10-Step model overview

The 10 Step Model shown below outlines a framework to position the work of requirements engineers (be they Business Analyst, Product Manager, Product Owner or someone else) relative to development. The model may be considered a process, a checklist or just an aide-memoire. The model attempts to relate various aspects of Agile requirements analysis advocated by different authors. Specifically the model looks to place User Stories - the most prevalent requirements approach in Agile - into a context.

The model assumes the classic Agile iteration or sprint cycle - time-boxed development episodes together with the product backlog / sprint backlog mechanism defined in Scrum (Schwaber and Beedle 2002) and XP (Beck 2000). These mechanisms can be seen on the right of the diagram. Since much has been written about this cycle already this description will focus on the wider requirements process.

Each step in the model will be executed multiple time, anyone executing the model as their development process would iterate through the model again and again - but different groups will iterate through different parts of the model, potentially on different rhythms. Some of the steps change their focus over time.

For example, the first pass of the stakeholder step may primarily be concerned with stakeholder identification and prioritisation, the second time through the focus may be on stakeholder needs, the third time on stakeholder objectives and then it might be time to review the stakeholder list all over again.

The backlogs

Two backlogs sit in the middle of the process: the product backlog and sprint backlog (to use Scrum terminology). These serve as temporary repositories.

The Product Backlog contains all the items that might be developed for the product. These are typically written in User Story format and may be considered development stories in their own right or epics, i.e. large stories which will be broken down into multiple smaller stories.
Figure 1: 10 Step Model
In Xanpan product backlog items are typically written on blue cards - and frequently referred to as blues.

At the start of each iteration a few blues are selected from the product backlog for development in the next iteration. If helpful these are broken down into a number of work tasks - typically written on white cards and, unsurprisingly, frequently called whites.

These whites - and the few blues - then constitute the **Iteration Backlog** or, to use Scrum terminology the **Sprint Backlog**.

The iteration backlog is the focus of the development teams work during the iteration - shown in the diagram below. As described elsewhere it may be supplemented by additional items which arise during the iteration and in all likelihood some of the backlog items will remain undone at the end of the iteration.

The requirements engineers/product owners may well work with the development team during the iteration to answer questions about iteration items, they may be involved in refining specifications or writing acceptance criteria, or even tests, during the iteration.

Requirements engineers will will also pay attention to the product backlog and the work which might be arising in the near future. Thus they are more concerned with the earlier steps in the model - shown below.

In some cases different people may focus on different backlog, e.g. when the SPO/TPO model is in use the Tactical Product Owner will spend more time in the iteration with the development team while the Strategic Product Owner will be more concerned with the Product Backlog and understanding customers/users.

Overtime the focus of requirements work will change, exactly how it will change depends on the environment the team is working in. Possibilities include:

- In a stable environment initially requirements work may focus on discovering requirements. Once the bulk of requirements are identified and placed in the backlog discovery slows and requirements engineers focus on specification and detail, in effect switching from breath to depth.
Figure 2: Development teams area of concern
Figure 3: Requirements teams area of concern
In a rapidly changing environment discovery work is ongoing, over time the team may focus on apportioning value to proposed items in order to reduce the potential work to do.

There are those who would argue that these backlogs should not exist and they have a point. However in most organizations - whether a full corporation or a loose grouping of individuals - there are ideas about what could, should, or might be done in future. Allowing these ideas to be captured and stored somewhere can be helpful - even if it is only a theoretic release.

Perhaps the ultimately metric of true Agility is the time it takes for an idea to move from “An idea” to a working, delivered, product in the hands of a customer. Pursuing this metric any time spent sitting in a backlog would be minimised.

However few organizations are ready to embrace this metric in the extreme. For these organizations some repository of “candidate ideas” is useful. Indeed, such a repository may also play a role in helping smooth (level) the flow of work into a team because they allow work to be held for a period of time.

Considering the requirements and delivery efforts as two parallel activities synchronised by the backlog allows each group to work at different pace. The groups will need to interact, indeed some requirements engineers will need to work closely with the development group, but decoupling the two groups can be a useful option.

The Steps

**Objective**: the objective is given from outside the model - usually from higher up the management chain. It is the reason the team are brought into being, the reason the project is started, it is the goal the work is aimed towards. (See *Time for Goal Driven Projects* (Kelly 2010a).)

**Stakeholders**: stakeholders are those people, and groups of people, who have some interest in the work being undertaken. Stakeholders have their own objectives for the work which might, or might not, align with the objective. Some stakeholders have more stake than others, and some are more significant than others.
This step is not confined to stakeholder identification, this step includes analysis of stakeholder “stake”: what stakeholders want from the system, the constrains they impose, how it will create value for them and more.

A stakeholder’s “stake” in the system - the thing they want from the system - constitutes an objective in its own right. Hopefully this will be congruent with the overreaching objective however stakeholders might hold objectives which are slightly different. Stakeholders may even hold objectives which conflict with other stakeholders or hinder the overarching objective.

The stakeholders group includes more than just customers. To start with stakeholders can be split into two large groups: internal stakeholders and external stakeholders. Within corporate IT departments the former will be the larger group while in software companies the latter will be the larger group.

Product Vision: while the objective is owned by the powers that created the team the product vision is created and owned by the team itself. The vision both expands on the objective and answers the objective. If the objective specifies a problem that needs solving the vision gives an answer.

The product vision needs to be distinguished from the corporate vision, or indeed any other vision within the organization. The product vision is concerned with what the product will be: Will it be a website? An iPhone app? Will it be used by many? Or by few? How will it address the original objective?

For a small company, a startup or one with only one product, the corporate vision may be little more than the product visions. But in a company with multiple products each product will have its own vision which should align with the corporate vision.

A product vision sets out overarching ideas of the thing to be built without being excessively technical or detailed. It can only be formulated when stakeholder needs and wants are understood.

Roles: narrowing the stakeholder base to roles places the focus on those who will actually interact with the system as envisaged by the vision. It is role holders who interact with the system and thus their needs that need to be considered when determining functionality.

User Stories explicitly ask the writer to specify the role for whom the story is for. While this is itself valuable and useful it can lead to an excessive emphasis
on usable features rather than non-functional or performance characteristics of the software. Further highlighting the users of the software can de-prioritize stakeholder needs and business value.

**What about other systems?**

A question that regularly comes up when writing User Stories concerns how to write a story when the “who” is another system. For example: “As the accounting system I would like to know when the payment system has made a payment so that I can update balances.”

On the whole I would rather stories were not written about other systems. It is rather hard to discuss requirements with a computer system and even harder to have a system estimate value. Ultimately there will be some individual or group who will obtain benefit from the combined work of the two systems.

Thus, the above example might become: “As the company accountant I would like the accounting system to be updated whenever a payment is made so that balances are correctly shown.” Now there is someone who can discuss details and value.

However sometimes this approach can over complicate stories and may be of little value when the ultimate beneficiary knows little about the system. In general I recommend avoiding stories about other systems but I am prepared to accept them when they simplify work and some effort has been made to understand the ultimate beneficiary and the value concerned.

While it might make sense to have another system listed as a Role for a User Story it never makes sense to write a Persona for another system. Such Roles are also an exception because the ultimate beneficiary, not the system, will probably be listed as the Stakeholder.
**Personas:** personas expand and elaborate certain roles, adding texture so requirements analysts, user design specialists and software developers can better understand and empathise towards those who will use the system. Not all roles will be developed into full personas, and different personas will come to the fore at different times.

While individual personas should not be based on individual, specific, users, personas should be based on an aggregated understanding of customers and typical users. Writing realistic personas requires contact with actual customers and market research.

As analysis proceeds from stakeholders through roles to personals there is a natural narrowing shown below.

Creation of personas is not as common as stakeholder and role identification. While it might make sense to skip personas on small efforts when there are a limited numbers of actual users they have a number of benefits.

- Using personas helps the development team recognise that the typical user is not like themselves, i.e. they are not a typical user and should not use themself as a reference point.

- Marketing departments often use personas and sharing personas helps communication. More importantly both product and marketing message are stronger if the target market is clearly visible during development.

- User interface (UI and UXD) designers also use personas in their work. Again communication and effectiveness are enhanced when developers a designers share personas.

**Create and manage stories:** when objectives and users are well understood it is time to start specifying what they system will do. Whatever the format used to describe the specifications something needs to be created. Once more than a few requirements have been captured there becomes a need to manage what has been created. This is the step into which much of the existing Agile literature fits: writing User Stories, Managing the Product Backlog and so on. If the 10-step model is being used as a process these process occur in tandem.
Figure 4: From stakeholders to personas
Grooming?

Some people like to talk of “Grooming the backlog.” By this they mean the regular review of the backlog, the removal of unnecessary or redundant stories, the fine tuning of existing stories, perhaps the addition of additional stories, and whatever other tasks are needed to keep the backlog short and orderly.

Unfortunately the term “grooming” in twenty-first century English English is often carries negative connotations. For this reason I prefer to avoid the term an find “managing the backlog” to be a perfectly acceptable term.

Acceptance criteria: once the essence of a story is captured some description of what constitutes done for the story needs be given. How will developers know to stop writing code? Testers know when to pass, or fail, functionality? And requirements specialists know something has actually been done? The answer to all these questions is a set of criteria that determines when a story is complete.

Acceptance criteria are not in and of themselves tests, let alone a test script. However they are the starting point for creating tests. Tools - such as FIT and Fitnesse and Cucumber for Behaviour Driven Development (BDD) - exist that can help take acceptance criteria and automatically create tests. Often acceptance criteria are the starting point for professional testers who create automatic or manual test scripts.

Development: once a need is identified, understood and acceptance criteria specified it is time to actually do the work, develop the software. Little needs to be said about this particular step because much has already been written about how development happens in Agile teams.

Product Delivery: once a need is met the product needs to be delivered to the customer. For some systems this is a trivial step, for others it is complicated and involved. Delivering a system in multiple discrete steps is very different from delivering a big bang all or nothing. Delivering a system as shrink-wrapped installable software on a CD is different to a software-as-a-service model. Inside a corporation delivering software updates to data centres may involve working with operations teams who may be employed by a different company altogether.
Value Delivery: last but by no means least is the need to close the loop and check that value is actually delivered. The key here is linking the finished product back to stakeholders’ needs and objectives. Few organizations can place a dollar amount on a single piece of functionality, for some it may be impossible; but since all requirements start with some stakeholder it should be possible to link the return to the stakeholder and check whether the thing that is delivered creates value.

At the simplest level value delivery is about ensuring that the product(s) being delivered address the needs of the objective setters and stakeholders - and thus delivers value. Sometimes this may involve going further than just asking: “Is this valuable?” Sometimes this step may involve demonstrating to stakeholders how the products delivered address the needs and move the team towards the ultimate goal.

Complicating matters there is frequently a temporal gap between the delivery of IT and realisation of value. In the extreme this gap can be measured in years, and in the case of aggregate GDP figures decades (Brynjolfsson 1993), (Brynjolfsson and Saunders 2009).

A second complicating factor is that for some organizations business value is not measured in financial terms at all. Consider for example a public sector body, a charity (Shriver 2009) or a quasi-commercial organization such as the BBC. The concept of “value to the business” exists but it may not be measured in numbers.

Returning to the original stakeholders, with the delivered product, and asking “Does this deliver value?” is one way to overcome these complications. The stakeholder should at least be able to describe how they obtain value from the product - or do not. This might be a temporary or partial solution, or it might be as good as it gets.

What no test step?

This model deliberately omits an explicit test step. Provided acceptance tests are created in an machine executable fashion (e.g. Gerkin style “given when then” or FIT style tables) and that the tool chain executes these tests then the vast majority of actual testing becomes a tiny part of the process.
There may still be a need for non-automated tests, e.g. exploratory testing, user acceptance tests or occasional large scale performance tests but what tests are needed, how often they are performed and how they change the process should be considered on a case by case basis.

From stakeholders to value delivery

At first sight it may seem odd for value delivery to appear at the end but this step is about closing the loop, ensuring value was delivered not just promised. Feedback gained at this step will be used in create stories, prioritising work and refining overall objectives.

There is symmetry between the stakeholder and value delivery steps. Stakeholders are ultimately the root of all requirements, no matter how technical. At the end of the day someone, somewhere, must want something from the system. For this person or group, the stakeholder, there is value (perhaps not financial) to having this thing done.

Value can only be assessed if the stakeholders are known. If nobody wants anything doing to a system then nothing should be done. If value cannot be assigned to work then there is no reason to incur the cost.

The stakeholder might not know what work they want doing, and they are often oblivious to the technical aspects, but then, there is no reason why they should. The route between stakeholder and technical change may be complex and non-obvious but it must exist.

Stakeholder analysis and value delivery are perhaps the two most important steps and the two certainly deserve more attention in future. Unfortunately these are largely absent from most Agile discussions.

Iterative

It cannot be emphasised enough that this model is iterative. Teams are intended to go around the model multiple times. Indeed, some steps in the model will be executed far more frequently than others. For example, additional development steps will be made every two weeks or so, but the
team may only deliver products every quarter, and only review the original objective even less frequently.

Perhaps the most important feedback loop in the model is the one described above in the Value Delivery step: that of demonstrating to stakeholders and ascertaining value. But feedback and iteration need to occur throughout the model.

The model is not intended for a single-pass - if it is used in this form it is little more than a vertical waterfall. Steps execute and re-execute, steps execute in parallel. Obviously requirements engineers can be refining the stakeholder map while developers are developing code and testers elaborating on acceptance criteria.

Requirements engineers and the development team need to regularly revisit each step of the model and update the outputs to incorporate learning from later steps which have already occurred; e.g. after delivering software to users (step 9) and hearing their comments user stories (step 6) should be updated, changed, removed, created and re-prioritised.

More tools and techniques

There is certainly no shortage of tools and techniques available to the contemporary Business Analyst or Product Manager for analysing needs. Whether it is stakeholder analysts, win-loss reports, Business Analysts modelling, UML diagrams or CATWOE (Cadle, Paul, and Turner 2010) the tools are available. This model does not try to show where each and every tool may be used: not only would it take too long but there are sometimes no clear answers.

What the model does do is firstly, place outputs and expectations at the start of the process: objective and stakeholders should provide a way in here. Secondly it shows where these tools can be used: the stakeholders and roles steps are about understanding customers and needs and it is in these stages that most analysis tools come into play.

While some tools will work within single steps in this model other tools will span multiple steps. The truth is, requirements discovery is not a neat and tidy exercise that occurs in clear-cut chunks. Like code development it involves intuition, insight and inspiration which cannot be scheduled.
As a result those charged with discovering, understanding and communicating activities are likely to have several different activity streams occurring at once, overlapping and informing one another.

For example, in tandem with this model forward-looking plans and scenarios need to be maintained. Release plans and product roadmaps (See *Three Plans for Agile*, (Kelly 2010b)) are both informed by the information gathered in the model and feed into the model.

**Three backlogs?**

Although this discussion sticks with the Scrum terminology of Product and Sprint backlog it might be better to differentiate between three backlogs:

- Opportunity backlog: all the ideas which have been suggested ever and have been considered worthwhile for recording. Recording such ideas does not in any way commit anybody to actually undertaking them.

- Validated backlog: items from the opportunity backlog which have been examined, researched and discussed enough to be considered valid candidates for future development.

- Iteration/Sprint backlog: the work that will be attempted in the current iteration.

While the iteration/sprint backlog plays the same role as it ever did - setting the agenda for the next iteration - splitting the product backlog allows for a clear separate of “good ideas” and “validated ideas.” Moving from the former to the latter involves checking ideas with stakeholders, measuring them against the over-arching goal, considering the benefits in the market or to the organizations and perhaps conducting experiments to measure benefits.

This three backlog model naturally maps to the three planning horizons discussed elsewhere, and to the commonly used Epic, Story, Task work breakdown used by teams:

- Opportunity backlog contains big items with little breakdown, Epics. These may be happen sometime in the longer term, over future quarters and years. They may appear on a roadmap or they may be more speculative.
Figure 5: Three backlogs
• Validated backlog items should be at the story size - small enough to be deliverable soon but demonstrating real business value. These items may be developed sometime in the next quarter so they appear on the quarterly plan.

• Iteration backlog items are here and now, they are task sized and are in the current iteration.

There is no point in doing more work on any item until it moves to the next backlog, into the next planning horizon. At each stage some items will disappear, upon closer examination they will not be judged worthwhile.

Epics need not be broken down in their entirety before any work is undertaken. Ideally the first stories broken out of any epic would be experiments which could test technology options and, more importantly, market and client reaction.

For example the first stories for an epic entitled “Launch French version” might describe a series of data gathering experiments to assess the size of the market and opportunities. Translation, payment and such can wait, they might never need doing.

Unplanned

Of course the whole backlog discussion assume work can be known and planned in advance. Unplanned work, or short notice, work always has a habit of appearing. Items may appear from nowhere and short-circuit the processes and be developed really soon.

For some teams unplanned work is a problem, indeed the regular appearance of unplanned work might itself indicate a problem with the requirements activity. Teams aiming for predictability may well aim to stamp out unplanned work.

But unplanned work can still be valuable, indeed unplanned work might actually be the most valuable work a team does. Unplanned work deserves extra celebration when delivered because it appears and meets a business need at short notice. Unplanned work should still be reviewed for value.

Over time teams need to consider whether to optimise themselves for planned work or unplanned work. Whether to stamp out and limit unplanned work or whether to embrace it.
Conclusion

This is a deliberately brief explanation of the 10 Step Model. I find it helps explain and pull together some of the ideas floating around the discussion on Agile Requirements and provides a reference point when discussions go beyond iterations.

Although it is simplest to explain as a process I shy away from calling it that. Rather I prefer to think of it as a check-list and a guide. The key point is that all these step should be happening, and happening in some form again and again. Certainly steps omitted but I’d like to hear an explanation of why any particular step was removed.

Perhaps it is better still to view this model as a starting point for your own model. Which steps would you remove? Which would you add? Would you reorder any?

History & Acknowledgements

Despite attempts to replace the term “Step” in the title I have yet to find an alternative name. “Part”, “Element” and so on imply no sequence but “Step” implies too much sequence.

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2009</td>
<td>Never shown publicly</td>
</tr>
<tr>
<td>2</td>
<td>February 2010</td>
<td>Published ACCU Overload &amp; shown at conferences and in training</td>
</tr>
<tr>
<td>3</td>
<td>2011</td>
<td>Minor revisions</td>
</tr>
<tr>
<td>4</td>
<td>June 2013</td>
<td>Two cycles introduced and 3 backlogs suggested</td>
</tr>
</tbody>
</table>

This is version 4.0 of the 10 Step Model and the most significant change for a while.

The author would like to thank Ric Parkin and the Overload editorial team for comments and suggestions prior to the first publication. Additional thanks for
due to everyone who has commented on the model following training classes and conference presentations.

Particular thanks are owed to Lindsey Brodie for reviewing the model and making suggestions which form the bulk of changes for version 3. Implicitly thanks are due to Tom Gilb for several of the idea embedded in this model - see the excellent Competitive Engineering (Gilb 2005). While Tom might not agree with every aspect of this model his ideas underlie much of the thinking.

(c) Allan Kelly - allan@allankelly.net - www.allankelly.net

References


