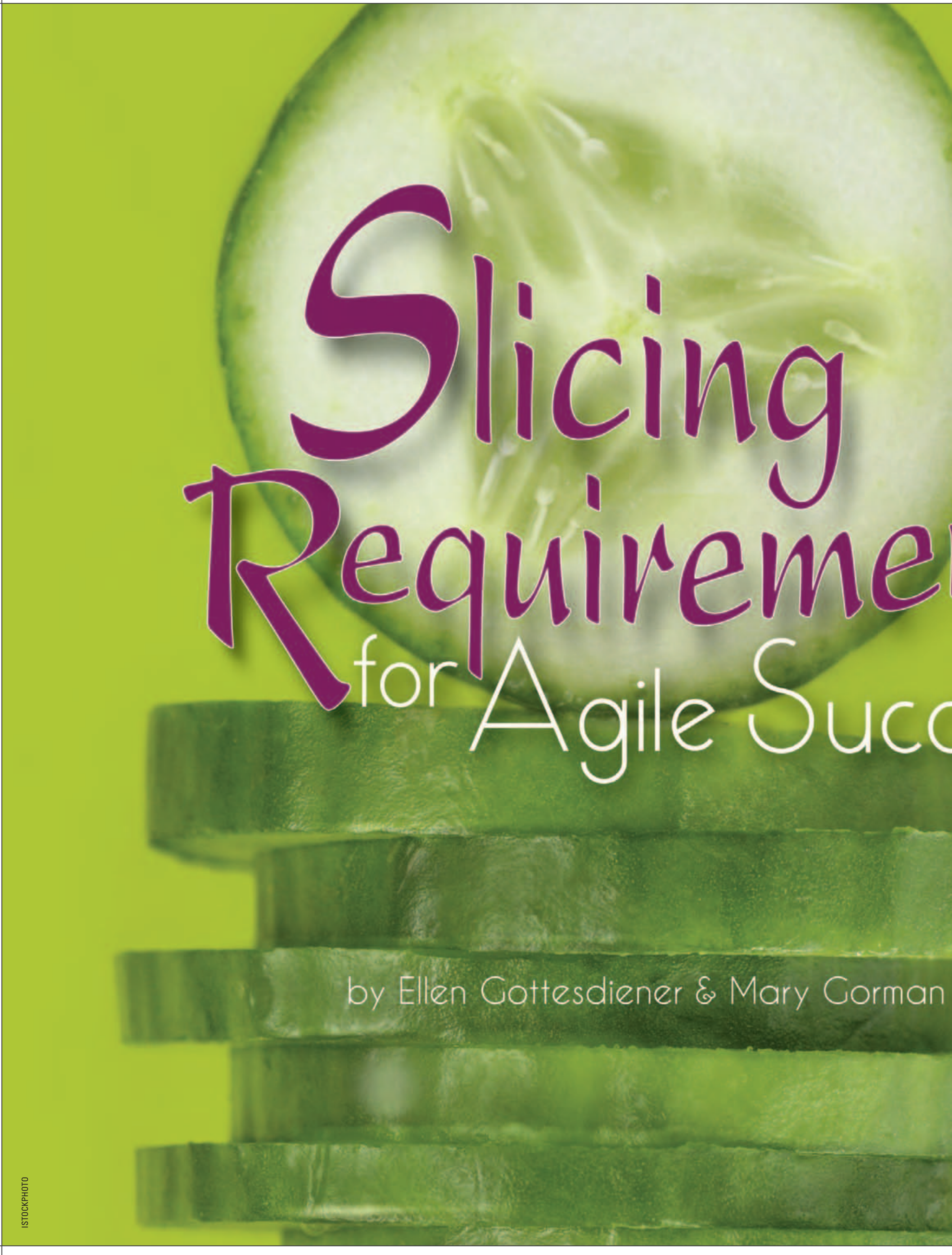
Slicing Requiremen for Agile Succ

by Ellen Gottesdiener & Mary Gorman



ISTOCKPHOTO

If you've struggled to find collaborative ways for your team and customer to understand evolving product needs while delivering value on a steady basis, you're not alone. Challenges include accurately estimating the work, readily defining conditions of satisfaction ("doneness" criteria) for requirements, churning on requirements as you begin development, being unable to deliver as promised because new requirements pop up in the middle of development, inability to easily pull requirements into current development work when there is excess capacity, or being unable to meet your delivery commitments.

A typical culprit is "chunky" requirements. They're too big, they aren't clearly prioritized, or the team doesn't understand them in sufficient detail to implement them efficiently.

We've used a variety of "splitting" techniques to ratchet up agile teams' awareness of the importance of working with requirements in the form of user stories. [1] Popular approaches include splitting by business process, database operations (e.g., create, read, update, delete), data groups, low fidelity versus high fidelity, and so on. [2, 3, 4]

Some teams find it hard to apply story-splitting strategies consistently. Others focus on the technical aspects of these strategies rather than base their work on a firm understanding of user needs. And still others neglect nonfunctional requirements, focusing only on functional requirements.

If this sounds like your agile team, stay tuned. We want to share a better way.

Slicing User Story Options Based on Value

Our requirements-slicing technique starts from the assumption that the team and the customer have a shared understanding of the product vision and goals, and they agree on what constitutes value and how value will be measured. Our focus here is on a practical way to pull the requirements out of those high-level needs so that the team and customer can deliver working software in smaller chunks over time.

To do that, teams and business customers need to jointly explore requirements options, identify the most important ones, and slice chunky stories into manageable pieces. The optimal slicing technique would be reusable in all problem domains, leverage the discipline of requirements analysis, be quick to learn and efficient to use, engage product owners (a.k.a. customers), put the "user" back into user stories, directly feed into acceptance tests, and deepen the entire team's knowledge of the requirements.

Based on our experience with the power of small, testable user stories and inspired by the work of Chris Matts and Olav Maasen on real options and feature injection [5, 6], Bill Wake and others on story splitting [1, 2, 3, 4], Jeff Sutherland and others on ready requirements [7, 8, 9, 10], Dean Leffingwell on a lean backlog [11], and Mike Cohn on minimizing team handoffs [12], we've defined a straightforward way to slice user stories.

What Is a Ready User Story?

Ready stories are small, well-understood, valuable user stories. Other names for user stories in this state include right-sized, sliced, iteration-level, queued for development, and analyzed stories.

A ready user story is highly valued by the product owner, needed for delivery in the next release or iteration, and understood well enough by the delivery team to estimate, plan, commit to, and deliver.

When Do You Make User Stories Ready?

You want to get user stories ready just before planning as part of *work-ahead analysis* or *backlog grooming*. You groom your backlog continuously so that items are ready before estimating and planning—whether you use timeboxed iterations or a continuous-flow mechanism such as kanban. The team needs to be ready, with enough requirements knowledge to specify acceptance criteria, reliably estimate the work, and meet delivery commitments. By conducting regularly scheduled, informal agile requirements workshops, the team collaborates with the product owner to groom the backlog for changing requirements and prepare user stories for upcoming delivery cycles.

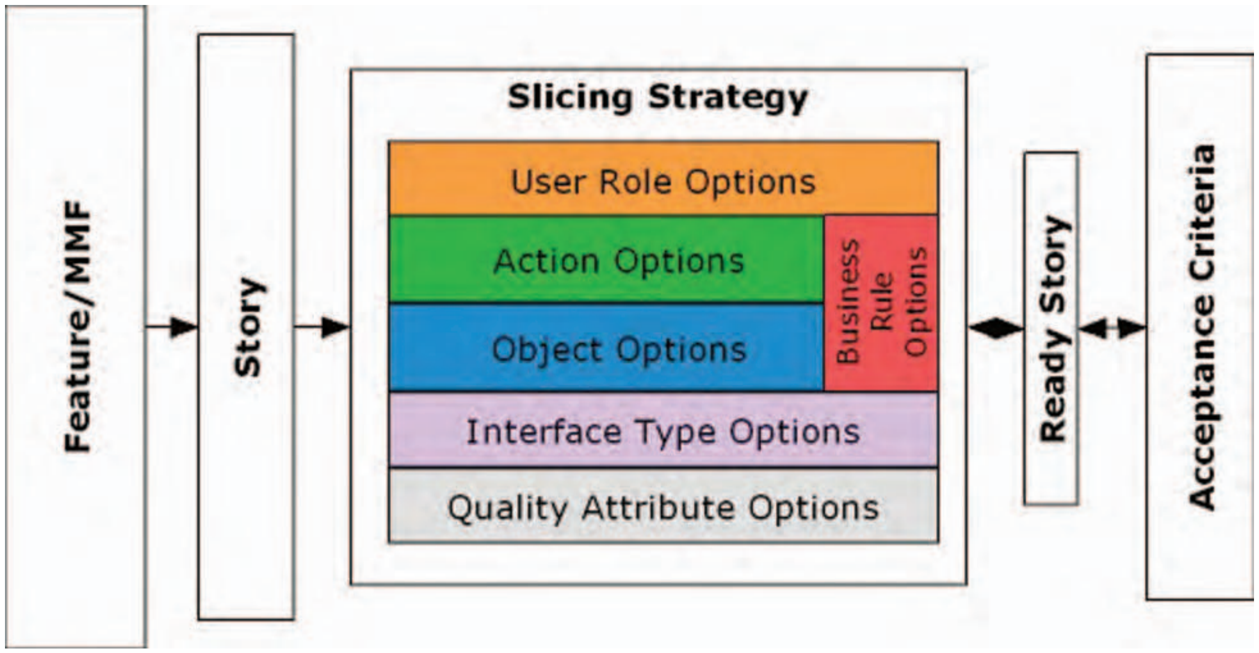


Figure 1: The six slicing elements of a user story

Analyzing User Story Options: Expand-Then-Contract

To get your user stories ready, our story-slicing technique follows an overall pattern of *expand-then-contract*. For each user story, in the expand phase, the product owner first elaborates possible options. Then, in the contract phase, the team and the product owner explore the options and quickly narrow them based on their value; this takes a matter of minutes. The result is a small, concise, thinly sliced story ready for development.

Let's explore this pattern in more detail.

In the expand phase, you begin with a single chunky story and explore the analysis options. Your analysis looks at six elements of the user story:

- Who acts in the *user role*? What are the types and states of that role?
- What *actions* are needed to meet the user's goal?
- What *data objects* are acted on? What are the types and states of those data objects?
- What *business rules* must be enforced for the actions and data objects?
- What *interfaces* are needed? What are their types—manual solutions, physical hardware devices, system-to-system interfaces, or user interfaces?
- What *quality attributes* constrain and control the sliced user story?

In the contract phase, the product owner collaborates with the team to select options using transparent, rational criteria—for example, return on investment, market drivers or events, or risk reduction. She uses prioritization techniques such as a tailored ranking scheme, stakeholder value, voice of the customer, and so on.

Throughout this expand-then-contract process, the team and product owner together gain crucial knowledge. As the product owner assigns a value to each option, she asks the team to gauge the effort and risks for implementing it, heightening her understanding of technical concerns and the development process. As team members discuss the options, they learn about the requirements. Team members question, challenge, and clarify the product owner's filtering criteria, deepening their understanding of the business domain and the requirements that will provide value.

Note that you will find this technique useful for product requirements but not team or technical work, which some teams represent as "technical" stories. And remember: As you proceed through each expand phase, if you discover that any of the six user story elements has no options, simply move to the next element.

Slicing Stories by Analyzing Options: An Example

Let's iterate through the technique again, this time looking at the six elements via a sample story.

You begin by selecting a high-value user story from your backlog. The story might originate from, say, a high-level feature or a minimal marketable feature (MMF). [13] Or, it might have been decomposed from chunkier requirements—what some teams call "epics."

On agile teams, these stories are usually written in the following form:

As a [*user role*]
I need to [*some behavior or action*]
so that [*business value*]

"Your goal is to attain a set of high-value, ready stories that are about equal in size and are sufficiently analyzed so that you can size, plan, and deliver them as promised."

Next, determine if the story is a candidate for slicing.

What makes a user story "too big"? That differs for each team. Your team's size and iteration or release length are factors. Some teams strive to size their stories so that they can complete a single story in a few hours or a day. Other teams have a longer time frame—two to three days at most.

Your goal is to attain a set of high-value, ready stories that are about equal in size and are sufficiently analyzed so that you can size, plan, and deliver them as promised.

Books and Beyond

Figure 1 shows the six elements of a user story.

In our sample, we're building an application for a business that sells products such as books, movies, music, and greeting cards. You select the big, chunky story: "As a *customer*, I want to *buy a product* so that I *can enjoy using it*." Follow along as we explore options and assign a value to each one.

Step 1: User Role Options: Types and States

To analyze the user role, we explore options for the *types* and *states* of the customer. The user role initiates the story and has the goal of obtaining value. The name of the role is based on the user's intention or goal. In the user story, the customer intends to buy; hence, we name the user role "buyer."

Working with the product owner, we define the types of buyers:

User Role Type Options
Individual Buyer
Corporate Buyer
Club Member Buyer
Employee Buyer

These types, along with all the domain terms, should be defined in the product glossary to enable a shared understanding of the domain (also providing a stable basis for domain-specific modeling).

Select for Value

Now ask the product owner to prioritize the user-role type options. Which one has the highest business value for the next iteration? Our product owner wants to focus on income and not slow down delivery with complex buyer types. She selects Individual Buyer.

This is our first use of the expand-then-contract pattern—but definitely not our last.

For the chosen user role type, next explore its possible states or the lifecycle stages it undergoes.

User Role State Options (for Individual Buyer)
New
Existing
Anonymous (can buy without providing a name)
Archived

Select for Value

Now that we've expanded the states, it's time to contract. Again, our product owner needs to narrow her options to the buyer state that will yield the highest immediate value. She selects Individual, Anonymous Buyer.

Step 2: Action Options

To identify all the possible actions that satisfy the selected user role's desired goal, start with the verb in the user story text: "I want to *buy* a product. . ."

Ask the product owner what typically happens (or, if the action is new, what will potentially happen) and what decisions must be made. For the individual, anonymous buyer, the buy action options are as follows:

Buy Action Options
Verify product cost
Calculate tax amount
Calculate total purchase amount
Apply discount
Apply wrapping fee
Arrange for shipping
Secure payment
Adjust inventory
Generate receipt
Post payment to accounts receivable

We suggest writing each action option on a sticky note so that the product owner can move them around. The order of the options is not critical at this point.

Select for Value

The product owner surveys the action options and determines the minimum options needed for the next delivery cycle. Encourage her to defer actions that are less valuable, occur infrequently, require data not yet built into databases, or are complex and better left until later.

Be sure to analyze any interdependent options—those that might need to be selected together. At the same time, look for independent actions. For example, ask your product owner, "Is it necessary to offer discounts in the next iteration or release, or can that wait?"

To guide the selection, consider the context of the action. For example, is the buyer shopping at a brick-and-mortar store or online? We learn that the immediate need is to support in-store purchases. Our product owner selects four options (checked):

Buy Action Options

- ✓ **Verify product cost**
- **Calculate tax amount**
- ✓ **Calculate total purchase amount**
- **Apply discount**
- **Determine wrapping fee**
- **Arrange for shipping**
- ✓ **Secure payment**
- **Adjust inventory**
- ✓ **Generate receipt**
- **Post payment to accounts receivable**

Step 3: Data Object Options: Types and States

Next, we focus on data objects. Your product owner should identify object options based on her prior selections of user role and actions. In our example, the objects are Product, Payment, and Receipt.

For each object option, identify types or variations. For example, the Product object's types include Book, CD, DVD, and so on.

Select for Value

The team has analyzed the objects and expanded its understanding of the objects' types. The product owner assesses the business value of the options and determines valid combinations. The product owner chooses the highest-value object types (checked items in table 1):

Product Type Options	*Payment Type Options*	*Receipt Type Options*
✓ Book	✓ Cash	✓ Cash receipt
CD	Credit card	Credit card receipt
DVD	PayPal	
Gift card	Purchase order	
Greeting card		
Electronic book reader		

Table 1

There's more. As you did with the user role, you need to explore possible lifecycle states for the objects specific to our story. In our example, the Book object has two states: new or used.

Select for Value

As before, the product owner now contracts the options. Her selection is New Book.

Book State Options

- ✓ **New**
- **Used**

What We've Sliced So Far

We started with the user story: "As a *customer*, I want to *buy a product* ..." We used the expand-then-contract pattern to slice three elements: user role (types and states), actions, and data objects (types and states). The story has been sliced:

"As an *individual anonymous buyer*, I want to *buy a new book ... paying with cash and receiving a cash receipt*."

The story includes four action options: verify product price, calculate total purchase amount, secure payment, and generate receipt.

At this point, you have partially sliced your story. Your team might choose to defer analyzing the three additional slicing options (business rules, interfaces, and quality attributes) until development. But, in our experience, the additional minutes spent exploring these other options before you commit to delivering a story can significantly help in creating a ready story while expanding the team's knowledge of the product requirements. Let's take a look.

Step 4: Business Rule Options

Business rules define constraints and conditions that must be satisfied. As part of developing your story's sliced, high-value actions and objects, you need to elicit business rules that must be enforced. Limiting business rule options to only the prioritized user role, actions, and objects streamlines your analysis work and ensures eliciting "fresh" rules.

Select for Value

The product owner again weighs the business value of these options for the next iteration. She chooses the following (checked):

Business Rule Options

- ✓ **Payment currency must be specific to purchase location**
- **Cash payment denomination amount must not be greater than ...**
- ✓ **Payment change amount is calculated as ...**
- **Receipt bar code is designed using ...**

Step 5: Interface Type Options

You can gain a high-level view of your story's interfaces by quickly drawing a context diagram of the user story or its higher-level, minimal marketable feature.

Focus on the interface options that are relevant to the high-priority user role (type and state), actions, and objects (type and state). Then, determine the appropriate type of interface (manual, hardware, system to system, or user).

Our sample user story requires interfaces for the book information (to verify the cost), the cash payment (to secure the payment), and the generated cash receipt.

Select for Value

Together, the product owner and team discuss the options. Based on business and technical factors, they agree on the following:

Book Interface Type Options

Scanner (hardware)
✓ **Keyed in data (UI)**

Cash Payment Interface Type Options

Cash machine (hardware)
✓ **Keyed in data (UI)**

Cash Receipt Interface Type Options

✓ **Printed in store (report)**
Faxed (system to system)
Emailed (system to system)

Step 6: Quality Attribute Options

Quality attributes are "the subset of nonfunctional requirements that describe properties of the software's operation, development, and deployment." [14] Teams sometimes neglect these crucial requirements until well into development. But, we've learned that teams need to explore options for performance, reliability, safety, security, scalability, usability, and so on if they're going to define a ready story and improve the quality of the product's ever-unfolding architecture.

There are several ways to specify quality attributes. For example, you can write them as user stories, use Planguage (a specification language) [15], list the quality attributes as user story acceptance criteria, or incorporate them into the user story text.

One quality attribute we need for our sample story is the response time for printing the receipt. Borrowing from Gilb's Planguage tags, you can specify response-time requirements as follows:

Tag: ResponseTime.CashReceiptPrintLaunch
Scale: Seconds
Meter: Elapsed time between pressing "Receipt" to the start of printing
Minimum: No more than 7 seconds
Plan: 4 seconds
Wish: 2 seconds

Alternatively, you can write your story's quality attributes on the back of the user story card (or in your backlog management tool)—for example, "Cash receipt begins printing within four seconds of pressing the Receipt key."

The Sliced Story

Here's a quick recap. We started with a big, chunky story: "As a *customer*, I want to *buy a product* ..." Using the story-slicing technique, we successively sliced it into these high-value options:

User role type and state: Individual, anonymous buyer
Actions: Verify product price, calculate total purchase amount, secure payment, generate receipt
Objects (type and state): New book, cash payment method, cash receipt
Business rules: Payment currency must be specific to purchase location, payment change amount is calculated as ...
Interfaces:
Book interface type: Keyed in data (UI)
Cash payment interface type: Keyed in data (UI)
Cash receipt interface type: Printed in store (report)
Quality attributes:
Tag: ResponseTime.CashReceiptPrintLaunch ...

Our thinly sliced, ready story meets the INVEST criteria: independent, negotiable, valuable, estimable, sized appropriately, and testable. [16]

In development, the team details each story's selected options. That includes building user acceptance tests, regardless of testing methods or tools—the Given-When-Then construct (e.g., jBehave, easyB, Cucumber), data tables (e.g., FIT or FitNesse), or other scripting tool. Meanwhile, work-ahead analysis continues on lower-priority options for upcoming delivery cycles.

And, yes, we have found this slicing technique useful for requirements in forms other than user stories, including use cases, snippets of text-user requirements statements, events, and features.

Slicing for Success

By collaborating with business customers to explore requirements options and successively slice them at the last responsible moment, the team can continually groom the backlog—and continually deliver well-understood, valuable requirements. Along the way, everyone in the project community benefits by expanding and deepening their requirements knowledge.

This "just enough, just in time" slicing method is a fast, efficient, repeatable technique that streamlines planning and jointly engages the customer and team in optimizing value—all goals of successful agile teams. **{end}**

ellen@ebgconsulting.com
mary@ebgconsulting.com

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.
■ References

Appreciations

Mary and Ellen Gottesdiener thank Susan Block, Mike Cohn, Chris Matts, Tom Poppendieck, Jeff Sutherland, and Bill Wake for their review and incisive comments on a draft of their article "Slicing Requirements for Agile Success." {end}

References

[1] Wake, Bill. "Twenty Ways to Split Stories." XP123: Exploring Extreme Programming, December 2005.

[2] Lawrence, Richard. “Patterns for Splitting User Stories.” October 2009.

[3] Rainsberger, J.B. "Splitting Stories: An Example." June 2007.

[4] Koskela, Lasse. "Ways to Split User Stories." Blurts on the Art of Software Development, blog, June 2008.

[5] Matts, Chris, and Olav Maassen. "Real Options Underlie Agile Practices." InfoQ, June 2007.

[6] Matts, Chris. Feature Injection Episodes 1 through 4. LimitedWIPSociety.org, May 2009.

[7] Sutherland, Jeff. "Practical Roadmap to Great Scrum: Systematically Achieving Hyperactivity." Agile Bazaar, Boston, November 2009.

[8] Jakobsen, Carsten, and Jeff Sutherland, "Scrum and CMMI--Going from Good to Great: are you ready-ready to be done-done?" Agile 2009, Chicago, 2009.

[9] Harvey, Jack. "The Product Owner Ready Board." Agile Product Owner Blog, November 2009.

[10] Beaumont, Serge. "Flow to READY, Iterate to DONE." Xebia, July, 2009.

[11] Leffingwell, Dean. "More on Lean, Backlog and Little's Law." Scaling Software Agility, January 2010.

[12] Cohn, Mike. *Agile Teamwork: 3 Ways to Minimize Handoffs*. *Better Software*, March/April 2010.

[13] Denne, Mark, and Jane Cleland-Huang. *Software by Numbers: Low-Risk, High-Return Development*. Prentice Hall, 2003.

[14] Gottesdiener, Ellen. *The Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements*. GOAL/QPC, 2005.

[15] Gilb, Tom. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Addison-Wesley, 2005.

[16] Wake, Bill, op. cit.

EBG Consulting, Inc. Principal Consultant and Founder **Ellen Gottesdiener** helps business and technical teams collaborate to define and deliver products customers value and need. Author of two acclaimed books, Ellen works with global clients and speaks at industry conferences. Learn more from her articles, tweets and blog, free eNewsletter and find resources on EBG's web site. Contact Ellen at ellen@ebgconsulting.com.

Mary Gorman, CBAP©, is senior associate at EBG Consulting and helps project teams explore, analyze, and build robust business and system requirements models. Mary serves on the Business Analysis Body of Knowledge committee of the International Institute of Business Analysis and is the leader of the elicitation subcommittee. She can be reached at mary@ebgconsulting.com and ebgconsulting.com.