FEATURE SLICING

RIGHT-SIZING FEATURES FOR SAFE® PROGRAM INCREMENTS

Written by: Ian Spence & Keith de Mendonca

IVAR JACOBSON INTERNATIONAL

One of the key activities that will help make your SAFe® program a success is the careful preparation of your Features prior to Program Increment (PI) planning. And one important part of this preparation is to slice up any of the targeted Features that are too large to be easily delivered within the PI.

In this guide, we would like to share some of our experiences in slicing Features. And, in tribute to Richard Lawrence and his popular Story Splitting poster, provide you with a complementary Feature Slicing poster for you to use in your SAFe program.

FEATURES - THEY'RE JUST BIG STORIES AREN'T THEY?

It is very tempting for Product Owners and Product Managers to treat Features as though they are giant User Stories, and to apply the same splitting approach to the Features in their program backlog as they already use for the User Stories in their team backlogs.

Just as we have found that it can cause problems to write Features using the Story format (see the Features and Capabilities article provided as part of the interactive Scaled Agile Framework web site www.scaledagileframework.com for further details) we have also found it problematic to apply the same splitting rules to Features as we do to User Stories.

In the Scaled Agile Framework a clear distinction is drawn between the purpose, structure and content of Features, and that of Stories (including User Stories and Enabler Stories):

- Features are the visible 'units' of business benefit that the customer recognises and it is at this level of Feature granularity that the customer will prioritise their needs.
- Features can span multiple user roles, user stories and use cases.
- Multiple teams may work simultaneously on the same Feature. Teams can swarm together to deliver Features.
- Features may take many Sprints to complete. The Features are implemented Story-by-Story.
- Features should be easily completed within a Program Increment; remember Features are to the longer Program Increments as Stories are to the Sprints.
- Stories are the units of work that each team breaks their Features into to help them incrementally develop and deliver the Features. They exist to help the team (and their stakeholders) examine, discuss, agree and sequence the work they believe is needed to deliver a Feature.
- Stories should be completed within a single two week Sprint.
- Stories can exist without Features, allowing the teams to make changes without the need for additional Features.

There are a number of sources of rules for Story splitting, our favourites being those published by Richard Lawrence (including his popular Story Splitting poster) and Dean Leffingwell (see the Story article provided as part of the interactive Scaled Agile Framework web site scaledagileframework.com). These rules still apply when using Features and really help teams in the identification, separation and sequencing of the stories they identify to implement the Feature but they are not as helpful when it comes to slicing up Features.

For example, we would never recommend deferring the performance requirements related to a Feature into a later Feature - although we may only implement these 'performance Stories' towards the end of development of that Feature. The same logic applies to handling error flows, CRUD and data variations; we may delay Stories which implement these important software attributes for a Feature towards the end of its development, but we would not defer this work to a separate Feature.

When slicing Features it is always important to remember that these are the true releasable elements of the system and so must always provide a robust, usable solution to the users. Features will be built Story by Story, but in themselves must provide a 'complete' usable solution. Note that a complete usable solution means 'no bits missing', rather than that all possible Stories have been implemented.

There is another important difference between splitting Stories and Slicing Features: when we split a Story we usually split the original Story into a set of similarly sized new Stories that completely replace the original Story; when slicing Features, we usually just find the most important slice and leave the rest to be addressed later.

SLICING FEATURES

So let us imagine that a team has just sized a Feature (typically in Story Points) and has discovered that it is too large to be easily delivered within the next Program Increment. To slice this Feature they will need to select a slicing method that lets them de-prioritize or defer some of the functionality. Once the Feature has been sliced, they will select those pieces of business value that will deliver customer benefit quickly.

What Slicing strategies can you use? We have identified the following 10 useful patterns for slicing:

1. KISS – Keep it simple stupid: aim for the simplest implementation that could be released without compromising the systems performance etc. This is often the happy path with some basic error handling. Defer the 'bells and whistles' to later features, once the basics are in place and working at a sufficiently high level of quality.

2. DEFER OPTIONAL BEHAVIOR – Does the Feature include lots of optional behaviours? Make the optional behaviours separate Features to be done once the core functionality is delivered.

3. SEPARATE BUSINESS VARIATIONS – Could the Feature be released incrementally to different areas of the business. Start with the simplest business variant first to generate fast feedback. For example could we do a simple solution for retail banking before fleshing out the solution and offering it to investment banking, trading and other banking areas? Also consider geographical variations – in many businesses the business rules vary by geography with different rules for the US, Europe and Asia. Perhaps we can launch the Feature in one geography before adding support for the others via additional Features.

4. SEPARATE DIFFERENT CHANNELS – Could the Feature be released incrementally to support different channels / routes to market / the same functionality over different mediums? For example different operating systems or the different channels used to provide retail banking to customers including mobile banking, the banking app, and in-branch banking. It is logically the same feature in all cases but the delivery of the Feature on each channel could be considered a different Feature (and in some cases may not be needed at all).

5. ADDRESS DIFFERENT USER GROUPS INDIVIDUALLY – Do different users want different sets of stories? Understanding the different user groups' needs can help you to split the Feature and better understand the specific benefits to each user group. For example our new Feature may appeal to different demographics but each demographic will want to apply the Feature in a different way. In this case slicing the Feature will allow us to focus on just the sub-set of the stories each demographic needs and capture the interest of the more important groups earlier.

6. CONSIDER INCREMENTALLY SOURCING DATA - Is all the data needed before any benefit can be provided? Perhaps the Data can be consumed incrementally or sourced from existing secondary sources?

7. ISOLATE SPECIAL VARIATIONS - First focus on the popular / high volume cases then add the more specialized corner cases as additional Features – you may find out that their value is very small and they are never needed.

8. BREAK OUT COMMON ENABLERS – Business Features often rely on the same underlying system behaviors, making the first of the features to be implemented look very large and complex. Breaking out these common enablers can de-risk, reduce the estimate for, and simplify the implementation of many business features.

9. FIND A STORY GROUP – Remember 80% of the business benefit is likely to come from 20% of the stories. Find these stories and treat them as their own Feature. See Story Mapping for more information.

10. BREAK OUT A SPIKE – Sometimes you don't know enough to even plan something. As a last resort break out a spike.

BAD SLICING

We have also encountered a number of dangerous anti-patterns including:

- Deferring non-functional requirements – a common pattern when splitting user stories that can cause problems when slicing features. When implementing the Feature we may focus on the non-functional aspects after we've got the initial stories working but we shouldn't release the Feature if it is below the acceptable quality and performance levels. We have seen many teams get into trouble by compromising on quality in their dash to get more and more Features into the product.

There are circumstances, of course, where a limited release of a Feature can be done to obtain feedback and that by explicitly limiting the number of users limit the number of applicable non-functional requirements that apply. Great care must be taken to ensure that people don't think that it can then be opened up to all and sundry without additional work, and the addition of further Features.

- **SLICING TOO EARLY** – Only slice a Feature if it is 1) needed in the near future and 2) too big (at the current time) to effectively flow through the system. Remember that the estimates to complete Features change over time - what appears to be too big today may have a much smaller estimate in the future.

- **OVER-SLICING** – There is no need to slice a Feature up into lots of smaller features all in one go. Generally, it is enough to find the first one or two slices to be implemented and leave the rest of the initial Feature to be addressed later, after the initial slices have been implemented.

- **SLICING BY COMPONENT** – We've found that technologists often find it hard to resist slicing things up by architectural component, sub-system or layer. This is a tactical solution that is often applied at the Story level, particularly when dealing with component teams or when team are struggling to fit their stories into two week sprints. This is a bad habit that should be resisted even more strongly when dealing with larger items such as Features.

- **FORGETTING THE FEATURE TESTING** – Features often require additional testing beyond that needed to complete their Stories. One slicing anti-pattern is to defer this testing to a later slice of the Feature rather than including appropriate amounts of testing in each of the new smaller Features being created.

- **SLICING BY OPERATION OR WORK-FLOW STEP** – it is often tempting to split a Feature up into a series of parts around the workflow steps or operations it involves. For example looking at the input, processing, and final output of the process as separate Features or, at a simpler level the creation, access, update, and deletion of items. This is a common anti-pattern when it comes to splitting Features, as there is little or no business value in just being able to complete the first step of something. Focus on identifying the core behavior involved in the Feature and creating the simplest end-to-end solution. Additional Features can then add variations, bells or whistles once the basic end-to-end functionality has been released and is in the hands of the users. This often leads to the situation where 90% of the Features are complete but the users still can't achieve any of their actual goals. They can enter things, find things, manipulate things but they can't actually complete the process they are trying to perform.

HOW TO SLICE A FEATURE

1 PREPARE THE INPUT FEATURE

WARNING – Don't slice Features unless something is needed in the next PI

Does the Feature satisfy INVEST*
(Except, perhaps sized appropriately)

YES

NO

Reformulate the Feature to clearly communicate the benefit or slice off one or more smaller Features which do satisfy INVEST and carry clear benefit.

Is the Feature size less than 1/10th of your program velocity?**
(or typically medium or smaller).

You're done

Continue. You need to split it.

* INVEST Features should be:
Independent
Negotiable
Valuable
Estimable
Sized Appropriately
Testable

** Velocity varies between programs but as a rule of thumb a program should be tackling at least the 'Top 10' Features hence the no greater than 1/10th of the program velocity guideline.

2 APPLY THE SLICING PATTERNS

Does the Feature have a simple core that provides most of the benefit and / or learning? This is often the happy path with some basic error handling.

KISS (KEEP IT SIMPLE)

Could you slice the Feature to do that simple core first and build on it later with further Features?

Does the Feature include lots of optional behavior (for example different ways to achieve the same goal)?

DEFER OPTIONAL BEHAVIORS

Could you make the optional behaviors separate Features to be done once the core functionality / most popular option is in place?

Does the Feature lend itself to being released incrementally to different areas of the business?

SEPARATE BUSINESS VARIATIONS

Could you deliver it one business at a time? Could you start with the simplest business variant to generate quick wins and fast feedback?

Does the Feature need to be delivered over different channels, different mediums or different routes to the customer?

SEPARATE DIFFERENT CHANNELS

Could you deliver it one technology / one channel at a time? Could you start with the channel or most value to the business and add the other channels over time?

Does the Feature involve different user groups with different goals?

Does the Feature involve different user groups that want different sets of stories?

ADDRESS DIFFERENT USER GROUPS INDIVIDUALLY

Could you give each User Group their own Feature? This can help you to better understand the benefits to each group. See also Break Out Common Enablers.

Does the Feature involve lots of data from many sources?

Could you deliver benefit with a sub-set of the data? Could the data be consumed incrementally or sourced from existing secondary source.

CONSIDER INCREMENTALLY SOURCING DATA

LAST RESORT

BREAK OUT A SPIKE

Are you still baffled about how to slice the Feature?

Can you define the 1..3 questions most holding you back?

Write a Spike / Knowledge Enabler with those questions in mind.

Do the minimum to answer the questions and then start again at the top of this process.

Does the Feature include Special Variations?

ISOLATE SPECIAL VARIATIONS

Could you focus on the most popular / highest volume cases first and treat the more specialized corner cases as separate Features? You may find that their value / cost ratio is very small and they are never needed.

Do many Features rely on the same underlying system behaviors (often making the first of them selected to be very large and complex)?

BREAK OUT COMMON ENABLERS

Could you break out the common enablers into their own 'Architectural' Features? Delivering the enablers can significantly de-risk, simplify and reduce the estimates for the other, related Features.

Does 80% of the value come from 20% of the Stories?

FIND A STORY GROUP

Could you find the set of most valuable Stories and develop and release them as their own Feature?

3 EVALUATE THE SLICING

Are the new Features roughly equal in size?

YES

NO

Try another pattern on the original Feature or the new Features that are too large.

Do each of the new Features readily fit into a PI (< 1/10 of the program velocity)?

Do each of the new Features satisfy INVEST?

Try another pattern to see if you can get this.

Is there an obvious Feature to start with that gets you early benefit, learning, risk reduction etc?

You're done, though you could try another pattern to see if it gets better results.

WARNING DON'T:
- **Defer non-functional requirements**
- **Slice too early**
- **Over slice**
- **Slice by component**
- **Forget the Feature testing**

IVAR JACOBSON INTERNATIONAL

ivarjacobson.com

Inspired by, and complementary to, "How To Split A Story", Richard Lawrence, www.agileforall.com

ABOUT IVAR JACOBSON INTERNATIONAL

IJI is a global services company providing high quality consulting, coaching and training solutions for customers seeking the benefits of enterprise - scale agile software development.

We are passionate about improving the performance of software development teams, and maximizing the delivery of business value through technology.

Whether you are looking to transform a single project or program or your entire organization with lean and agile practices, we have solutions to suit your needs.

www.ivarjacobson.com

EUROPE

SWEDEN
Ivar Jacobson International AB Runbovägen 1B
192 48 Sollentuna
+46 8 515 10 174

UNITED KINGDOM
Central Point
45 Beech Street
London
EC2Y 8AD
+44 (0)207 953 9784

NORTH AMERICAN OFFICES

UNITED STATES
211 N. Union Street
Suite 100 PMB 10055
Alexandria, VA
22314
+1-703-434-3344

ASIA PACIFIC OFFICES

CHINA
Room 706,
113 Zhichun Road,
Haidian Dist,
Beijing, China
+86 10 6268 0480

SINGAPORE
10 Anson Road
International Plaza #31-10
Singapore 079903
+65 9772 3538

IVAR JACOBSON
INTERNATIONAL