site vpn customer, WHO HAS a need for movable high-speed access point,
MovIT IS A virtual private network service
THAT provides 4G access with quick launch and backup capabilities
UNLIKE competing vpn services
MovIT has one contract, one bill and one help desk

User story example (too large verb):



The operation in question, the verb is "Pay online" and it is too big. Here are some exploration questions that can lead to splitting options:

– Am I authenticated as a registered customer already? If not is that part of another scenario?

– Can I choose different payment options (invoice, credit card, PayPal, bitcoin, gold nuggets ...)?

– Do I want to choose an extra security layer by authenticating myself to the external card holder as well or would that just frustrate me? Do I have to for regulatory purposes?

– Can I receive payment confirmation and by what means (mail, sms, paper, fax, pigeon ...)? What minimal amount if information about my payment do I legally have the right to get hold of?

– What happens if the credit card bank denies my online payment?

– Can I receive help or support if I have issues completing my payment and if so, how?

This story can easily yield 6–10 user stories and tie into non-functional/ cross-cutting aspects of security and error handling.

Here is another example of a user story in need of splitting because the data (object) is too big:

View usage metrics

As a	multi-site VPN customer
I want	to view my usage metrics for my virtual private network service together with their related costs on a monthly basis
So that I	can see how much my coffice working habits cost me

The data in question, the object, is "usage metrics" and it is indeed too big. Here are some exploratory questions that can lead to splitting options:

– What viewing options do I have, through my invoice or through an online self-service portal? If the latter is that part of another scenario or even a totally different system/product?

– Can I choose different period options (last 24 hours, weekly, monthly, quarterly, yearly ...)? How far back can I access my historical usage metrics?

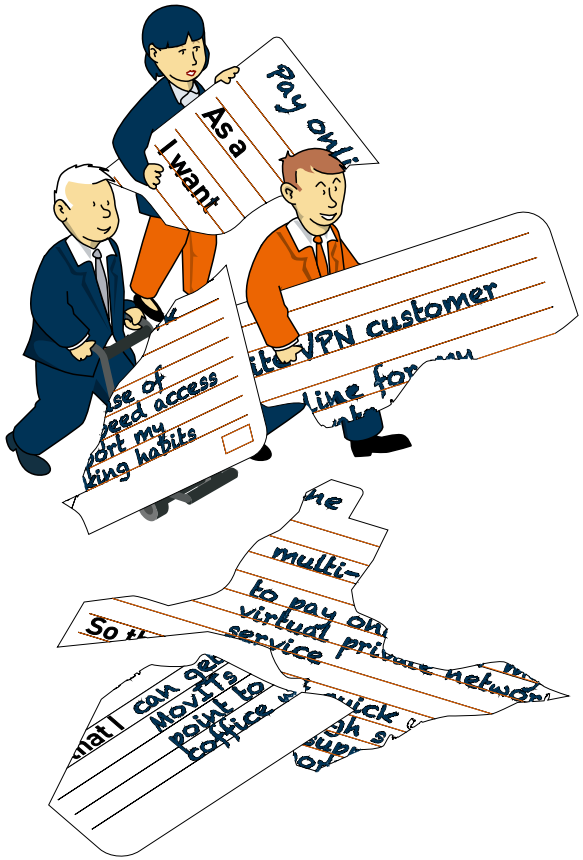
– What parameters of usage metrics do I want to see (if several which are more crucial)?

– If I have several different services at your company, can I access all my different usage metrics in one point or do I have to do it for each separate service (which really would make me very annoyed)?

– What pricing information/model do we need to access in order to give the customer the cost/metric information? Are there business rules we need to take into account?

This story can easily yield 4–8 user stories and ties into billing process and potential system integration, a classic customer lifecycle feature.

Story splitting is great fun once you get into it. The cross-functional conversations needed to find the "right" splitting strategies usually provide really good requirement value. Please note that even for release candidates in the first lane we should not split all, only the ones we know will end up in the earliest sprints or iterations. Splitting and writing user stories are two very tightly connected steps and they should be organized into a combined session with several diverge- and merge-sessions so teams can collaborate on a larger story set and come together and report to each other.



A large user story is split into several smaller user stories when details, like acceptance testing, are added. This story splitting practice is the cornerstone and driver of iterative delivery.

USER STORY WRITING

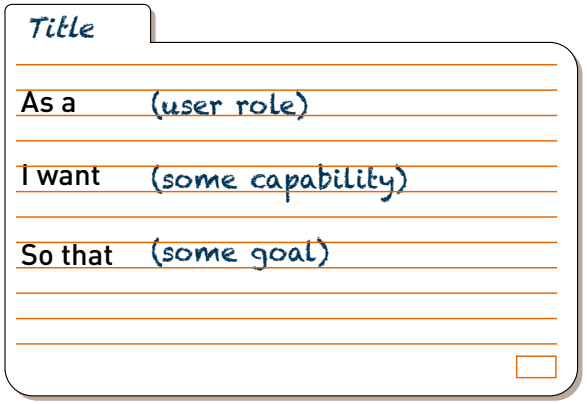
A user story is written from a user-centric perspective and it describes:

- **who**
- wants **what**
- and **why**

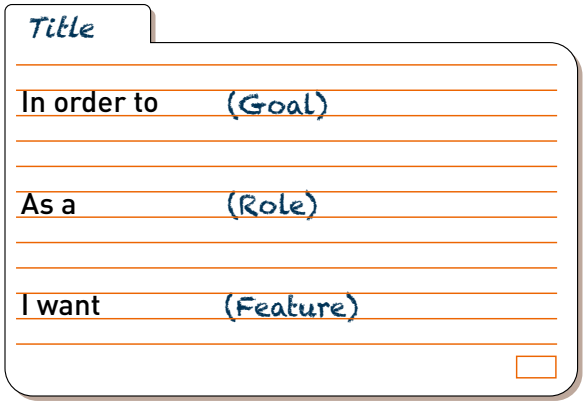
The two most important pieces of advice regarding user stories that also point to their XP origin are:

– "A user story is a reference point for a conversation"
– "Card, conversation, confirmation"

The user story title works as a short summary, it is not a requirement. It is a small token that supports requirement conversations during 1 week and 2 week iterations typical for XP:



Another format is:



Capability and Feature mean the same thing i.e. an observable feature that a user role interacts with to solve a task, expressed as a verb + object.

We don't want to get stuck in a format war. Here is some advice regarding our experience in working with user stories and their format:

– There are three main elements to a user story in both examples above. Choose whichever format that suits you but don't skimp on these elements. We see too many user stories that don't contain goals at all.

– We have stressed using a clear verb + object title of the story as a way to delay story writing until it is clear that it will provide value in the near future. This is the "I want" part and it is important to be as clear and precise as possible on the verb and object, it makes it easier to derive test examples.

– The most important part of the user story format is the goal. The goal is not a synonym of the verb + object, it is the real need. The feature or "I want" is a means to an end, nothing more. If you have difficulties pinning down the goal then don't skip it. It is a good signal that further exploration or postponing is needed.

– Be precise on the role. Very often teams still get stuck in the lazy habit of stating the role as "user". This makes the user story more ambiguous, am I a surfer, an evaluator, a premium customer or a multi-service customer? I will have different goals and needs and the user stories need to be tested based on these specifics. There are probably functional gaps lurking as well because of not differentiating the roles properly.

– When we encounter teams working with pseudo stories (or fake stories as Gojko Adzic and David Evans call them in their new book Fifty quick ideas to improve your user stories) and technical stories we work hard to identify the reasons why the teams use these formats and try to help them find the root cause instead.

– Not everything should be written in user story format. We feel it fits best for capabilities, not conditions or constraints (usually called non-functional requirements). Capabilities are observable functions someone interacts with and non-functional requirements are cross-cutting concerns touching on all or clusters of user stories.

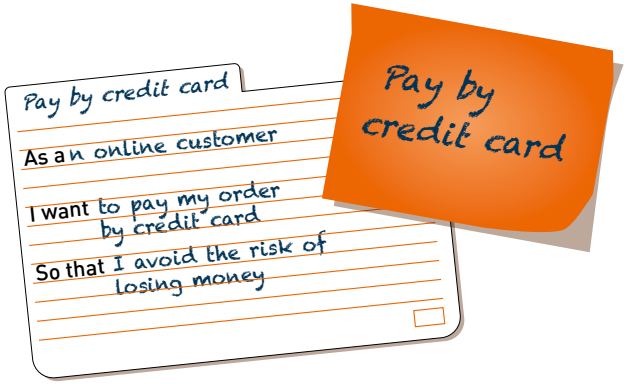
– Epics are large user stories, usually too large to fit into a single iteration. It is still a user story and is still expressed in user story format. We usually just talk about large or small stories or other T-shirt sizes. Many teams we have worked with have been confused about the term and its meaning so we rarely use it any more.

– You do not have to use the user story format if you don't want to. Please make sure you still write precise verbs + objects titles and differentiate be-

tween capabilities where users execute actions. Here you need to connect the subject and/or actor or role to the capability so that you still are user-centric. Add a description to the title. We do recommend the user story format for describing capabilities – all the three key elements are there.

User stories were initially written, processed and kept on index cards. Nowadays, many teams manage their user stories electronically in different types of backlog and agile planning/tracking tools. In workshops and during loop work we strongly advise working using index cards, or post-it notes, to get everyone attending the workshop involved in the process and able to contribute in an easy way. The goal is to get everyone writing stories, and learning that it is great fun.

1. We identified user story titles into user stories simply by adding a description in accordance with the format above. telling who wants what and why.



A user story has a lifecycle which we can see in the various steps in this loop. This means that an initial user story description must be discussed and processed from business, test and coding perspectives, many times before the story is ready for implementation.

The format above is only half the story. A complete story contains significantly more information where the most important is the acceptance criteria. As it's hard to pick acceptance criteria out of the air, we need a better, methodical approach.

Acceptance criteria can be explored with the help of the Behavioral Driven Development format, which describes an implemented user story's expected behaviors. It matches perfectly with the user story technique and if test data is included it makes the outcome crystal clear. Acceptance criteria can be written on the back of the index card of the story. It can also be formulated as a test example, or an actual small customer facing acceptance test.

Given some initial context (the givens)
And connect multiple givens
When an event occurs
Then ensure some outcomes
And connect multiple outcomes

The beauty of this format is that it gives us the necessary preconditions (given), the action we want to execute (when) as well as the post-conditions that should appear as consequences of the executed action (then).

A word of advice, if this format feels difficult to work with, just use test examples as promoted in Specification by Example (Gojko Adzic). The outcome will be roughly the same, your user story will be measurable and testable.

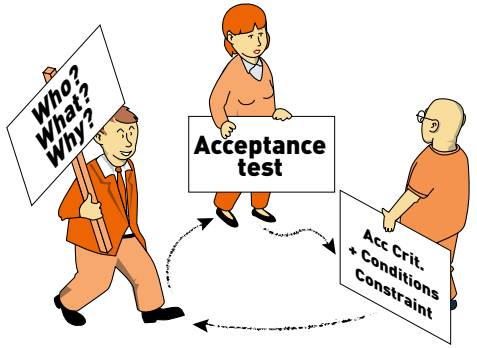
2. We start to talk about how we are going to demo the implemented user story and write a test example/scenario (covering the happy path), where the expected outcomes emerge as acceptance criteria.

3. Then we discuss if there are any alternative paths to be demoed and what their expected outcomes are. These are our next set of acceptance criteria (or potentially other stories).

4. When we talk about test examples we realize that the initial user story needs further clarification or perhaps needs to be split into two user stories. Clarification often leads to other acceptance criteria.

5. We also get ideas on how implementation should be done. Implementation details usually add acceptance criteria.

6. Finally we discuss whether there are any conditions or constraints that the specific user story must meet, e.g., performance criteria or business rules. These become acceptance criteria and test examples.



7. The identified acceptance criteria, acceptance test examples and additional information is written down, either on the back of an index card or on additional post-it notes.

A note on definitions of done and their relation to acceptance criteria and user stories. Before entering iteration planning a user story should have acceptance criteria and/or test examples. How else can we know when we have fulfilled the product behavior a user needs to fulfill his or her goal?

A definition of done is the quality contract between product owners and the delivering organization. We need to agree on what we mean when we say a user story is done. Passing acceptance tests, passing various automated test builds as well as updating release notes on a product wiki can be definitions of done steps. Meeting all identified and agreed acceptance criteria for the user story can also be a definition of a done item. So acceptance criteria can be part of a definition of done but a definition of done is not an acceptance criteria.

Our user story conversations not only help us to capture acceptance criteria, we also discover specific conditions/constraints and omissions. Not only have our conversations validated the user stories but the previous steps of exploration and validation mean that we are fairly confident that these are the right needs described.

Source	Product Vision	Feature Listing	Use-case diagram	User Persona	User story
Who	Customer segment		User role	Persona	User role
What	Product	Feature	User story title		User story description
Why	Key benefit & Unique differentiator			Goal	Goal

Story splitting and writing have generated a number of detailed user stories. How do we get things sorted for 1st release again? Remember that we strive to not split and write all user stories in the first release lane, only the ones we think would need immediate attention in the first release. The rest stays described and estimated on a higher level, to be dealt with ongoing product backlog refinement sessions when the release has started.

STORY MAPPING

A story map (together with the product vision) tells the "whole story/narrative" of a product, a technique described by Jeff Patton. User role, feature and user stories are organized into a structured model with a horizontal backbone of role driven activities and vertical details in the form of user stories to fulfill the activities:

1. Arrange user roles from left to right in priority order

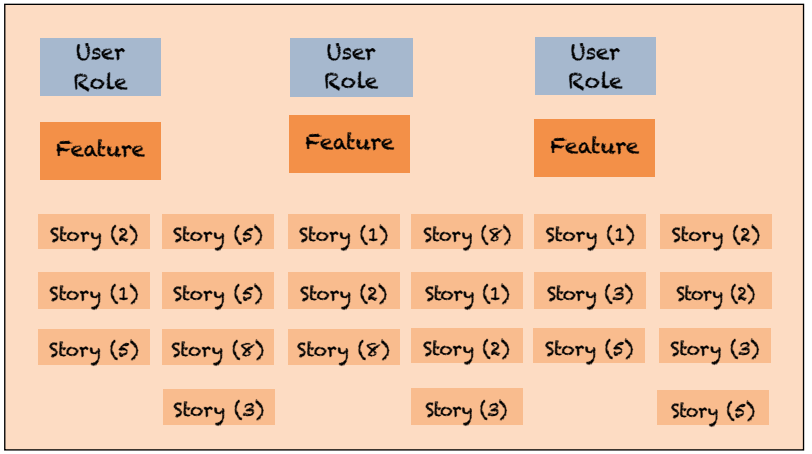
2. Arrange features from left to right in time sequence order (or priority order if sequence is unimportant)

3. Draw connections between user roles and features. User roles commonly use more than one feature.

4. Organize all decomposed user stories beneath each feature. A high position indicates they are absolutely necessary, lower position indicate they are of less value.

Tip – split into smaller team fractions, populating one feature each to ease collaboration and to gain speed, before merging them all together on one big wall.

5. Finally, discuss and capture driving global conditions and constraints as well as time constraints.



NOTE: Story mapping is called experience mapping (as-is) or customer journey mapping (to-be) in the service design domain.

Customer journey mapping is a kind of process prototyping for simulation and validation of the complete customer experience to make sure it works before it is implemented. This is exactly what we need to do with the story map before release planning.

Now that we have the refined and described user stories together with our overall feature list (some in more detailed format and smaller format than others, still staying as high level feature titles or lower feature title format) we can build the user story map to take a step back and validate the explorations so far and to support release planning.

VALIDATE THE STORY MAP

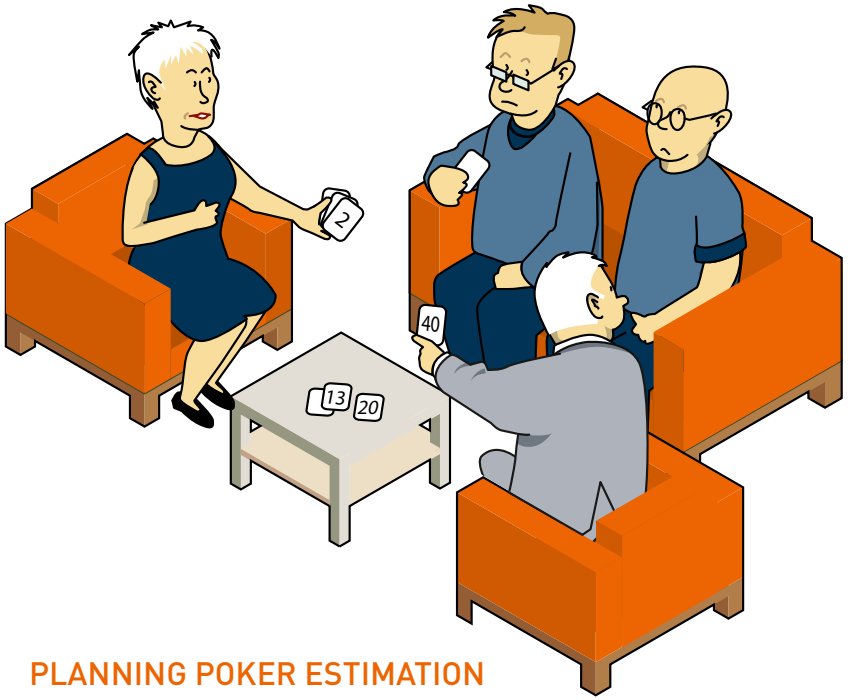
We take a step back and look at the complete picture and simulate the use of the product. We are looking for omissions;

1. Is there anything missing or looking strange?

2. Should we change the work flow order?

3. Does that user role really need that feature?

After a few simulation cycles, including changes and adjustments, we all agree that now we know what we know, for now!



PLANNING POKER ESTIMATION

Splitting user stories gives us new stories that we need to estimate before final release prioritization. We can bucket estimate again with T-shirt sizing but many teams want to put a numeric estimation to their user stories as part of the actual release planning.

As we approach implementation, conversation within the team is valued higher than being fast. So this time we estimate by means of planning poker. It takes a little longer, but this effort will pay off at iteration planning. There is good value in this step but you could see it as more optional than other steps in this loop. This is because:

– Some teams stick with their bucket estimating and don't put numeric estimations on their user stories. How can they survive without numeric estimations? They focus on their capacity to deal with a certain amount of XS, S and M items and after a few iterations learn "how many seats in the bus" their capacity is the equivalent of.

– Some teams use bucket estimation of T-shirt sizing and after quick group discussions (without the planning poker format) slot out rough story point numbers according to the scale below.

1, 2, 3, 5, 8, 13, 20, 40, 100

PLANNING POKER STEP BY STEP

1	2	3	4	5	6	7
Each team member gets a set of cards.	A meeting moderator selects a work package or a user story to estimate.	The person who knows best what is being estimated reads the story and gives a brief explanation.	The team discusses and asks questions to get a better understanding of what is to be estimated.	Each team member makes their own estimates by selecting a card without showing the others.	All show their cards simultaneously. If the estimates differ significantly the highest and the lowest values briefly explain their assumption behind their estimation.	The team iterates the estimation again following the previous steps until the estimation converges or the team agrees on an estimate.

1. Run planning poker while walking around or sitting next to the user story map.

2. Mark stories with story points estimates.

3. Remember the INVEST model – do another split if required and adjust the user story map if needed.

If you are going for numeric estimations, maybe use the Fibonacci sequence. Please don't forget that estimation is another form of collaborative requirement work when organized cross-functionally. A lot of teams get stuck in the actual meaning of the story point numbers and what they mean. They are an estimation of the size of the effort.

In our experience a good estimation session is filled with requirement conversations. During the session we validate our hidden assumptions and different interpretations of the need when estimating. Here is an example we developed for a customer a couple of years ago to get over their first numeric fixation and start having conversations about the user story instead:

Example of a 1 story point story:

- we understand the need clearly
- we understand what part of the application is affected
- we can (quickly) identify where the "fix" needs to be done in the code base
- the fix can be made in one place of the code base (not spread out)
- the need is "local", will not affect the behavior of other features
- fixing the need will not affect the data model
- fixing the need will not affect the architecture
- if the need is an improvement request it is probably about changing the behavior of an existing feature (adding, limiting rules and sub-behaviors)
- if the need is a bug it is probably about changing a faulty behavior in/or connected to an existing feature

Example of a 5 or 8 story point story:

- in order to understand the need we have to run a design session
- we understand what part of the application is affected, but need to figure out what other parts of the application could also be affected
- after some investigation we can identify what the right implementation strategy should be
- the fix will affect several aspects of the codebase and needs to be done in several places
- the need can and will affect the behavior of other connected features
- fixing the need can and probably will affect the data model
- if the need is an improvement request it is probably about adding a new feature in the application, or seriously revamping an existing feature (i.e. maybe for maintainability, performance, or usability reasons)
- if the need is a bug it is probably about addressing a silver bullet issue over several "layers" in the code-base

Example of a 8 or 13 story point story:

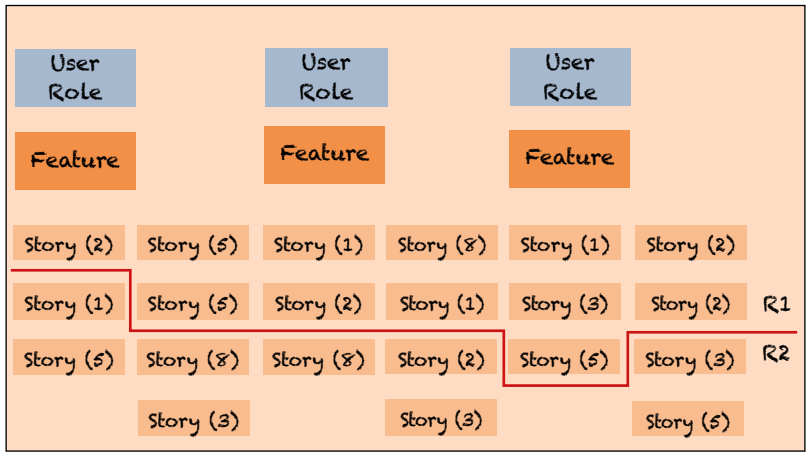
- in order to understand the need we have to have design sessions and architectural sessions
- we need to investigate what part of the application is affected and how other applications are affected (cross screening needed ongoing)
- we probably need to proof-of concept (silver bullet) first to find the right implementation strategy
- the fix will affect multiple layers of the application as well as other applications
- fixing the need can and probably will affect the data model and will probably create complexity for testing, building and deploying
- we may need to use new technology (3rd party software, new versions, major releases of current libraries)
- if the need is an improvement request it is probably about implementing a series of interconnected features or revamping messy/scary parts of the codebase that can hit us performance wise/raising more bugs easily

This guide shows our inclination not to use higher estimations than 13. You will survive really well using 1, 2, 3, 5 and 8 and no higher value on the user stories in a first release lane. Some of the higher values can be used for the high level feature titles but usually we stay with the bucket estimate T-shirt sizes on the larger items.

RELEASE PLANNING (FINAL)

Final release planning couldn't be easier:

1. Draw a line across the story map to mark what's included in 1st release.



2. Mark stories with 1st release.

3. Calculate the total number of story points for 1st release

4. Divide the release in iterations based on team velocity (or by guessing if no historical data exists)

5. Calculate the expected release date, based on the number of iterations multiplied by iteration length. Add one or two iterations depending on estimated risk.

6. Calculate expected cost based on the number of iterations multiplied by the cost for running a team for one iteration.

7. Present time and cost estimates as an interval; actual +– risk factor of one/two iterations.

If your teams don't use numeric estimation and the numeric velocity concept you can use boxed budgeting per iteration and do a rough total budget box for the release. As the iterations progress the teams realize their capacity of delivering stories of certain sizes during an iteration and can adjust the overall budget box early on if they foresee the need for more iterations.

At this point the loop is concluded and the first iterations of the first release can start. Through cross-functional collaborative sessions you have explored the customer lifecycle, potentially touching on multiple systems as part of the overall delivery of the customer experience you have prioritized to deliver.

Since the majority of the requirement content gathered was explored but not decomposed and described up front you should have a manageable amount of release content to work with and continue the refinement work during iterations. Teams can spend 5–10 % of iteration work to continually explore and refine requirements in parallel with the delivery flow, or more to the point, as part of the delivery flow. They revisit the methods described in this loop as the toolbox for this ongoing refinement work.

What the loop does provide is an initial exploration of the customer lifecycle that gives just enough data to get started with enough confidence, working just-in-time and accelerated with requirements. Even more important, we have to have disciplined and useful cross-functional requirement conversations that help us explore what the right needs are – helping us to deliver value.

... NOW WHAT?

How to support steady requirement work during the release

So far this loop has described examples of upfront agile requirement work, meaning exploration before initiating the first iteration. In some organizations there is no other alternative to this approach. Imagine the typical proposal to an external customer set-up where this loop in essence is an agile sales strategy.

That is one way of executing this loop. If you have less of an initial investment constraint on ramping up people to the teams or more fixed product teams you could also organize the requirement work in this loop during the first and second iteration of the actual release. By doing so you would concentrate on starting development directly after the use case modeling session.

This way the remaining work in this loop can be scheduled into the upcoming iterations, while the team starts to implement some of the low hanging fruit that very could be stamped early with "immediate attention". By executing the loop this way you may have your validated release plan at the end of the second iteration while having sampled some initial implementation and demo results.

Now is the point where you will start to reap the real requirement value of working in an agile delivery flow. Here are some recommendations on how to make sure that agile requirement work turns into a steady flow, not just a one-shot engaging the team early in the release.

Visualization

We recommend teams to have a product board next to their task and/or Kanban board. The product board supports the team to zoom in and zoom out on product aspects, clusters of features, role definitions and other requirement related content. It is too easy to write lots of user stories, dump them into a digital tool and drown in backlog items every time the team needs to pull from the backlog.

Requirements need context in order to be understood and shared. The product board would benefit from having the elevator pitch clearly stated. It can also contain the high-level view of the user story map, staying in high-level feature title land. You can extend this visualization to illustrate critical areas touching on non-functional attention. You can also use the story map to illustrate what areas are done by color coded magnets, which areas are not relevant for now or which are critical. It can also contain user persona overviews which help the team to use the personas as intended, to arbiter different user needs during development and highlight the "voice of the customer".

Finally, we also encourage a slot on the product board showing the high-level view of the release plan with time constraints. This way the team can zoom in and zoom out from a short term iteration perspective to the long term release constraints.

Refinement is ongoing

Requirements are not for free, you have to pay the piper as we said in the start of this book. We have heard people in organizations adopting agile questioning whether requirements are needed at all in an agile delivery model. Just to be clear on our view. For us requirements, i.e. customer needs, are at the core of agile.

In this loop we tried to keep a disciplined approach on not taking on too much inventory along the requirement road. A customer lifecycle-driven strategy will produce a lot of requirement content. The key strategy to survive is the continued refinement work that takes place during iterations:

– Refinement sessions (also called backlog grooming or backlog refinement in Scrum)

A cross-functional session can contain the entire development team together with their product owner, customer representative and/or product manager depending on which agile method you use. It could also be executed in a pure three amigo constellation (one business analyst, one developer, one tester). It can take place one or twice during the iteration and can take 1–2 hours. Be sure to keep up a steady and predictable cadence for these sessions so teams and stakeholders learn to really use the opportunity given. The main purpose of the session is to take on new requests and explore them and/or decompose already existing high level features into smaller user story format. Participants collaborate using slices of the requirement techniques mentioned in this loop: they might model the single feature, using splitting strategies, estimate and discuss acceptance criteria and test example. They identify new explorations needed that might have to be scheduled into coming iterations before further decomposition can take place. Be careful not to just focus on validating the upcoming iteration candidate user stories. You need to take a look at upcoming high-level areas on the product board as well as being able to swiftly respond to new incoming requests.

– Requirement workshops with end-users and/or customers and other stakeholders

The development team supports the elicitation of new needs by participating or running requirement workshops where they engage different stakeholder groups. Requirement workshops of this kind are usually either about eliciting needs or validating interpretations of needs with models, user stories and mock-ups.

Let's not forget that the iteration planning session (sprint planning in Scrum) is an opportunity for refining requirements as well. While agreeing on the scope of the iteration we might clarify acceptance criteria and test examples for a user story, decide to split one and quickly draft and estimate the new user stories and put them aside. Don't get too stuck in expecting a complete and neat decomposed candidate list of user stories for the iteration from a product owner. Iteration planning is a window for cross-functional requirement conversations. It is not a forum for handing over an order and wait for your take-out or say "talk to the hand and wait for the next sprint". We need to work more collaboratively with requirements.

Again, the product board can support teams in their ongoing refinement and decomposition work as it provides an important product centric look ahead when it is too easy to get bogged down in implementation details. It is a question of cognitive survival.

Demo means validating executable requirements

With an agile delivery model the previous hard line and difference between validation and verification gets blurred. In a more conventional phase based and documentation heavy approach it is crucial to validate the requirement

documents as part of the upfront requirement work. It will take a long time before someone gets to see implemented features specified in the documents. This is not the case with an agile delivery model.

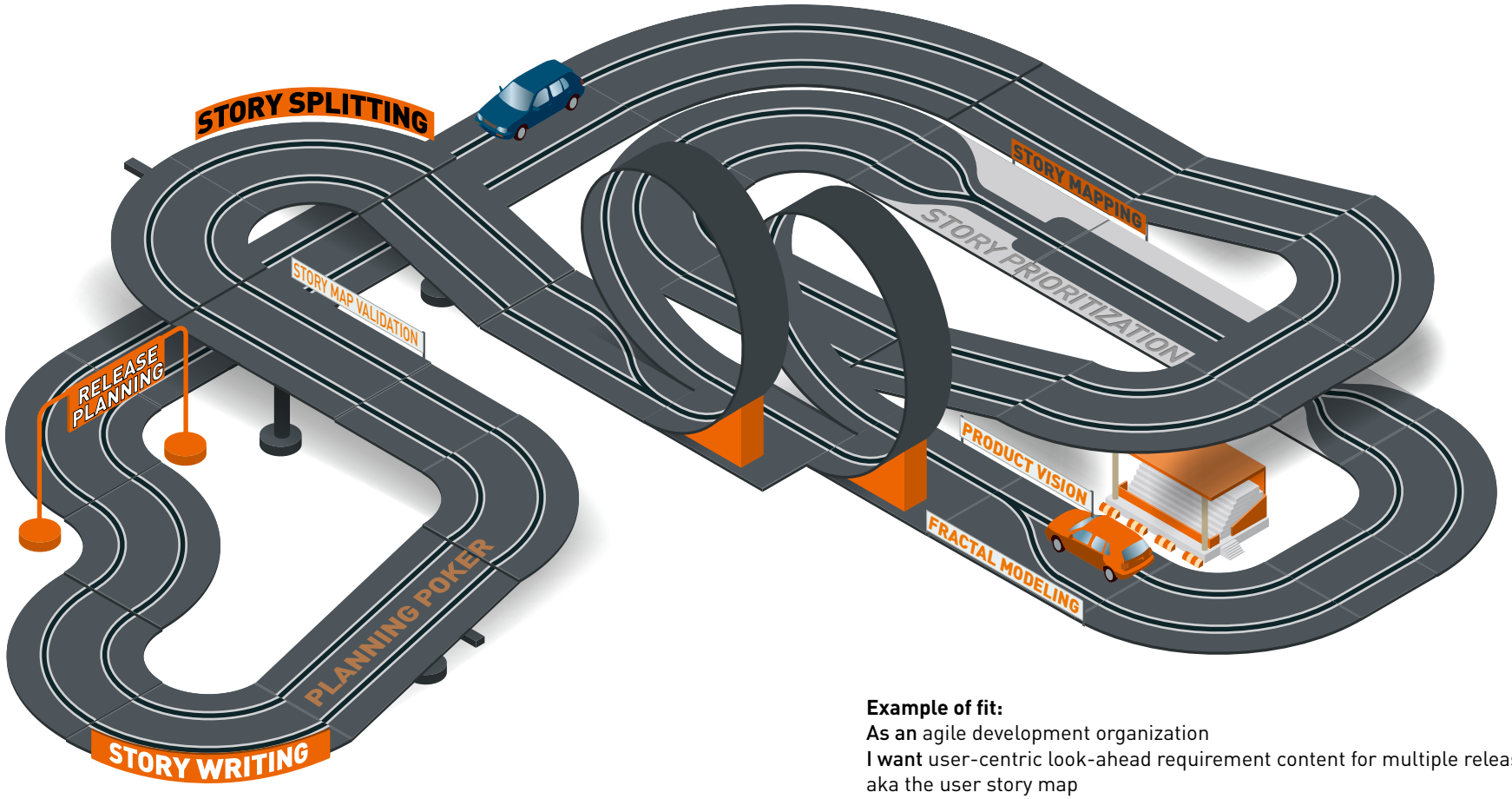
The demo (or review in Scrum) is where the teams present their produced result of value from the iteration. We encourage teams to view the demo as a requirement session, a validation of executable requirements. It should be organized as a requirement workshop where the emphasis is on validating the executable requirements (that are in a done state) and elicit new needs or improvements on the executable requirements.

An agile delivery model is all about the requirements and the lifecycle of a user story. How fast can you make your requirements executable?

Extract data for documents – not the other way around
An agile team working collocated would be fine keeping the release plan in this visualized format but most organizations have policies and standards pointing at templates or tools. This is especially the case in multi team and distributed team contexts where digital tool support is a prerequisite for survival.

Most modern collaboration tools manage user stories, acceptance criteria, test cases, story points, sketches, and release/iteration labels, etc. Required documentation can easily be extracted – it is just a click away. If your require sign-offs on formal paper, please do it this way and not the other way around.

Don't fall back into analysis paralysis and waste your time on making a huge requirement inventory upfront. Just nail the overall project (or release) scope and wait with the details until the "last responsible moment".



LOOP 2: GO!

Aim: User story map driven work mode

Background:
Legacy environment with multiple dependencies

Preconditions:
Domain model

Example of fit:
As an agile development organization
I want user-centric look-ahead requirement content for multiple releases, aka the user story map
So that I get release-based long term planning based on the right needs

Acceptance Criteria:
Verify that you have a validated product vision
Verify that you have a first level of feature decomposition through fractal mapping
Verify that you have the user story map backbone with roles and global constraints identified, aka the walking skeleton
Verify that you have prioritized your immediate next release candidates in your user story map
Verify that your release candidates are in a written and estimated user story format with acceptance criteria and placed in the user story map
Verify that you have a validated release plan in user story format covering prioritized stories and smaller stories as first iteration candidates

Time box:
5 working days

PRODUCT VISION

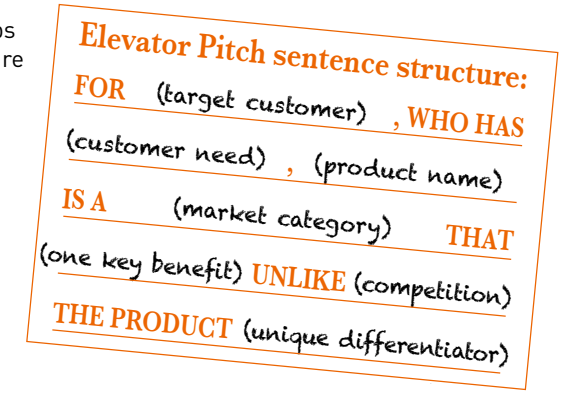
This loop starts with product visioning which is a good starting point from a requirements perspective. Usually a larger effort of planned work stays in a "concept state" while product management and/or chief product owners sort out delivery prioritization between this specific effort and all other efforts described in, for example, a product roadmap. "Concept state" at this point usually means 1–5 powerpoint slides that describe some overall need from a business perspective and the justifications for this specific need and effort.

From an agile requirement perspective we need a good product-centric starting point that clearly states the key capabilities and how they fit the target customer group needs. We need to clear away "business and project debris" and achieve consensus on the "what" and hence product visioning techniques are useful to create precision on the business level of the overall requirement.

As this loop covers a legacy environment there is an "as-is" situation and we are probably dealing with a product roadmap of some sort as "concept state" input. It may even contain a shopping list of high level feature ideas that still need to be converted into a good starting format for the requirement work. The product vision as a starting point helps us to pin down on a high level what the next crucial step in the product lifecycle will be and how that step fits the need of the target group.

The product vision step is part of the planning and collaborative practices of XP, eXtreme Programming, to initiate the product effort identification. Typical for XP is the playful, collaborative and accelerated nature of the vision activities. Don't mistake this useful requirement starting point as a technique only suitable for the development of new products or services to market. It fits ongoing development for existing products as well.

The elevator pitch helps us capture and structure our answers to these questions into a self-exploratory vision.



Imagine a time slot of 30 seconds to sell and/or inform someone about your product. Which information is essential to deliver a clear message? The elevator pitch is a good way to take the "concept state" and turn it into a distinct summary of the product effort you are about to initiate work on. What if the "concept state" is not able to answer the key questions enforced in the elevator pitch format, or if different business stakeholders have distinctively different interpretations? Then we fail quickly and business stakeholders are shipped back to explore and answer the questions raised in the elevator pitch session. Thus it is a very useful validation step and a good starting point for the requirement work.

Here is an elevator pitch example for a customer case we will explore to help you understand examples of output from the various techniques in this loop:

"For projects and agile teams who need to plan, execute and monitor their work, Eutaxia is a cloud based group collaboration tool that offers easy coordination of individual and team todos in a project. Unlike Project place and Microsoft Project, Eutaxia provides a dynamic search interface with tagging for customizing the tracking of todos in your project."

As a later step in the product lifecycle of Eutaxia, one of the releases implemented time and material support into the collaboration tool. Here is an example of how the elevator statement shows ongoing work on an existing product:

"For smaller Eutaxia-using businesses who need to shorten the time from work planned to work invoiced, the Eutaxia time & material release offers easy time and material reporting on specific todos with billing report support for faster invoicing of assignments coordinated through Eutaxia. Unlike conventional time tracking and billing applications, Eutaxia time & material release is centered on the workflow of assignments which reduces duplication of administration and data in different systems."

For existing products you can also use the "unlike" statement to differentiate and compare to previous releases of the product. Sometimes it can be difficult to pinpoint competitor differences. You can use the format of the elevator pitch to focus on the improvements being provided.

We go through the format and diverge into smaller cross-functional break out groups, to ease cooperation and to get multiple alternatives before narrowing them down and merging into one end-result.

1. The elevator pitch sentence structure is written on a white board, or flip chart, with breaks for the keywords, e.g., target customer.

2. The keywords are written on post-it notes to ease duplication and replacement.

3. The exercise runs in time boxes of 15–20 minutes, each ending with a demo. We show each other to generate new ideas. A useful practice is to create a large matrix on the whiteboard where every breakout group writes their candidate with clear column references to each step in the format.

4. The last time box is focused on narrowing down alternatives into one end-result. We use dot-voting to facilitate collaborative decision-making. This can be done by either dot voting on specific candidates or on specific format parts in the different swim lanes that result in a merged elevator pitch built from ideas from several candidates.

The elevator pitch gives us an initial understanding; for whom we should do what and why.

Source	Product vision	Fractal model	User story
Who	Customer segment		
What	Product		
Why	Key benefit & Unique differentiator		

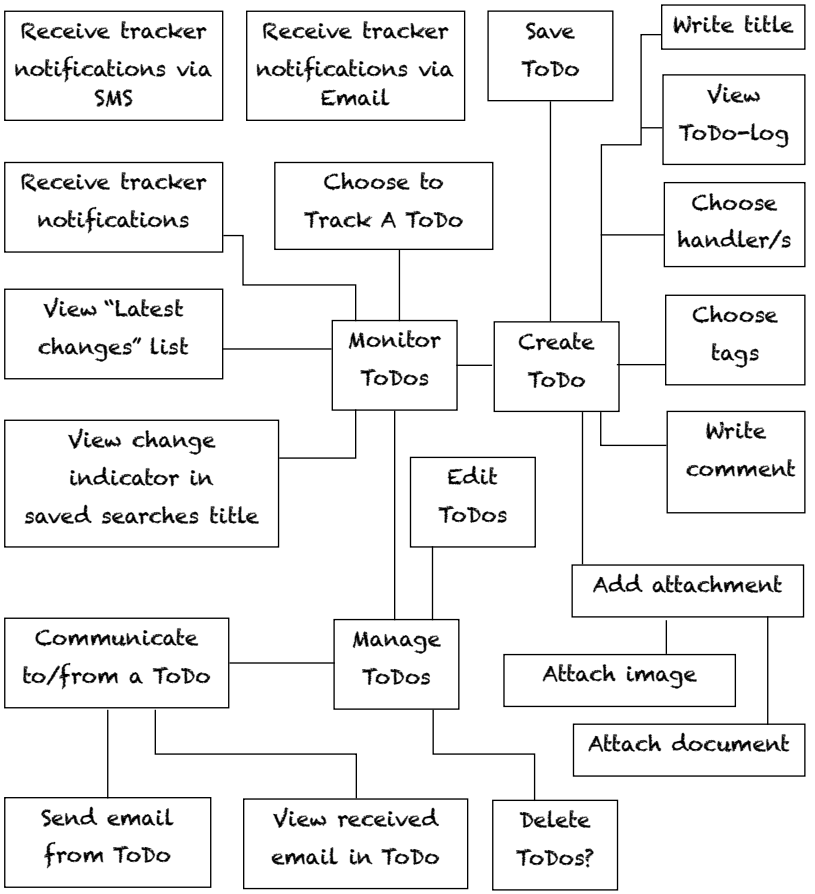
Let's hunt down some initial features that tell us more about what main product scope and capabilities we could aim for in the upcoming releases.

FRACTAL MODELING

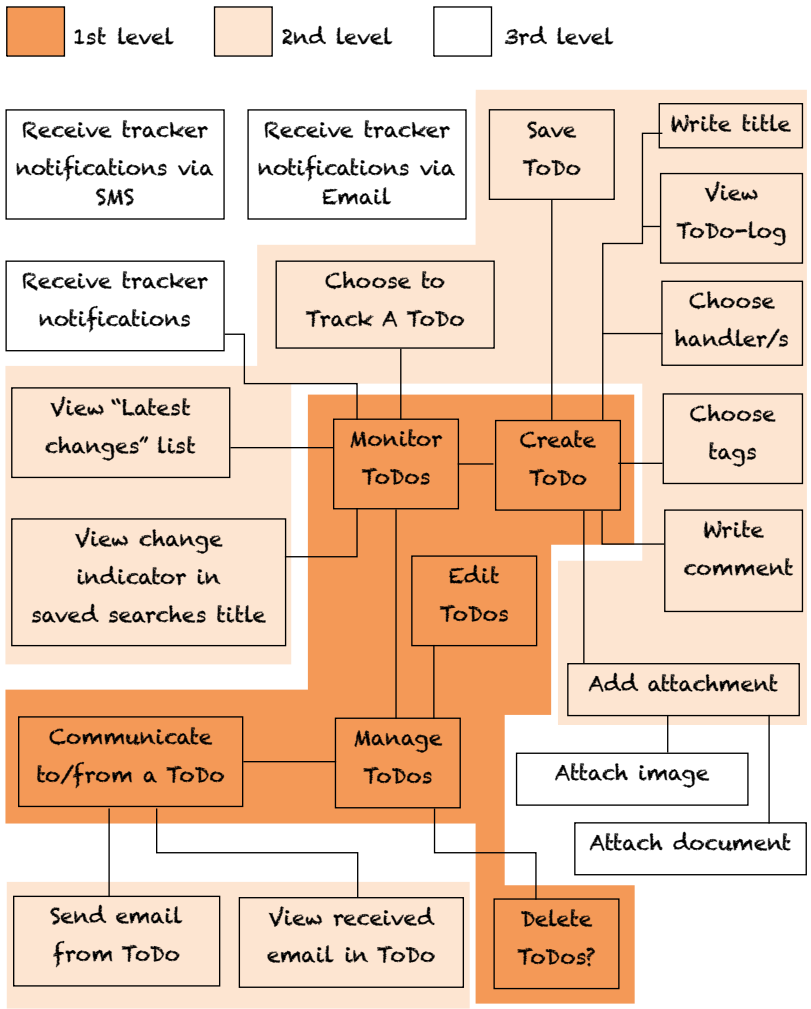
Fractal modeling is an exploration technique for quickly establishing a great number of user story candidates without too much methodology, but still in a structured way.

We came up with the format a couple of years ago when working with several different clients who struggled with use case modeling and process modeling formats. They felt that the more formal modeling techniques were too restricting in early requirement work and forced exploration to follow specific steps to answer to the elements of the modeling technique.

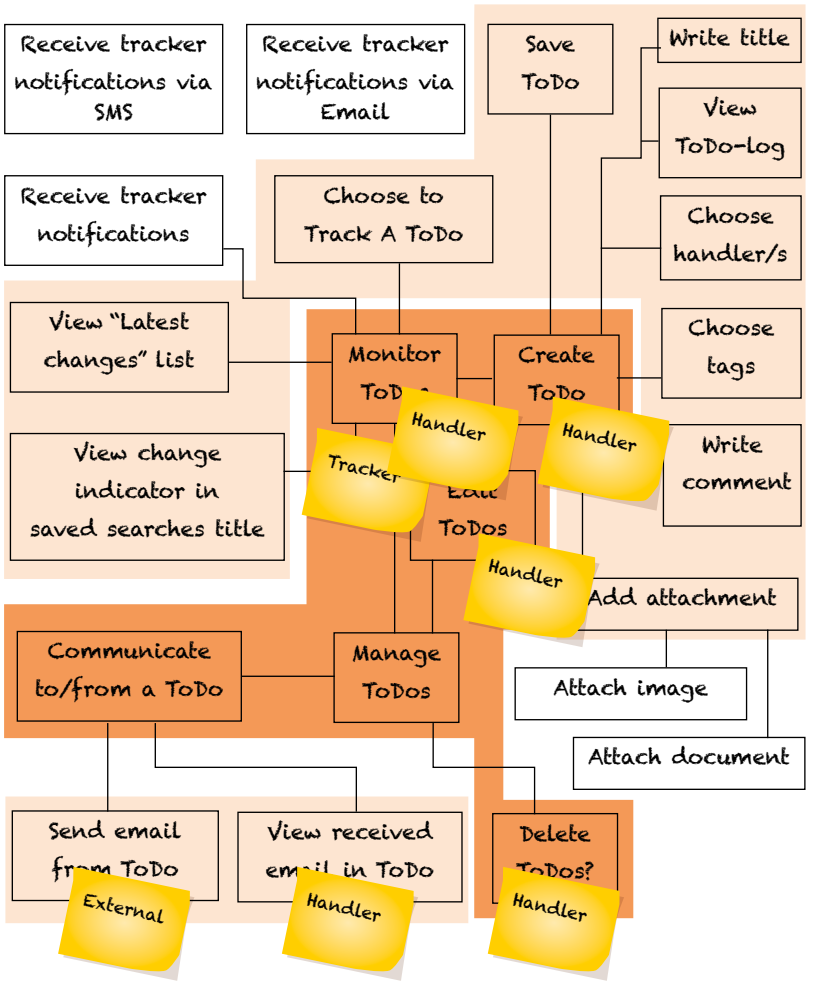
A fractal model is a feature node in the middle of a mind map that is being explored and decomposed through fractal branches in the mind map.



We chose the mind map format because many are familiar with the base format and how to collaboratively build them. The starting point is a high level feature title stated in verb + object format – in this example "manage todos". From this node the team explores the next level fractal branches of verb + objects (lower level feature titles) that decompose this high level feature title. The format allows for continued exploration in one branch in several levels or a quick swap to create a new fractal branch. You don't have to force the team to explore all level 2 branches before continuing to level 3 branches. Different fractal branches can be more in-depth and contain more levels than others.



An experienced facilitator with knowledge about different modeling techniques can guide the team to add modeling elements as part of the exploration, such as adding a "role" layer to all or some identified verbs + objects in the different fractal branches. As yet another layer you can circle an entire fractal branch and add a non-functional layer on a different color-coded post-it or you can circle the entire model and state more global non-functional attention areas.



This is a very useful technique aiming for breadth before depth:

1. Use blank flip charts, plastic film or a whiteboard for modeling.

2. Write the high level feature title on a post-it note in verb + object format, and stick it to the center. This is your starting point from where you can decompose and explore different fractal branches of the feature.

3. Split the feature using all possible variants of common splitting patterns:

   - Workflow steps or CRUD
   - Data types or parameters
   - Business rules or interface variations

4. Write the next level feature titles in verb + object format, on post-it notes and stick them to the model. Explore one branch at a time or move between branches, depending on the dynamics and attention of the team and the conversations.

5. Draw line connections between adult and children in the separate fractal branches while exploring. Use color codes to visualize the level 1, 2 and 3 of the branches.

6. Perform steps 1–4 in cycles, as long as new lower-level feature titles fall out. Depending on the granularity of your starting node you should avoid exploring beyond level 3 and definitely not on all fractal branches. It would probably not be valuable at this point. Add a role layer and explore if there are any non-functional areas of attention that are specific to one or several fractal branches.

7. Take the next high level feature, write it on a post-it in verb + object format and stick it to the center and explore away. Rinse and repeat, and focus only on those high-level feature ideas (or shopping list candidates) that you can directly connect back to the product vision. You might have a shopping list of 5–10 high level features and decide to only fractal model 4 of them at this point. Document by taking pictures or keep the flip charts.

The initial 5–10 high-level features have been broken down to 30–50 lower level feature titles. These are our user story candidates, written as user story titles. We have quickly generated a lot of opportunities based on the initial what, but how many of them are truly relevant from the users' perspective of solving their problems and achieving their goals?

Source	Product vision	Fractal model	User story
Who	Customer segment		
What	Product	User story title	
Why	Key benefit & Unique differentiator		

User story mapping will help us sort things out and show us the big picture. The high level feature titles we used as nodes are a good starting point for building the backbone a.k.a. the walking skeleton. Our user story candidates identified in the fractal modeling session will help us build the story map and connect it to the backbone. In our experience it is usually a good thing to separate these two steps: explore through fractal modeling and then build the narrative with the user story map.

USER STORY MAPPING

A story map (together with the product vision) tells the "whole story/narrative" of a product, a technique described by Jeff Patton in his book, User Story Mapping. User roles, feature and user stories are organized into a structured model with a horizontal backbone of role driven activities and vertical details in the form of user stories to fulfill the activities. When facing an existing system you can color code which horizontal role activities are "as-is", already existing and which need to be added in the upcoming release you and your team are exploring. The same goes for vertical content, using different color sets to indicate new features added and/or improved.

1. Arrange user roles in from left to right in priority order.
   Roles can be both customer related and back office related but they should tie to the overall system boundary. Don't forget to add "hidden or indirect users" who receive a report once every month on, say, time & material or usage metrics. They may sound like the villain of the piece but they are still part of the narrative and as such still related to the system boundary. Please also define the roles with precision. We have seen too many examples of user story maps that don't contain any roles at all or only the role "user". In the case of Eutaxia you can be a todo handler, a todo tracker, a report maker and an admin.

2. Arrange feature titles from left to right in time sequence order (or priority order if sequence is unimportant). Be sure that you are presenting the main user activities that take place in the narrative – this is your backbone. From fractal modeling you have your list of high-level feature titles and that will get you started. The narrative will help you identify which main user activities you have missed.

3. Draw connections between user roles and features. User roles commonly use more than one high-level feature title because they perform more than one user activity to achieve their outcome.

4. Organize all decomposed user story candidates beneath each feature/user activity. A high position indicates they are absolutely necessary, lower positions indicate they are of less value. Usually you add user story candidates in an "and" and "then" pattern below each user activity. The "and" pattern means just stating a vertical list of user tasks (user story candidates) that a user role can take in order to achieve their goal. For one of these user tasks you might add a "then" pattern horizontally, placing user story candidates that can have a sequential dependency. For example, under the user activity in the backbone "monitor todos" I will add all user story candidates in vertical positions sorted in order of importance. But for the user story candidate "choose to track a todo" I can add a horizontal "then" which could be "abandon tracking of a todo" (which can only be done if I first choose to track a todo).

> Tip – diverge into smaller team fractions after having built the user roles and backbone of user activities together. Populate one user activity/high level feature title per group using the content of the fractal models to fill out the backbone vertically. Have product owners or customer representatives move between groups to give input on the vertical sorting and positioning. Don't be afraid to make assumptions in the group if you don't have access to these roles right there and now. Your assumptions will be very visible and you will validate the user story map in a later step. This diverge strategy usually eases collaboration and helps in gaining speed before merging them all together on one big wall.

5. Discuss and capture driving global conditions and constraints as well as time constraints. Review any non-functional areas of attention identified while fractal modeling.

6. Walk through the user story map after merging and look for omissions as well as duplicated user tasks in the vertical content.

By now we have spent 2–2.5 days in the loop. We have a product vision, we have a fractal model exploration rebuilt into a user story map narrative. Our map contains both user roles, an understanding of their user activities (and goals) as well as the user story candidates (user tasks) needed for them to fulfill their outcome, sorted in order of importance. We have explored a lot of requirement content, but not a staggering amount, probably well below 100 items.

All of our user story candidates are still in verb +object format only. We have not yet written them out in actual user story format. For the first time we have a rough view of which user stories will need immediate attention early in the release. It is time to start writing user stories. Please note that we might still keep some of the user story candidates in verb + object title format if they are further down in the vertical areas of the map.

USER STORY WRITING

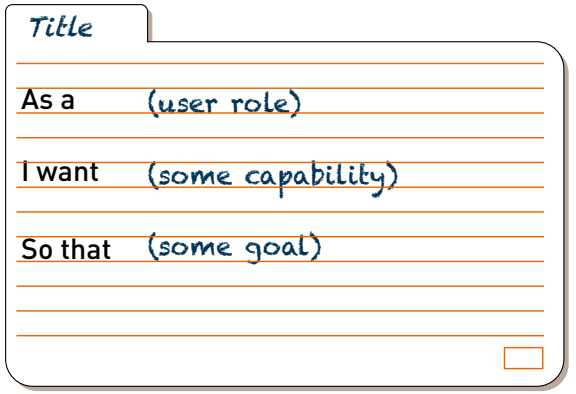
A user story is written from a user-centric perspective and it describes:

who
wants **what**
and **why**

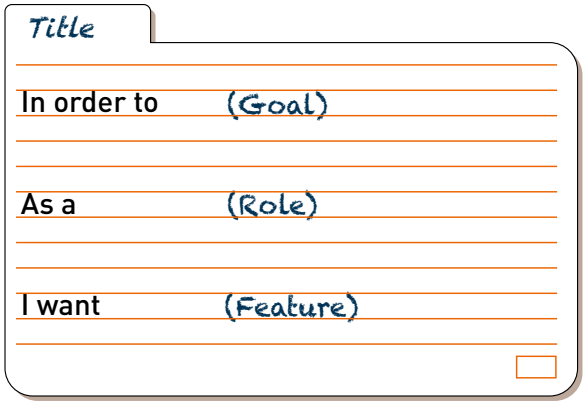
The two most important pieces of advice regarding user stories also point to their XP origin:

– "A user story is a reference point for a conversation"
– "Card, conversation, confirmation"

The user story title works as a short summary, it is not a requirement. It is a small token that supports requirement conversations during 1 week and 2 week iterations typical for XP:



Another format is:



We don't want to get stuck in a format war. Here is some advice regarding our experience in working with user stories and their format:

– There are three main elements to a user story in both examples above. Choose whichever format that suits you but don't skimp on these elements. We see too many user stories that don't contain goals at all.

– We have stressed using a clear verb + object title of the story as a way to delay story writing until it is clear that it will provide value in the near future. This is the "I want" part and we think it is important to be as clear and precise as possible on the verb and object, it makes it easier to derive test examples.

– The most important part of the user story format is the goal. The goal is not a synonym of the verb + object, it is the real need. The feature or "I want" is just a means to an end, nothing more. If you have difficulties pinning down the goal then don't skip it – it is a good signal that further exploration or postponing is needed.

– Be precise on the role. Many times teams still get stuck in the lazy habit of stating the role as "user". This makes the user story more ambiguous, am I a surfer, an evaluator, a premium customer or a multi-service customer? I will have different goals and needs and the user stories need to be tested based on these specifics. There are probably functional gaps lurking as well because of not differentiating the roles properly.

– When we encounter teams working with pseudo stories (or fake stories as Gojko Adzic and David Evans call them in their new book *Fifty quick ideas to improve your user stories*) and technical stories we work hard to identify the reasons why the teams use these formats and try to help them find the root cause instead.

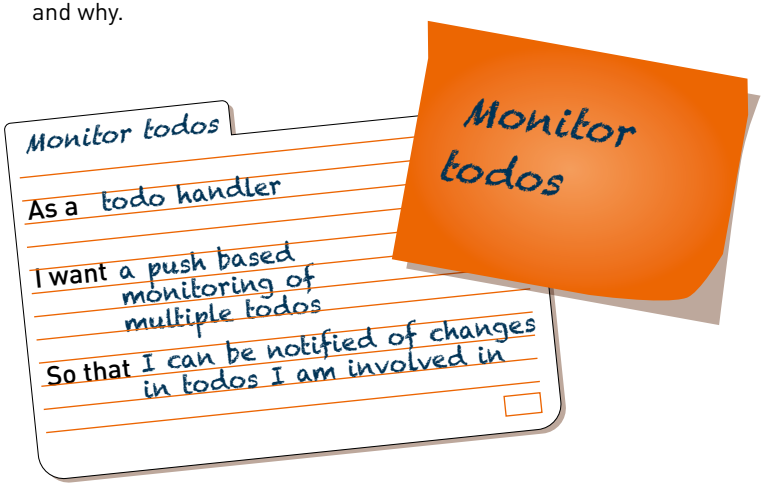
– Not everything should be written in user story format. We feel it fits best for capabilities, not condition or constraints (usually called non-functional requirements). Why? Because capabilities are observable functions someone interacts with and non-functional requirements are cross-cutting concerns touching on all or clusters of user stories.

– Epics are large user stories, usually too large to fit into a single iteration. It is still a user story and is still expressed in user story format. We usually just talk about large or small stories or other T-shirt sizes (see Bucket estimating in loop 1). Many teams we have worked with have been confused about the term and its meaning so we rarely use it any more.

– You do not have to use the user story format if you don't want to. Please make sure you still write precise verbs + objects titles and differentiate between capabilities where users execute actions. Here you need to connect the subject and/or actor or role to the capability so that you still are user-centric. Add a description to the title. We do recommend the user story format for describing capabilities – all the three key elements are there

User stories were initially written, processed and kept on index cards. Nowadays, many teams manage their user stories electronically in different types of backlog and agile planning/tracking tools. In workshops and during loop work we strongly advise working using index cards, or post-it notes, to get everyone attending the workshop involved in the process and able to contribute in an easy way. The goal is to get everyone writing stories, and learning that it is great fun.

1. We turn identified user story titles into user stories simply by adding a description in accordance with the format above telling who, wants what and why.



2. We start to talk about how we are going to demo the implemented user story and write a test example/scenario (covering the happy path), where the expected outcomes fall out as an acceptance criteria.

   The format above is only half the story; a complete story contains significantly more information, where the most important part is the acceptance criteria. As it's hard to pick acceptance criteria out of the air, we need a better, methodical approach.

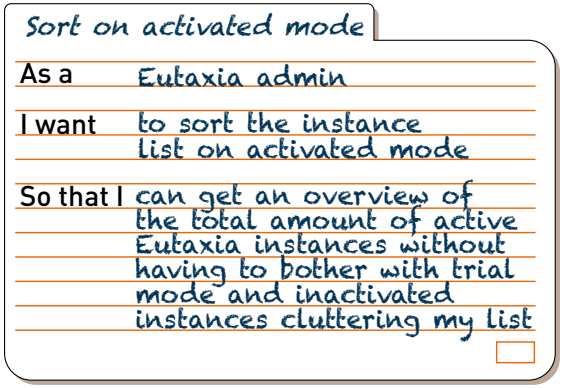
   Acceptance criteria can be explored with the help of the Behavioral Driven Development (BDD) format, it describes an implemented user story's expected behaviors. It matches perfectly with the user story technique and if test data is included it makes the outcome crystal clear.

   Given some initial context (the givens)
   And to connect multiple givens
   When an event occurs
   Then ensure some outcomes
   And connect multiple outcomes

   It is great value even if you just use the given-when-then format and don't proceed with the automated acceptance tests that are part of BDD, although we recommend doing that too if possible.

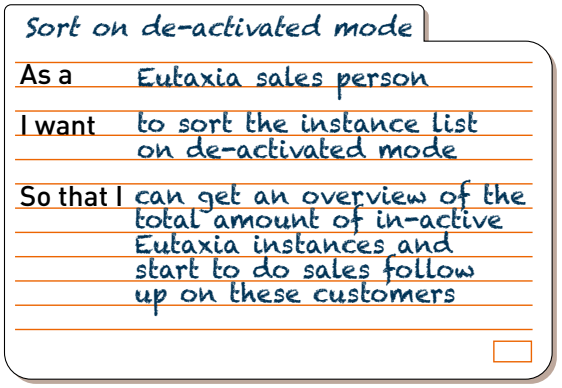
   Acceptance criteria can be written as such on the back of the index card of the story. It can also be formulated as a test example, or an actual small customer facing acceptance test.

Here are some Eutaxia related examples that cover features from the backend control panel for setting up new Eutaxia instances for customers:



Given the list of instances is sorted alphabetically (as default)
And the list consists of more than two instances
And the instances have different activation modes (on is green, off is yellow)
When the admin filters the list after activation mode "on"
Then the list should instantly rearrange to only show active instances
And the list of active instances should be sorted alphabetically
And the list should only show instances being green
And the list total should be same number as the list summary says

Another Eutaxia Control panel example of given-when-then as a good way to derive acceptance criteria:



Given the list of instances is sorted alphabetically (as default)
And the list consists of more than two instances
And the instances have different activation modes (on is green, off is yellow)
When the host staff filters the list after activation mode "off"
Then the list should instantly rearrange to only show de-activated instances
And the list of de-activated instances should be sorted alphabetically
And the list should only show instances being yellow
And the list total should be same number as the list summary says

And one more example:

Change the user limit

As a Eutaxia sales person

I want to change the user limit on a specific Eutaxia instance

So that I can bill the customer for their upgraded user amount

Given an active instance
And the current user amount is 0–5
When the sales person sets the user limit to 0–10 users
And saves the change
Then users 6, 7, 8, 9, 10 can be created from within the instance
And the instance reference person receives an automated email with info on the update
And the billing department receives an automated email with info on the update
And the next invoice should be based on the updated user limit

The beauty of this format is that it gives us the necessary preconditions (given), the action we want to execute (when) as well as the post-conditions that should appear as consequences of the executed action (then).

A word of advice. If this format feels difficult to work with, just use test examples as promoted in Specification by Example (Gojko Adzic). The outcome will be roughly the same, your user story will be measurable and testable.

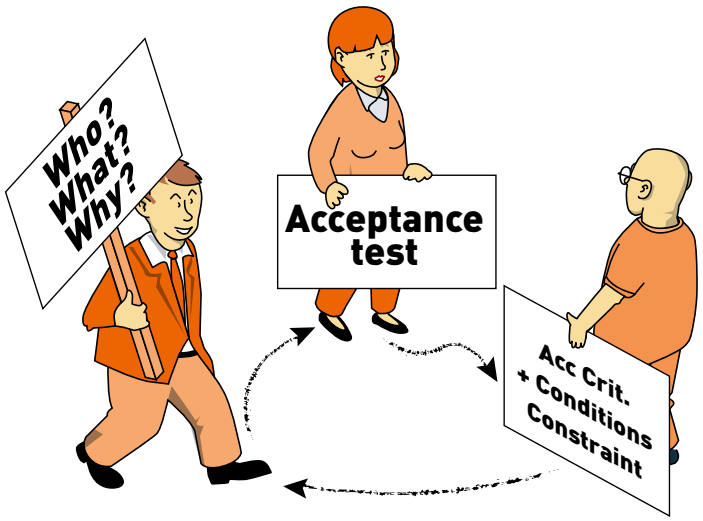
3. Then we discuss if there are any alternate paths to be demoed and what their expected outcomes are. These are our next set of acceptance criteria (or potentially other stories).

4. When we talk about test examples we realize that the initial user story needs further clarification or perhaps needs to be split into two user stories. Clarification often leads to other acceptance criteria.

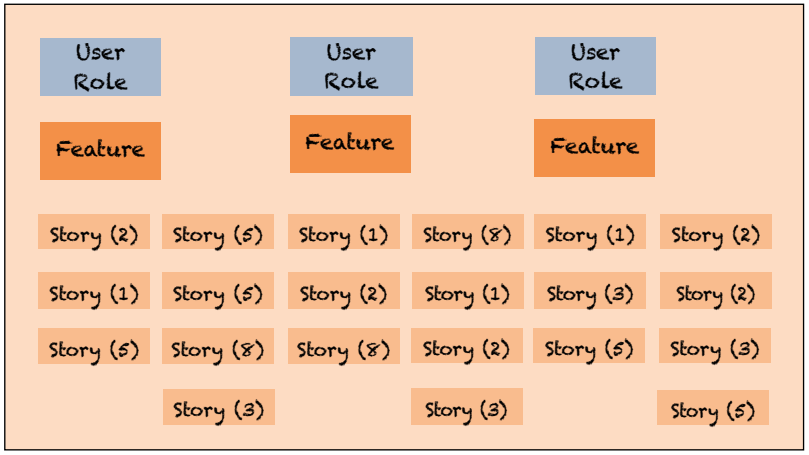
5. We also get ideas about how implementation should be done. Implementation details usually add acceptance criteria.

6. Finally we discuss whether there are any conditions or constraints that the specific user story must meet e.g. performance criteria or business rules. These become acceptance criteria and test examples as well.

7. The identified acceptance criteria, acceptance test examples and additional information are written down, either on the back of index card or on additional post-it notes.



8. Organize stories in accordance with user story map structure, i.e. place stories beneath corresponding features.



9. Estimate the effort associated with each story by means of planning poker.

   See planning poker description on page 35 if you are interested in the estimation aspects.

10. Mark stories with story point estimates

11. Prioritize which user stories add most value or are necessary to do early for technical reasons.

12. Place prioritized stories high on the map.

A note on definitions of done and their relation to acceptance criteria and user stories. Before entering iteration planning a user story should have acceptance criteria and/or test examples. How else can we know when we have fulfilled the product behavior needed to achieve a user goal?

A definition of done is the quality contract between product owners and the delivering organization. We need to agree on what we mean when we say a user story is done. Passing acceptance tests, passing various automated test builds as well as updating release notes on a product wiki can be the definition of done steps. Having met all identified and agreed acceptance criteria for the user story can also be a definition of a done item. So acceptance criteria can be part of a definition of done but a definition of done is not an acceptance criteria.

Our user story conversations not only help us capture acceptance criteria, specific conditions/constraints and omissions are also discovered. Not only have our conversations validated the user stories but the previous steps of exploration with the user story map mean that we are fairly confident that these are the right needs described.

Source	Product vision	Fractal model	User story
Who	Customer segment		User role
What	Product	User story title	User story description
Why	Key benefit & unique differentiator		Goal

A word of advice – if an electronic storage format is preferred, do this administrative task after the workshop.

As a rule of thumb, all stories above 5 story point size need further decomposition. In this loop, where we deal with an existing product and domain model, much of our user story splitting strategies were covered in our fractal modeling work and our user story mapping sessions. Our user story writing session combined with using the given-when-then format also helped us to get to an actionable state on the most high prioritized user stories in the user story map. If you want to know more specifics on user story splitting strategies, take a look at loop 1 on page 23.

The work of writing user stories and exploring acceptance criteria meant that new user stories were identified. They need to be organized into the user story map. After that it is time for validation of the map to ensure that the product, and its processes, deliver the desired outcome and meet the needs of the customer.

VALIDATE THE USER STORY MAP

We take a step back and look at the complete picture and simulate the use of the product. This should preferably be organized as a map demo with product owner and/or customer representatives available for feedback. You might also do walkthroughs with the different user roles in the map to ensure that their goals, primary activities and main user tasks are covered by the user story map horizontally and vertically with our user story candidates.

From a requirement perspective we are looking for omissions;

1. Is there anything missing or looks strange?

2. Should we change the workflow order?

3. Does that user role really need that feature?

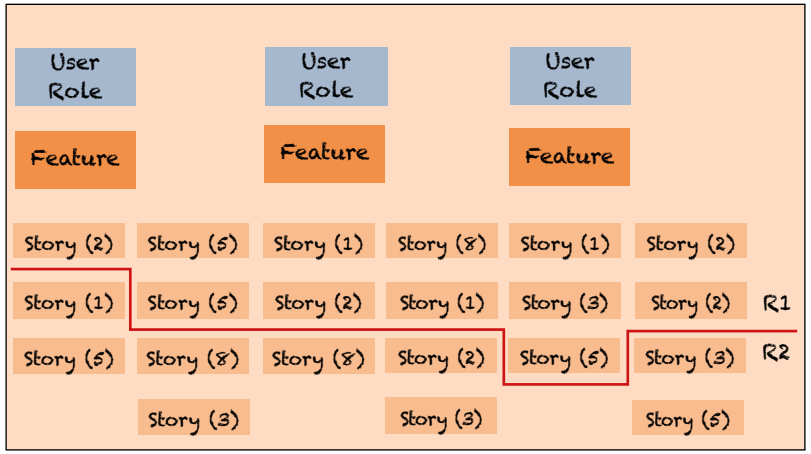
After a few simulation cycles, including changes and adjustments, we can reach an initial agreement that this is good enough to proceed.

Let's move on to release planning.

RELEASE PLANNING

We are done with the stories and we need a look-ahead plan before pulling the first stories for implementation. Release planning couldn't be easier, we basically need to tag release candidates in the user story map:

1. Draw a line across the storyboard to mark what's included in the 1st release.



2. Mark stories with 1st release.

3. Calculate the total number of story points for the 1st release

4. Divide release in iterations based on team velocity (or by guessing if no historical data exists)

5. Calculate the expected release date, based on the numbers of iteration multiplied by iteration length. Add one or two iterations depending on estimated risk.

6. Calculate expected cost based on the number of iterations multiplied by the cost for running a team for a single iteration.

7. Present time and cost estimates as an interval; actual +– risk factor of one/two iterations.

If your teams don't use numeric estimation and the numeric velocity concept you can use boxed budgeting per iteration and do a rough total budget box for the release. As the iterations progress the teams realize their capacity of delivering stories of certain sizes during an iteration and can adjust the overall budget box early on if they foresee the need for more iterations.

At this point the loop is concluded and the first iterations of the first release can start. Through cross-functional collaborative sessions you have explored your next step in the product lifecycle of an existing product. Fractal modeling and user story mapping created the active queue from which one or several teams can pull from into their iteration planning.

The user story map will contain user story candidates still in verb + object title format, even user story candidates for the first release might not be decomposed. This means that you should have a manageable amount of release content to work with and continue the refinement work during iterations. Teams can spend 5–10 % of iteration work to continually explore and refine requirements in parallel with delivery flow, or more to the point, as part of the delivery flow. They revisit the methods described in this loop as the toolbox for this ongoing refinement work. The user story map is a great help on this continued journey.

What the loop did provide was a fast and flexible modeling strategy that helped the team to explore the user narrative with a user story map. This gave just enough data to get started with enough confidence, working just-in-time and accelerated with requirements. Even more important than that, we also need to have disciplined and useful cross-functional requirement conversations that helped us explore what the right needs are – helping us deliver value.

... NOW WHAT?

How to support a steady requirement flow during the release

So far this loop has described examples of upfront agile requirement work, meaning exploration before initiating the first iteration, albeit faster than in loop 1. The time box in question fits well within a first boot strapping iteration of, say, two weeks, combining it with the setup of the development and testing infrastructure, initial mock ups and some architectural spikes needed to validate the release strategy.

Now is the point where you will start to reap the real requirement value of working in an agile delivery flow. Here are some recommendations on how to make sure that agile requirement work turns into a steady flow, not just a one-shot engaging the team early in the release.

Visualization
We recommend teams have a product board next to their task and/or Kanban board. The product board supports the team to zoom in and zoom out on product aspects, clusters of features, role definitions and other requirement related content. It is too easy to write lots of user stories, dump them into a digital tool and drown in backlog items every time the team needs to pull from the backlog.

Requirements need context in order to be understood and shared. The product board would benefit from having the elevator pitch clearly stated. Even if you use a digital tool for your backlog the product board should contain the user story map with at least the user roles, the backbone and user story titles. Use references to the digital issue numbers in your tool where actual descriptions can be stored together with other user story related data.

You can extend this visualization to illustrate critical areas touching on non-functional attention. You can also use the user story map to illustrate what areas are done and how different parts of the user story map are queued into the ongoing release or as candidates for the next one.

If you modeled using fractal modeling you can put printed photos of the different models and their fractal branches on the product board. You can use color magnets or colors as presented above concerning the user story map but you can also use the fractal models on the product board to signal what is "as-is" and what is "to-be" in a legacy environment. This allows the team to illustrate their ongoing exploration of new requests and changes that may or may not be included into the user story map – yet.

Finally we also encourage a slot on the product board showing the high level view of the release plan with time constraints. This way the team can zoom in and zoom out from a short-term iteration perspective to long term release constraints.

Refinement is ongoing
Requirements are not for free, you have to pay the piper as we said in the start of this book. We have heard people in organizations adopting agile questioning whether requirements are needed at all in an agile delivery model. For us requirements, i.e. customer needs, are at the core of agile.

This loop produces less inventory than the first loop when it comes to explored requirement content. You might end up with between 80 to 150 items of varied size and importance. The key strategy to survive is the continued refinement work that takes place during iterations:

– Refinement sessions (also called backlog grooming or backlog refinement in Scrum)
A cross-functional session can contain the entire development team together with their product owner, customer representative and/or product manager depending on which agile method you use. It could also be executed in a pure three amigo constellation (one business analyst, one developer, one tester). It can take place once or twice during the iteration and can take 1–2 hours. Be sure to keep up a steady and predictable cadence for these sessions so teams and stakeholders learn to use the opportunity given.

The main purpose of the session is to take on new requests and explore them and/or decompose already existing high level features into smaller user story formats. The participants collaborate using slices of the requirement techniques mentioned in this loop: based on fractal modeling and the actual user story map they might model the single feature, using splitting strategies, estimate and discuss acceptance criteria and test examples. They identify new explorations needed that might have to be scheduled into coming iterations before further decomposition can take place. Be careful to not just focus on validating the upcoming iteration candidate user stories. You need to take a look at upcoming high-level areas on the product board as well as being able to swiftly respond to new incoming requests.

– Requirement workshops with end-users and/or customers and other stakeholders
The development team supports the elicitation of new needs by participating or even running requirement workshops where they engage different stakeholder groups. Requirement workshops of this kind are usually about eliciting needs or validating interpretations of needs with models, user stories and mock ups. Let's not forget that the iteration planning session (sprint planning in Scrum) is an also opportunity for refining requirements. While agreeing on the scope of the iteration we might clarify acceptance

criteria and test examples for a user story, decide to split one and quickly draft and estimate new user stories and put them aside. Don't get too stuck in expecting a complete and neat decomposed candidate list of user stories for the iteration from a product owner. Iteration planning is a window for cross-functional requirement conversations. It is not a forum for handing over an order and wait for your take-out or say "talk to the hand and wait for the next sprint". We need to work more collaboratively with requirements than that.

Again – the product board can support the teams in their ongoing refinement and decomposition work as it provides an important product centric look ahead when it is too easy to get bogged down in implementation details. It is a question of cognitive survival.

Demo means validating executable requirements
With an agile delivery model the previous hard line and difference between validation and verification gets blurred. In a more conventional phase based and documentation heavy approach it is crucial to validate the requirement documents as part of the upfront requirement work. It will take a long time before someone gets to see implemented features specified in the documents. This is not the case with an agile delivery model.

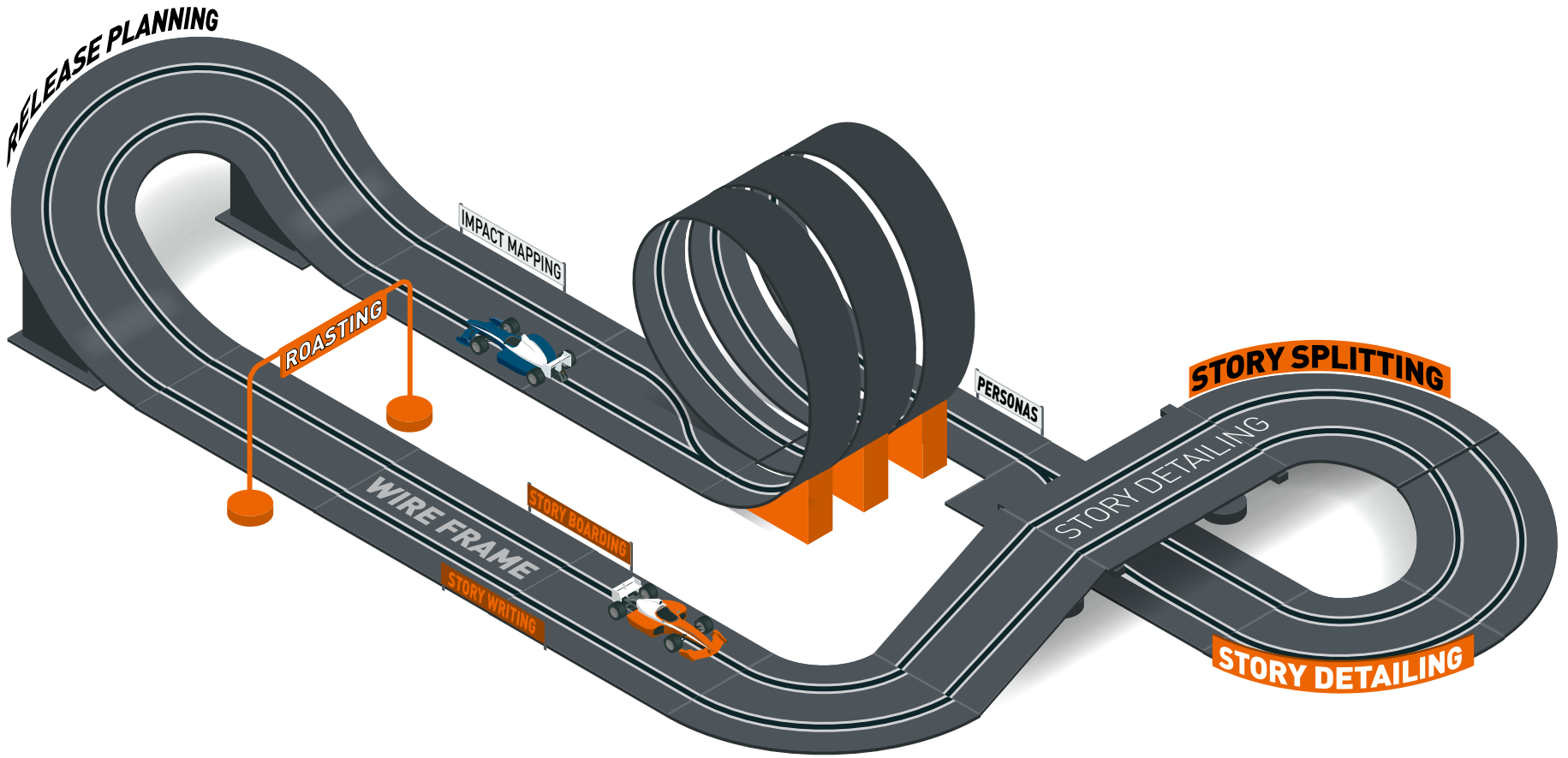
The demo (or review in Scrum) is where the teams present their produced result of value from the iteration. We encourage teams to view the demo as a requirement session, a validation of executable requirements. It should be organized as a requirement workshop where the emphasis is on validating executable requirements (that are in a done state) and elicit new needs or improvements needed on the executable requirements. Use the user story map to help users and stakeholders to understand what part of the map is being demoed so they also can zoom in and zoom out from iteration to release queue perspective.

An agile delivery model is all about the requirements and the lifecycle of a user story. How fast can you make your requirements executable?

Extract data for documents – not the other way around
An agile team working collocated would be fine keeping the release plan in this visualized format but most organizations have policies and standards pointing at templates or tools. This is especially the case in multi-team and distributed team contexts where digital tool support is a prerequisite for survival.

Most modern collaboration tools manage user stories, acceptance criteria, test cases, story points, sketches, and release/iteration labels, etc. Required documentation can easily be extracted – it is just a click away. If you require sign offs on formal paper, please do it this way and not the other way around.

"What you have demoed today exceeds my vision, *I hereby accept this delivery* and I am happy to move on to the next iteration."



LOOP 3: GO & DO!

Aim:
Prototype-driven work mode

Preconditions:
Collocated team

Example of fit:

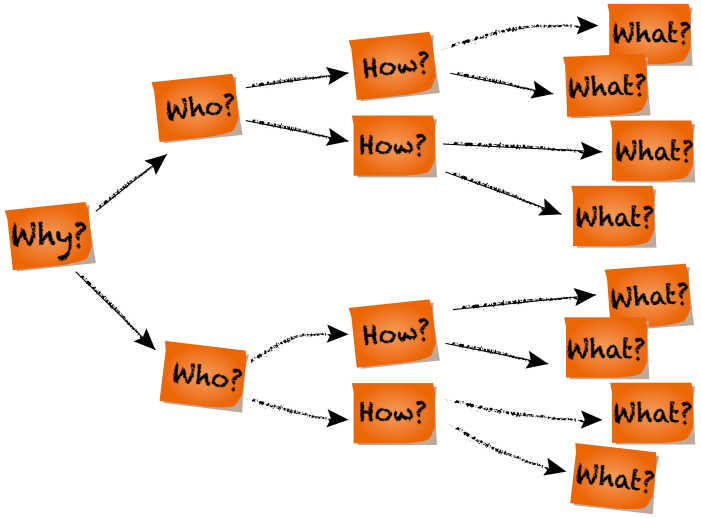
As a	cloud and app development organization
I want	an accelerated and prototype-driven way-of-working with requirements
So that I	get to the minimal product on the market with the right timing

Acceptance Criteria:
Verify that you have a prioritized prototype visualized in a storyboard format
Verify that requirement details are provided by user stories in storytelling format for contextualizing purposes
Verify that storyboard and storytelling content are validated with stakeholders external to the team

Time box:
3 working days

IMPACT MAPPING

Impact mapping was created by Gojko Adzic as a means to shift the conversation from what is needed to why we want to achieve a certain type of business goal or effect and identify the impact needed in order to reach the goal. The model is about understanding the behaviors that need to change among key stakeholders in order to achieve a sustainable result. It is a good starting point for this loop being more accelerated than the two first loops. We need to ensure that we are exploring the right value from start.



As such an Impact map is a specific mind map format, which organizes your ideas in a logical structure – deriving the minimal amount of capabilities (what) needed tied together with strategies on how to facilitate the changed behavior (how) among key stakeholders (who) that need to be impacted in order to achieve the goal (why).

- First, identify the **goal**, why is this important?
- Next, who are the stakeholders/**actors** who we need to influence and impact (who can block it or support the goal)?
- What are the actual **impacts** we need the actors to engage in as a desired change of behavior in order to progress towards the goal?
- Finally, what organizational activities and/or software capabilities, in short the **deliverables** that will facilitate the impacts being achieved?
- At this point you can focus on finding the **shortest path** through the map in order to execute the goal. Applying this strategy gives you the least "wasteful" first view of the work needed.

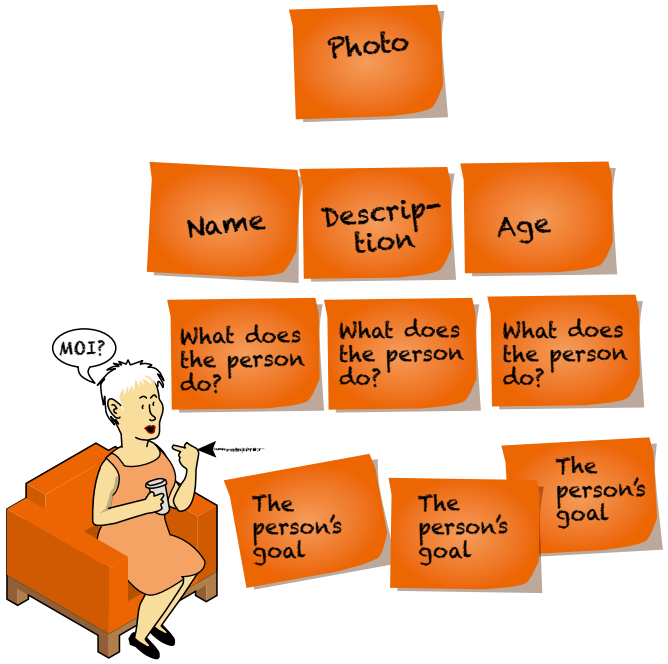
Source	Impact Map	User Persona	Storyboard	User Story
Who	Stakeholders	User Persona	User Persona	User role
What	Features		Features	User story description
Why	Goal	Goal		Goal

To make sure we are on the right track with whom, let's zoom in on our stakeholders, especially the user roles involved.

USER PERSONAS

A user persona is an archetypal description of a fictional person's characteristics, demographics and goals.

Based on the user related stakeholders in the impact map, we make 2–4 user personas illustrating their top three characteristics, demographics and goals. Once again, post-it is king.



The user personas give us a foundation for user roles and we are especially interested in their goals, which generate storyboard frames and stories.

Source	Impact Map	User Persona	Storyboard	User Story
Who	Stakeholders	User Persona	User Persona	User role
What	Features		Features	User story description
Why	Goal	Goal		Goal

A brief word of advice, user personas can also be used for UX trade-off decisions as well as global conditions and constraints prioritization.

Let's move on to storyboarding based on the goals of user personas in combination with "the whats", our main capabilities from our impact mapping session.

STORYBOARDING

A storyboard illustrates user interactions to achieve a goal. The board is written from a specific user persona perspective and it consists of a series of frames, which are visualized similar to a comic strip. Each frame shows sample data. High-level features are associated with each frame.

"Time reporter" log in	"Time reporter" time spent	"Time reporter" Review time report	"Time reporter" Clock time for work task	"Admin" Set project variables

We have experimented with different formats for describing the high-level features and capabilities presented in the image above. You can use the user story format but we want to give an example of how to express this higher level and save the user story format for the lower level features. We call this format "textual storyboards". It fits really well with the frames and will support good conversations and validations even on this high level. Here are some textual storyboard examples from the Eutaxia Control panel we worked with during loop 2:

Storyboard A. Control panel: add a free trial instance
Role: Hoster staff – in this case Sales person S

1. S is in a customer meeting, extolling the virtues of their new hosted service Eutaxia. Customer seems interested and S logs into the Control panel and starts to add an instance.

2. S quickly inserts information he receives from the customer in the meeting
   – a unique identifier for the instance (decided by S)
   – the domain name to be used for mails in/out of the system (provided by customer), e.g. helpdesk@netnetnet.eutaxia.se, helpdesk123@net-netnet.eutaxia.se
   – "the from" description to be used in outgoing emails from the system (provided by the customer), e.g. Pelle Pallesen – NetNetNet AB (help-desk123@netnetnet.eutaxia.se)
   – select language for the instance (Swedish or English)
   – admin password for the instance
   – tags the instance as free trial, the period in question and adds a refer-ence person with contact information for the instance
   – fixes the amount of users allowed (0–5, 6–10, 10–15, 15–30, 30–50, 50–100)
   – turns the instance on
   – saves the information he inserted

3. S opens a browser and navigates to http://netnetnet.eutaxia.se and logs in as the admin user with the admin password. The customer receives a quick product demo by S and they agree to try it out for the free trial period and to have a meeting about the decision to buy or not.

4. S is a crafty sales person and creates a case in the customer free trial system about this decision meeting and emails from the instance to the people that will be in the meeting.

Storyboard H. Control panel: looking for more business
Role: Hoster staff – in this case sales person S.

1. Sales person S is trawling for more hosted service business, he logs into the Control panel to check out the status of the instances.

2. S looks at the list of all instances, it is long.

3. He looks at the summary and sees "total: 93, 79 on, 14 off"

4. S looks at the list again which is sorted alphabetically, the colors varying for instances that are on or off. To him it seems a mess.

5. He sorts by status (off) and get a list view of 14 instances, all the color of X.

6. S trawls down the list of instances, checks some info in their separate CRM system and realizes he can follow up on at least 5 of these during the upcoming week.

7. He logs out of the Control panel and starts to turn his follow-up plans into action.

Storyboard G. Control panel: remove instance
Role: Hoster staff – in this case sales person S

1. S gets a notification from the Control panel via email at the end date prompting a removal of Customer G

2. S logs into the Control panel, and scrolls in the list of instances for Customer G and finds them.

3. He starts to remove the instance:
   – sets the removal date (immediately or a date in the future)
   – tags the instance for a back up of data before removal
   – writes the reason
   – saves the changes

4. Customer G reference person receives an automatically generated confirmation email informing about the date the service will removed and inaccessible, and that they can retrieve their data for up to X months after removal.

This textual storyboard format describes the high level features (titles in verb + object format) needed. They also provide the business context and flow that helped to build the frame-based narrative flow in a different way than a user story map would. As such storyboarding, especially this combination of frames with textual storyboards, is a smooth next step from the impact map because it builds on the main capabilities identified in the impact map.

As soon as we are happy with the storyboard, user interface design gets on our radar. Why? Is it not too soon? Is it not too solution oriented? No, we have confidence in having caught the most crucial high-level feature needs for now. We trust that the upcoming iterations in the release will give us further refining opportunities when needed as well as learning and feedback.

Source	Impact Map	User Persona	Storyboard	User Story
Who	Stakeholders	User Persona	User Persona	User role
What	Features		Features	User story description
Why	Goal	Goal		Goal

WIREFRAME

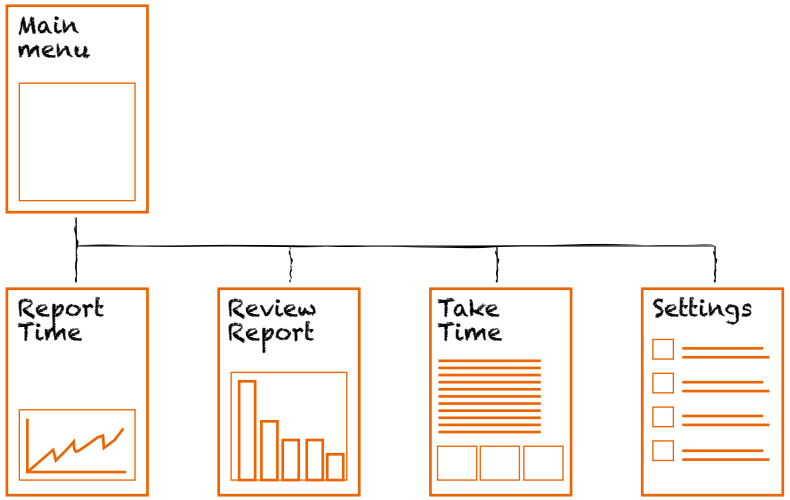
A wireframe is a rudimentary skeleton of an interface. These are specifically designed to understand the space and structure of the website or app and are primarily aimed at capturing usability and functionality.

Wireframes are quick to create as they only display the web site's framework with little or no details, color or graphics. We prefer to draw them on a white board and take pictures for sharing and storage but there are great tools out there for fast prototyping with wireframes.

Each textual storyboard scenario with its related frames is explored by drafting wireframes to visualize the design discussions and decisions. This way we can validate our high-level requirements while achieving a high-level UX strategy as well as getting a fast feasibility validation.

Wireframes have helped us to capture usability and save us the step of more detailed mock-ups. We let detailed user interface design evolve during implementation while refining our understanding of the needs. Let's go back to the storyboard backbone and proceed with feature decom-

position. We now have the rough implementation strategy in the form of wireframes to support that decomposition rather than taking it the other way around.



STORY WRITING

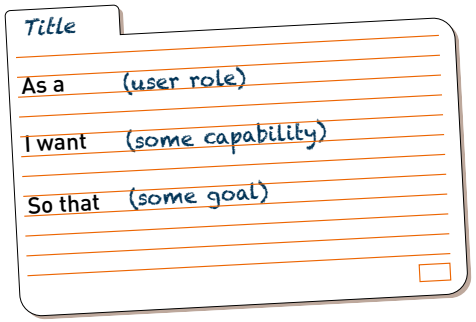
A user story is written from a user-centric perspective and it describes:

who
wants **what**
and **why**

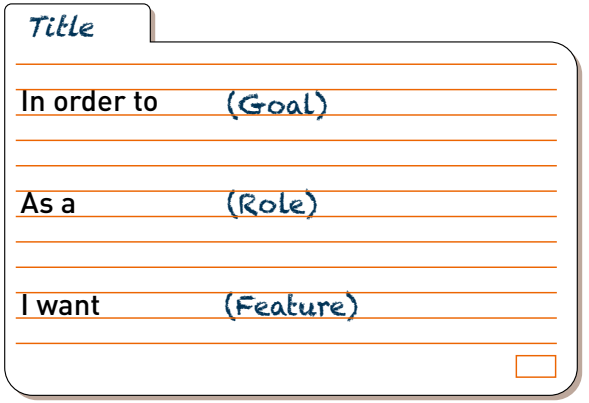
The two most important pieces of advice regarding user stories that point to their XP origin is:

– "A user story is a reference point for a conversation"
– "Card, conversation, confirmation"

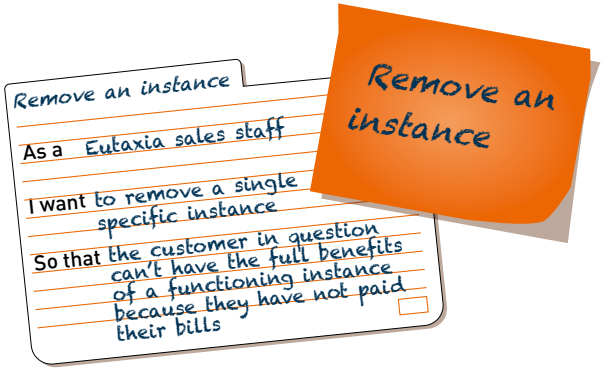
The user story title works as a short summary, it is not a requirement. It is a small token that supports requirement conversations during the 1–2 week long iterations typical for XP:



Another format is



At this point we turn our textual storyboards (verb + object title) into user stories simply by adding a description in accordance with the format above telling who, wants what and why.



Here is some advice we want to share regarding our experience in working with user stories and their format:

– There are three main elements to a user story in both examples above. Choose whichever format that suits you but don't skimp on these elements. We see too many user stories that don't contain goals at all.

– We have stressed using a clear verb + object title of the story as a way to delay story writing until it is clear that it will provide value in the near future. This is the "I want" part and we think it is important to be as clear and precise as possible on the verb and object, it makes it easier to derive test examples.

– The most important part of the user story format is the goal. The goal is not a synonym of the verb + object, it is the real need. The feature or "I want" is just a means to an end, nothing more. If you have difficulties pinning down the goal then don't skip it. It is a good signal that further exploration or postponing is needed.

– Be precise on the role. Very often teams still get stuck in the too lazy habit of stating the role as "user". This makes the user story more ambiguous. Am I a surfer, evaluator, premium customer or a multiservice customer? I will have different goals and needs and user stories need to be tested based on these specifics. There are probably functional gaps lurking as well because of not differentiating the roles properly.

– When we encounter teams working with pseudo stories (or fake stories as Gojko Adzic and David Evans call them in their new book Fifty quick ideas to improve your user stories) and technical stories we work hard to identify the reasons why teams use these formats and try to help them find the root cause instead.

– Not everything should be written in user story format. We feel it fits best for capabilities, not conditions or constraints (usually called non-functional requirements). Capabilities are observable functions someone interacts with and non-functional requirements are cross-cutting concerns touching on all or clusters of user stories.

– Epics are large user stories, usually too large to fit into a single iteration. It is still a user story and is still expressed in user story format. We usually just talk about large or small stories or other T-shirt sizes. Many teams we have worked with have been confused about the term and its meaning so we rarely use it any more.

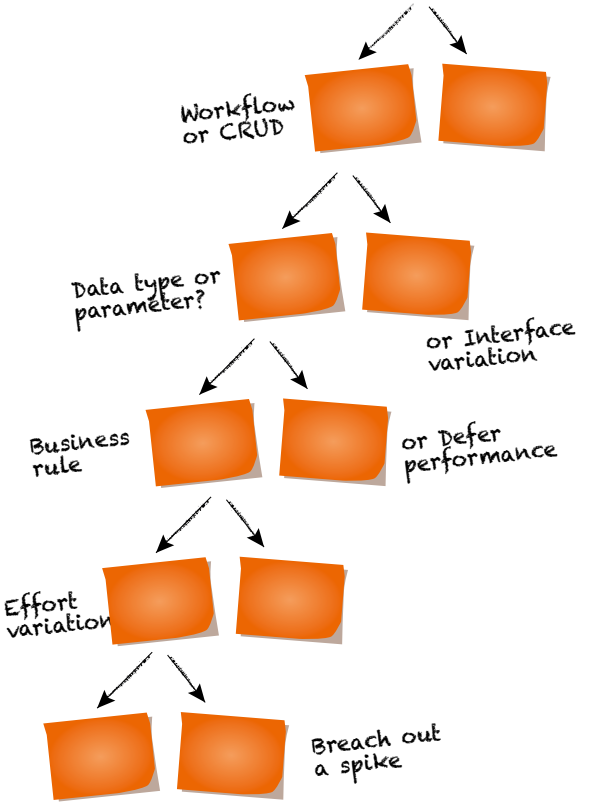
– You do not have to use the user story format if you don't want to. Please make sure you still write precise verbs + objects titles and differentiate between capabilities where users execute actions. Here you need to connect the subject and/or actor or role to the capability so that you are still user-centric. Add a description to the title. We recommend the user story format for describing capabilities. All the three key elements are there.

User stories were initially written, processed and kept on index cards. Nowadays, many teams manage their user stories electronically in different types of backlog and agile planning/tracking tools. In workshops and during loop work we strongly advise working using index cards, or post-it notes, to make everyone attending the workshop deeply involved in the process and able to contribute in an easy way. The goal is to get everyone writing stories, and learn that it is great fun.

The format above is only half the story; a complete story contains significantly more information, where the most important is the acceptance criteria. We are not there yet. Our user stories are too big and need to be split.

STORY SPLITTING

The general guidance on splitting user stories favor vertical slices before horizontal slices, e.g. splitting stories through all architectural layers. In the image below you can see the most common splitting strategies for user stories.



We want to have refined user stories so that they can be pulled into the iterative delivery flow, we want them to be actionable. The INVEST model serves as a high-level definition of ready, when a user story is actionable:

- Independent. Reduced dependencies = easier to plan
- Negotiable. Details added via collaboration
- Valuable. Provides value to the customer
- Estimable. Too big or too vague = not estimable
- Small. Can be done in less than a week by the team
- Testable. Good acceptance criteria and test example

In our experience you can gain a lot by focusing on splitting stories based on operations (too large verbs), based on data (too large objects) first because they are directly related to the requirement work – the quality of the conversation, finding the functional gaps, or the too large holes in the Swiss cheese. Splitting based on deferring performance and even by business rules, etc., are more about finding value and/or technical implementation strategies. The challenge here is not to get stuck in decomposition, slipping into inventory creation and story splitting gold plating. Please remember the following:

– Every user story has to provide value, we are not shipping crippleware just because it was a more comfortable way to deal with implementation difficulties. By crippleware case we mean both delivering the wrong thing in the wrong way as opposed to delivering the right thing, right and fast.

– Decomposition will only take you so far. Do not get fooled into thinking that breaking down a problem into its component parts and then aggregating those parts will yield the original problem (or feature). It still has to make sense from a business and user perspective and it still needs to be worth it from a technical implementation perspective.

Having split and written the user stories coming from the textual storyboard scenarios, we can now return to the storyboard and add our user stories. We can cluster our textual storyboard scenario with related frames and wireframes together with their user stories. This gives good context. We recommend story splitting workflow (page 23) for further reading, but practical experience is really what counts. Just do it!

What makes a good split is dependent on iteration length, integration, build and deployment process. Smaller stories are always better but require continuous integration/deployment opportunities.

ROAST THE STORYBOARD

We take a step back and look at the complete picture and simulate the use of the product. We are looking for omissions, wrong design and potential issues with look and feel;

1. Is there anything that looks strange?

2. Should we change the workflow, logical order?

3. Is the user interface easy to use?

After a few simulation cycles, including changes and adjustments, we all agree that the storyboard is actionable. Now we need to revisit the user stories and make sure that the ones we want to pull first into iteration planning become actionable.

STORY DETAILING

The user stories created so far are only half the story; a complete story contains significantly more information, where the most important thing is the acceptance criteria. Because it is hard to pick acceptance criteria out of the air, we need a better, methodical approach.

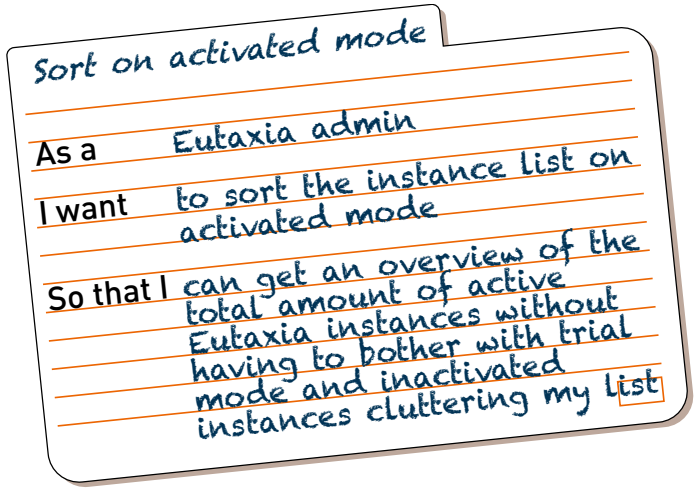
Acceptance criteria can be explored with the help of the Behavioral Driven Development (BDD) format, describes an implemented user story's expected behaviors. It matches perfectly with the user story technique and if test data is included it makes the outcome crystal clear.

Given some initial context (the givens)
And to connect multiple givens
When an event occurs
Then ensure some outcomes
And to connect multiple outcomes

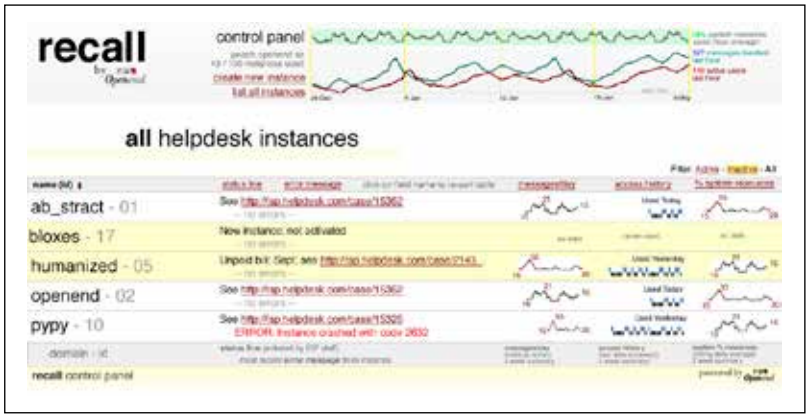
It is great value even if you just use the given-when-then format and don't proceed with the automated acceptance tests that are part of BDD although we recommend doing that as well if possible.

The acceptance criteria can be written as such on the back of the index card of the story. It can also be formulated as a test example, or an actual small customer facing acceptance test.

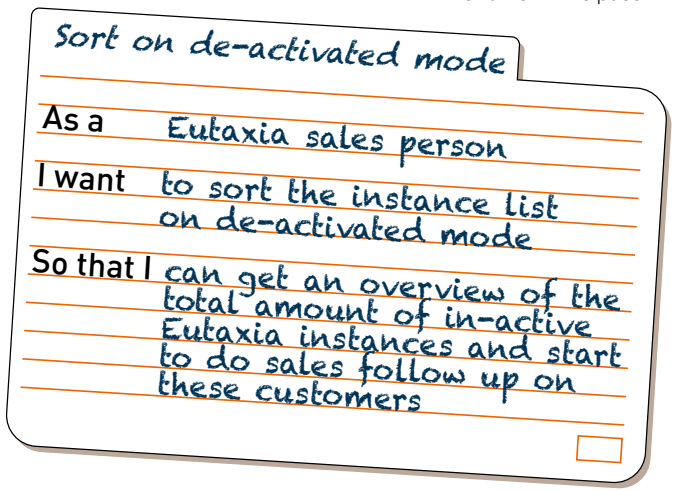
Here are some examples that show user stories coming from the textual storyboard scenarios being explored with the given-when-then format:



Given the list of instances is sorted alphabetically (as default)
And the list consists of more than two instances
And the instances have different activation modes (on is green, off is yellow)
When the admin filters the list after activation mode "on"
Then the list should instantly rearrange to only show active instances
And the list of active instances should be sorted alphabetically
And the list should only show instances being green
And the list total should be same number as the list summary says



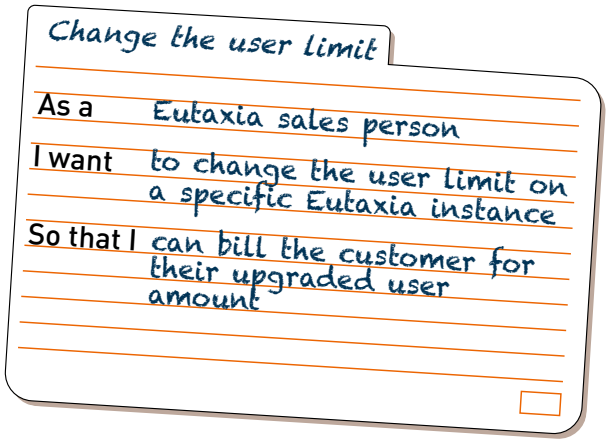
Wireframe: All helpdesk instances



Another Eutaxia Control panel example of given-when-then as a good way to derive acceptance criteria:

Given the list of instances is sorted alphabetically (as default)
And the list consists of more than two instances
And the instances have different activation modes (on is green, off is yellow)
When the hoster staff filters the list after activation mode "off"
Then the list should instantly rearrange to only show de-activated instances
And the list of de-activated instances should be sorted alphabetically
And the list should only show instances being yellow
And the list total should be same number as the list summary says

And one more example:



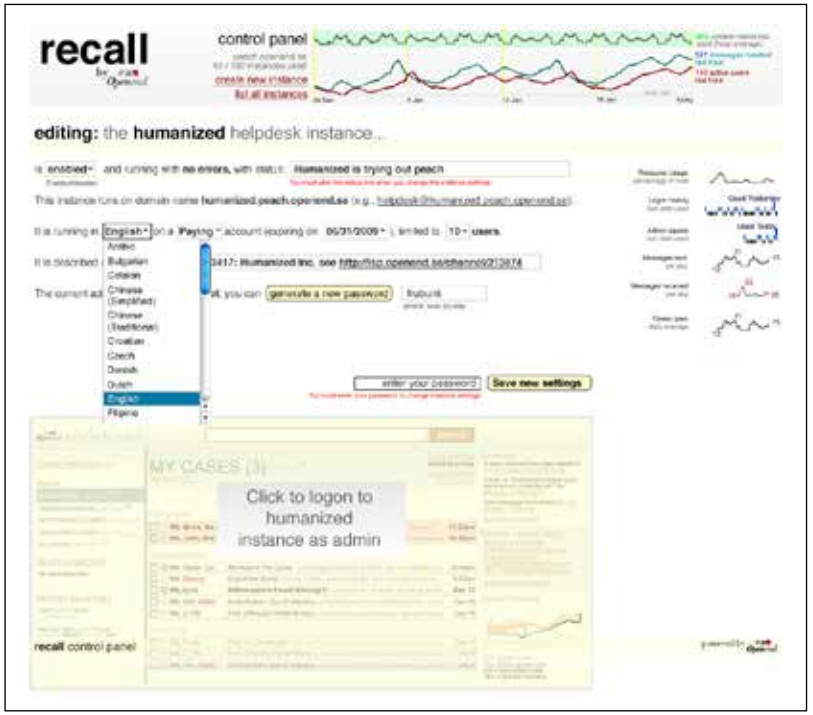
Given an active instance
And the current user amount is 0–5
When the sales person sets the user limit to 0–10 users
And saves the change
Then user 6, 7, 8, 9, 10 can be created from within the instance
And the instance reference person receives an automated email with info on the update
And the billing department receives an automated email with info on the update
And the next invoice should be based on the updated user limit

The beauty of this format is that it gives us the necessary preconditions (given), the action we want to execute (when) as well as the post-conditions that should appear as consequences of the executed action (then).

A word of advice – if this format feels difficult to work with, just use test examples as promoted in Specification by Example (Gojko Adzic). The outcome will be roughly the same – your user story will be measurable and testable.

1. Then, we discuss if there are any alternate paths to be demoed and what their expected outcomes are. These are our next set of acceptance criteria (or potentially other stories).

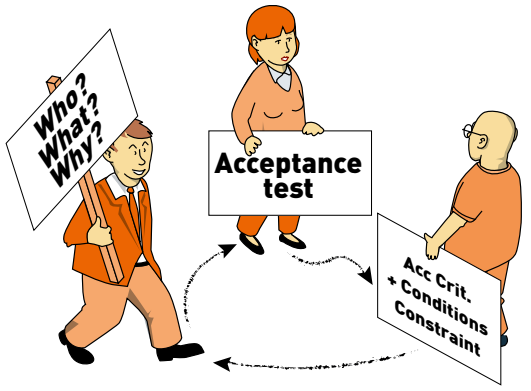
2. When we talk about test examples we realize that the initial user story needs further clarification or perhaps needs to be split into two user stories. Clarification often leads to yet other acceptance criteria.



Wireframe: editing

3. We also get ideas on how the implementation should be done. Implementation details usually add acceptance criteria.

4. Finally we discuss whether there are any conditions or constraints that the specific user story must meet e.g. performance criteria or business rules. These become acceptance criteria and test examples as well.



5. The identified acceptance criteria, acceptance test examples and additional information is written down, either on the back of index card or else on additional post-it notes.

6. Organize stories in accordance with user storyboard structure, i.e. place stories beneath corresponding feature.

7. Estimate the effort associated with each story by means of planning poker.

   See planning poker description on page 35 if you are further interested in the estimation aspects.

8. Mark stories with story points estimates

9. Prioritize which user stories that add most value or is necessary to do early for technical reasons.

A note on definitions of done and their relation to acceptance criteria and user stories. A user story, before entering an iteration planning, should have acceptance criteria and/or test examples. How else can we know when we have fulfilled the product behavior a user needs in order to fulfill his or her goal?

A definition of done is the quality contract between product owners and the delivering organization. We need to agree on what we mean when we say a user story is done. Passing acceptance tests, passing various automated test builds as well as updating release notes on a product wiki can be definition of done steps. Having met all identified and agreed acceptance criteria for the user story can also be a definition of done item. So acceptance criteria can be part of a definition of done but a definition of done is not an acceptance criteria.

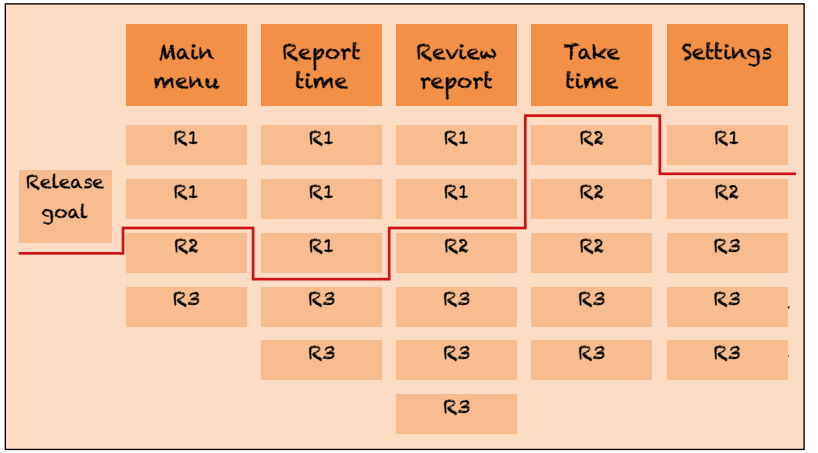
Our user story conversations not only help us to capture acceptance criteria – specific conditions/constraints and omissions are also discovered. Not only has our conversations validated the user stories but the previous steps of exploration with the storyboard steps means that we are fairly confident that these are the right needs described.

Source	Impact Map	User Persona	Storyboard	User Story
Who	Stakeholders	User Persona	User Persona	User role
What	Features		Features	User story description
Why	Goal	Goal		Goal

PULLING FROM THE STORYBOARD

We are done with the user stories and have an actionable user storyboard. We now need a look-ahead plan before pulling the first stories for implementation. Release planning couldn't be easier:

1. Draw a line across the storyboard to mark what's included in 1st release.



2. Mark stories with 1st release.

3. Calculate the total number of story points for 1st release.

4. Divide release in iterations based on team velocity (or by guessing if no historical data exist).

... NOW WHAT?

How to support a steady requirement flow during the release

This loop has the least amount of upfront agile requirement work, meaning exploration before initiating the first iteration. For a lot of organizations this strategy looks like risky business. For others, this loop is the only way to survive in a rapidly changing global online market.

Risky business or not. Now is the point where you will reap the real requirement value of working in an agile delivery flow. Here are some recommendations on how to make sure that agile requirement work turns into a steady flow, not just a one-shot engaging the team early in the release. In the case of this loop it is a vital and necessary part of delivery flow because a more accelerated and hands on strategy was chosen.

Visualization
We recommend teams have a product board next to their task and/or Kanban board. The product board supports the team to zoom in and zoom out on product aspects, clusters of features, role definitions and other requirement related content. It is too easy to write lots of user stories, dump them into a digital tool and drown in backlog items every time the team needs to pull from the backlog.

Requirements need context in order to be understood and shared. The product board would benefit from having an overview of the impact map for recall reasons, a photo can many times be enough to jog the memory. The impact map is the main justification for the strategy chosen – the development team and involved stakeholders would do well to revisit it regularly.

The product board is also where your user storyboard with its feature clusters of user stories, wireframes and maybe even the textual storyboard scenarios stays visible and available. You can extend this visualization to illustrate critical areas touching on non-functional attention as well. It can also contain user persona overviews which help the team to use the personas as intended, to arbiter different user needs during development and high light the "voice of the customer".

Finally we also encourage a slot on the product board showing the high-level view of the release plan with time constraints. This way the team can zoom in and zoom out from short term iteration perspective to the long term release constraints.

Refinement is ongoing
Requirements are not for free, you have to pay the piper as we said in the start of this book. We have heard people in organizations adopting agile questioning whether requirements are needed at all in an agile delivery model. For us requirements, i.e. the customer needs, are at the core of agile.

In this loop we have very little requirement inventory to talk about. The key strategy to survive this risk taking is the continued refinement work that takes place during the iterations:

– Refinement sessions (also called backlog grooming or backlog refinement in Scrum)

A cross-functional session can contain the entire development team together with their product owner, customer representative and/or product manager depending on which agile method you use. It could also be executed in a pure three amigo constellation (one business analyst, one developer, one tester). It can take place once or twice during the iteration and can take 1–2 hours. Be sure to keep up a steady and predictable cadence for these sessions so teams and stakeholders learn to really use the opportunity given.

The main purpose of the session is to take on new requests and explore them and/or decompose already existing high level features into smaller user story formats. Participants collaborate using slices of the requirement techniques mentioned in this loop: they might wireframe the single feature, using splitting strategies, estimate and discuss acceptance criteria and test example. They identify new explorations needed that might have to be scheduled into coming iterations before further decomposition can take place. Be careful not to just focus on validating the upcoming iteration candidate user stories. You need to take a look at upcoming high level areas on the product board as well as being able to swiftly respond to new incoming requests.

– Requirement workshops with end-users and/or customers and other stakeholders

The development team supports the elicitation of new needs by participating or even running requirement workshops where they engage different stakeholder groups. Requirement workshops of this kind are usually either about eliciting needs or validating interpretations of needs with models, user stories and mock ups.

Let's not forget that the iteration planning session (sprint planning in Scrum) is also an opportunity for refining requirements. While agreeing on the scope of the iteration we might clarify acceptance criteria and test examples for a user story, decide to split one and quickly draft and estimate the new user stories and put them aside. Don't get too stuck in expecting

a complete and neat decomposed candidate list of user stories for the iteration from a product owner. Iteration planning is a window for cross-functional requirement conversations, it is not a forum for handing over an order and wait for your take-out or say "talk to the hand and wait for the next sprint". We need to work more collaboratively with requirements.

Again, the product board can support the teams in their ongoing refinement and decomposition work as it provides an important product centric look ahead when it is too easy to get bogged down in implementation details. It is a question of cognitive survival.

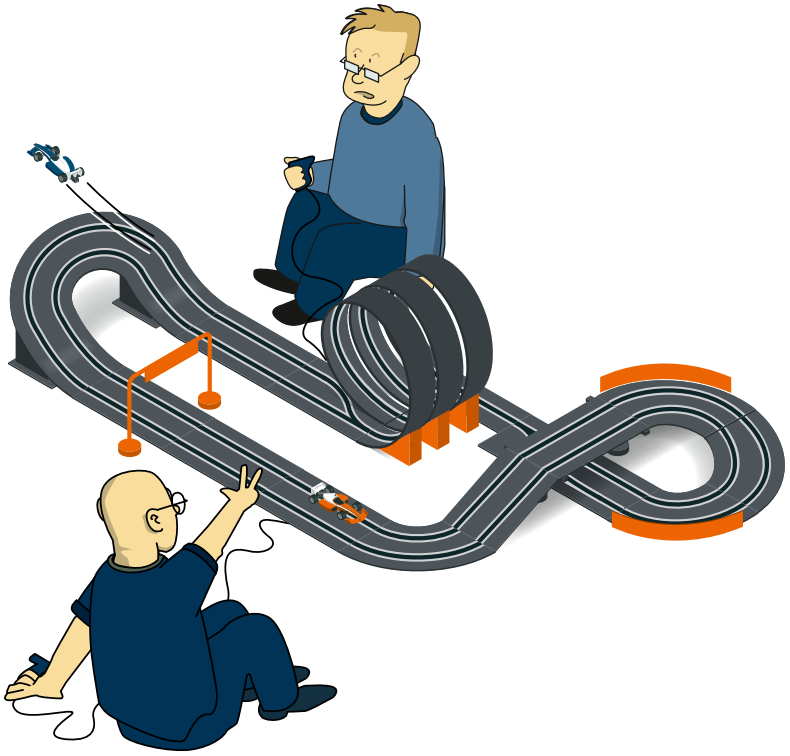
Demo means validating executable requirements
With an agile delivery model the previous hard line and difference between validation and verification gets blurred. In a more conventional phase based and documentation heavy approach it is crucial to validate the requirement documents as part of the upfront requirement work. It will take a long time before someone gets to see implemented features specified in the documents. This is not the case with an agile delivery model and definitely not in this loop.

The demo (or review in Scrum) is where the teams present their produced result of value from the iteration. We encourage teams to view the demo as a requirement session, a validation of executable requirements. It should be organized as a requirement workshop where the emphasis is on validating the executable requirements (that are in a done state) and elicit new needs or improvements on the executable requirements.

An agile delivery model is all about the requirements and the life cycle of a user story. How fast can you make your requirements executable?

Extract data for documents – not the other way around
An agile team working collocated would be fine keeping the release plan in this visualized format but most organizations have policies and standards pointing at templates or tools. This is especially the case in multi-team and distributed team contexts where digital tool support is a prerequisite for survival.

Most modern collaboration tools manage user stories, acceptance criteria, test cases, story points, sketches, and release/iteration labels, etc. Required documentation can easily be extracted – it is just a click away. If your require sign offs on formal paper, please do it this way and not the other way around.



You need to adjust the practices, order, combination and *speed of execution* to fit into your context.

FURTHER READING & ACKNOWLEDGEMENTS

We hope you have enjoyed reading about our 2 cents on how to combine different agile requirement techniques. Remember, the loop approach of three main agile requirement strategies that we play with in this book are three out of many. You will pick and choose and put together the loop you need, we wish you the best of luck.

If you are interested in more hands-on support on how to execute the techniques in the loops we are continually updating the blog with our workshop materials for you to use free of charge. Hopefully these workshop designs will make it easier for you to get started.

And a final note, if you thought we were a bit skimpy on the details regarding how multiple teams coordinate their requirement work and development in general then you are correct. Those details are so important and crucial that they deserve their own "... in 60 minutes"! Keep a lookout on the blog for the upcoming "Scaled Agile in 60 minutes". We aim for a release of the next book during 2015.

Thanks and salutations ...
Firstly, we would like to thank Softhouse Consulting AB and the Scalare EU project for letting us spend work time on this instead of our evenings and weekends which we initially had planned to do. (our families are grateful too!)

Secondly, our salutations go out to the following people who have inspired us:
- James Shore and Diana Larsen
- Henrik Kniberg
- Gojko Adzic
- Jeff Patton
- Alistair Cockburn
- Thiagi
- Eric Ries
- Always: Fred Brooks and Gerald Weinberg
- and many more ...

ABOUT THE AUTHORS

Bea Düring and Håkan Kleijn are Agile coaches working for Softhouse Consulting AB in Sweden.

Bea is a high school teacher who accidentally dropped into the IT business. As a consultant manager she read eXtreme Programming explained, got hooked and became an evangelist of XP techniques. She has been coaching large and small-scale agile adoption with focus on change management and Agile requirements.

Håkan, MSc in chemistry, started a career focused on quality in the pharma industry. Beginning with ISO, he has fallen for every hype ever since: CMMI, RUP, UML, ITIL, Lean software development, Six Sigma, eXtreme Programming, Scrum and Kanban. He has been coaching small-scale and company-wide agile adoptions, as well as being globally responsible for software configuration management.

Please visit us at:

http://en.softhouse.se/info

http://softhouseeducation.se/shop

Twitter: @2bea73

beatric.during@softhouse.se
hakan.kleijn@softhouse.se

Softhouse Consulting

Stockholm
Tegnérgatan 37
SE-111 61 Stockholm
Phone: +46 8 410 929 50
info.stockholm@softhouse.se

Göteborg
Kungsgatan 19
SE-411 19 Göteborg
Phone: +46 31 760 99 00
info.goteborg@softhouse.se

Malmö
Stormgatan 14
SE-211 20 Malmö
Phone: +46 40 664 39 00
info.malmo@softhouse.se

Karlskrona
Campus Gräsvik 3A
SE-371 75 Karlskrona
Phone: +46 455 61 87 00
info.karlskrona@softhouse.se

Växjö
Willans Park 3
SE-352 30 Växjö
Phone: +46 455 61 87 00
info.vaxjo@softhouse.se

www.softhouse.se

SOFTHOUSE