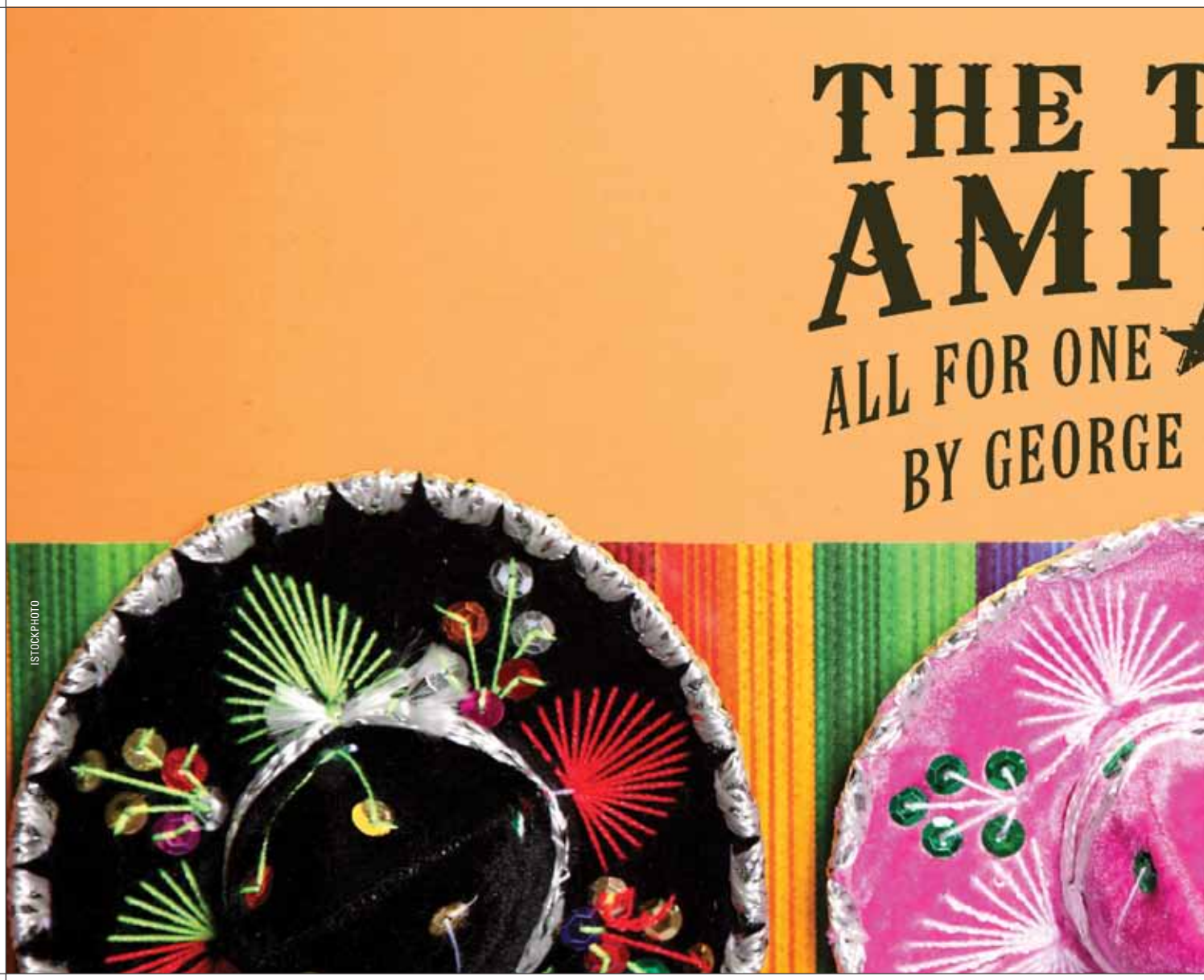


ISTOCKPHOTO

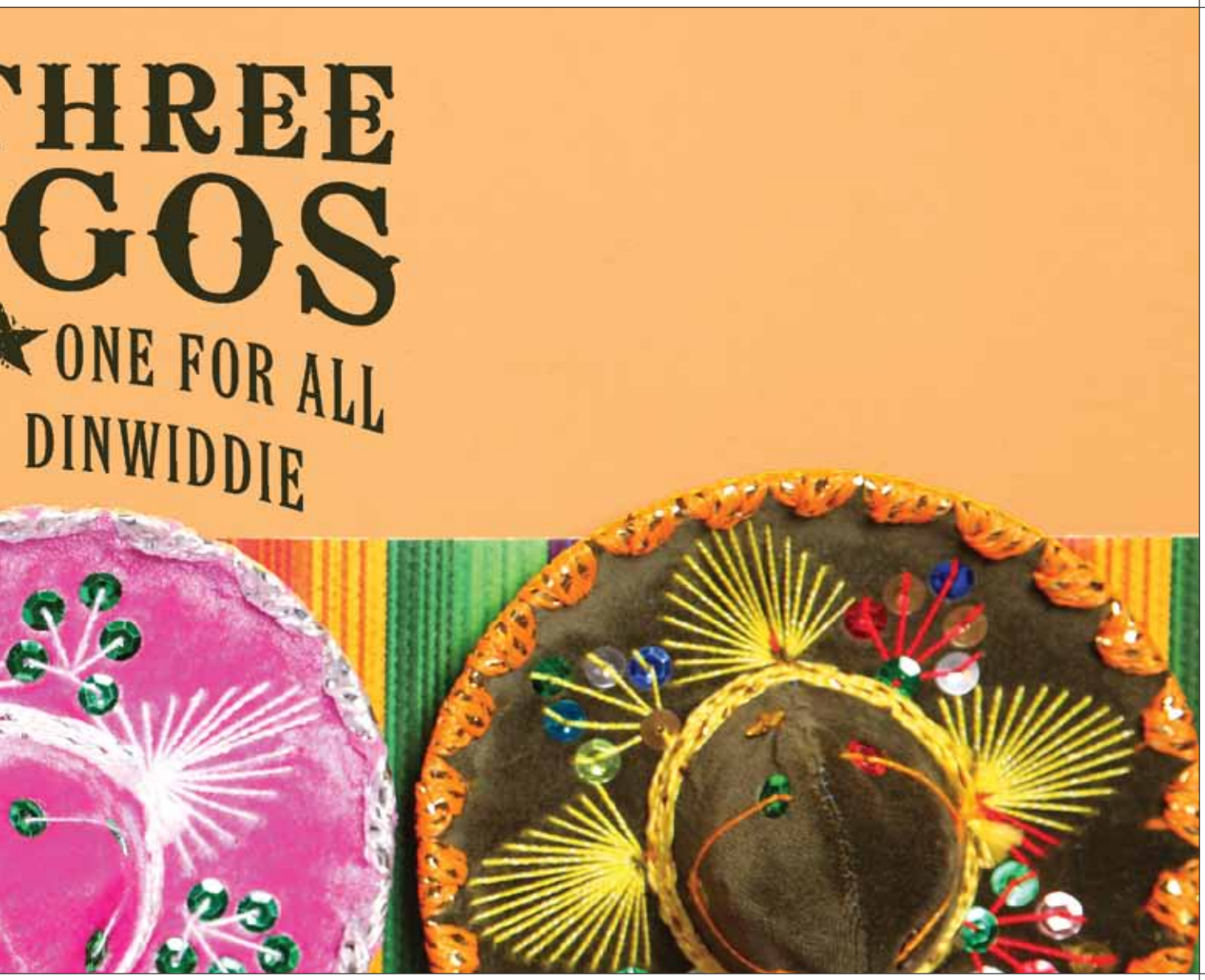
Our story so far...

Tom Tester paged through the story descriptions in the iteration backlog. *The vendor report shall list vendors in the order of percentage they met their monthly sales quota*. "What if the vendor hasn't yet reported their monthly sales?" Tom wondered. "Or what if it's a provisional vendor that doesn't have a set quota? We've got a few cases like that." He turned to the next description: *The sales manager shall be notified of all vendors below 50 percent of quota*. "Notified in what way? Isn't the vendor report a form of notification? How can I test this?"

Sitting a few feet away, Paula Programmer began to implement the Item Profitability report. *The profitability of each SKU in inventory shall be calculated according to the SKU Accounting document on the accounting department fileserver*. She looked at the fileserver directory and saw a number of files. SKU Accounting 2010, SKU Accounting 2011, SKU Accounting-proposed, SKU Accounting-Jane, SKU Accounting-Harry. The one with the latest timestamp was SKU Accounting-Jane, and Jane was the VP of accounting. That must be the right one. Right on page one it clearly said, *Item profitability is the sum of selling prices for the SKU, divided by the sum of item cost, minus 100 percent*. Paula closed the file, thinking "That was easy," without reading the section on page three: *Special calculations for profitability of SKUs used as loss leaders*. She was already programming. "For 'sum of item cost,' I can just multiply 'current cost' by the number of items sold," Paula thought. "I should be done with this story by this afternoon."

Alan Analyst fretted over his notes. Sally Security Auditor had been emphatic that customer details for medical device purchases be protected from easy access. Did these purchase details also need to be encrypted in the database? Could they even do that and still provide the access needed by customer support when the customer called in for help? "I'd better specify encryption; better safe than sorry. The programmers will come back and tell me if it can't be done," Alan thought.

"I wonder when the security team will make the decision as to which categories of products should be treated as 'sensitive'? Perhaps I should just reference their specification docu-



ment. Surely they'll have it finished soon." Alan turned his attention to the unanswered questions regarding marketing tests.

Software development can be a tough and lonely business.

The Three Amigos to the Rescue

One for all, and all for one. We'll have a better shot at getting the right system sooner if we collaborate.

The Three Amigos are the essential stakeholders of the system being developed: The business (or product owner, in Scrum terms), the developer, and the tester. These three represent the viewpoints of what the system is intended to do, what can and cannot be implemented, and what might go wrong or be misunderstood.

These three viewpoints represent orthogonal ways of looking at the system, and I strongly recommend having at least three people involved. It's true that some people are good at seeing things from more than one of these viewpoints. It's tempting to theorize that you can get the same results with fewer people. In actual practice, though, it's hard to give multiple viewpoints their due at the same time. Having different people for each viewpoint will help reduce the number of important points missed.

Don't limit yourself to just three stakeholders, however. Depending on the system being developed or the portion of the system, you may find other stakeholders are essential to provide other points of view. Perhaps you have dedicated user experience experts who have deep understanding of how people use your system and how to avoid misuse. Or it may be important to consider the needs of the systems operations people who keep it running, or the customer service people who handle the calls from customers. Maybe the finance department needs to provide information on tricky aspects of fiscal control. Or the security team needs to ensure that the system is protected from malevolent users. Include these other viewpoints as needed. Just because this approach is called the Three Amigos doesn't mean it's limited to three people. I've often seen a "Three Amigos" consisting of four people.

On the other hand, don't assume that more is better. Only include people who have a stake in the success of the system. Many times there will be multiple programmers and testers on the team. Don't include *all* the developers and testers in

"...a well-functioning Three Amigos will continue the dialog so that any misunderstandings will be discovered and corrected early..."

each meeting, as that dilutes the sense of responsibility. With that dilution, important points might be overlooked. At the same time, the need for everyone to have his say will make the meetings drag on longer. You might have different developers and testers at different times, but only have one representative for each. Changing the individual participants from meeting to meeting helps get everyone involved, gives everyone a break, and keeps it fresh.

Getting the right people—and only the right people—in the room is always preferable for the best results.

How the Three Amigos Work Together

The Three Amigos need to collaborate to define what needs to be done, how they'll know when it's done, and that it's done correctly. After implementation, the Three Amigos need to review the work to make sure it's still correct from everyone's point of view. In between these two points, a well-functioning Three Amigos will continue the dialog so that any misunderstandings will be discovered and corrected early, and so any new insights can be incorporated to everyone's advantage. Here's how it might take place in a typical iterative-incremental agile lifecycle:

Backlog Grooming

Before development of a user story, generally sometime during the previous sprint or iteration, the Three Amigos sit down to discuss the upcoming work. We don't want to wait too late to add details, because the delay will slow us down. We don't want to do this too far in advance of implementation, because the details might get stale.

Detailed knowledge imperceptibly deteriorates over time, and the context changes such that the knowledge may no longer apply. As we learn new things in the process of doing the work, it changes the way we think about the details of the functionality we want. External events can also change what we want the system to do.

The clarification of the planned work is often called *backlog grooming*. Some teams delay discussing the details until they start planning the next increment of work. When this happens, I've generally seen the discussion of what the work includes overwhelm the planning, resulting in a long session, indistinct acceptance criteria, and uncertain planning.

In one or more small group meetings, the Three Amigos examine the planned stories one by one. Starting with the highest priority item, we examine each story, discussing it to gain a shared understanding of the intent. We flesh out that understanding by proposing examples: "Do you mean that when _____ then _____?" "Yes, but if _____ happens then _____." We then look at these examples to determine what's an essential element for the desired functionality and what's an incidental detail required by the chosen implementation. The goal is to come up with examples that cover all of the important scenarios and to boil them down to the essence of the functionality.

During this meeting, take all the notes you find helpful or think might be helpful later. The important thing is to emerge with a complete but minimal set of examples to illustrate the functionality. These examples don't need to be elaborate or detailed, but they do have to be clear enough that they communicate the intent of the functionality.

Planning Meeting

When we get together to plan the next increment of development work, such as the sprint planning meeting in Scrum, the Three Amigos will use these examples to communicate the intent to the full development team. The content of the examples will give the full team a reasonably good idea of what will be involved to implement the user story. The number of those examples will give team members a good idea of its size. If the story is split into multiple stories, the examples clearly indicate which functionality is assigned to which story. This is a big timesaver in my experience. I've seen team discussions go round and round, with some members thinking the story is a small slice and others thinking it includes many other scenarios. If you're estimating the size of stories, the explicit nature of the examples makes it much easier.

"Doesn't this take as long as if the entire team developed the examples?" I hear this question a lot. No, the examples communicate better than does more abstract text. If the examples for a story don't cover a particular situation, either that example describes another story or it's new functionality. Most of the time you'll want to defer that new functionality to give time for serious consideration. Make a note of the example, too. It'll be a big help when you come back to it.

Development and Verification

During development, the examples are turned into automated tests. The user story is not considered complete until the examples pass.

Turning the examples into automated tests should be a pretty easy chore, depending on how you've expressed those examples and the tools you're using. This is best done prior to implementing the story, but might also be done as work begins on the story. Hooking the example up to drive the code under development may have to wait until the story is a little further along. That's OK. Just hook it up as soon as there's an appropriate place to connect it. This work is often a collaboration between tester and programmer.

Exploratory Testing

As each bit of functionality becomes available, you'll want to take a look at it. Does it fulfill the original intent of the

functionality? Does it seem reasonable? Does the functionality still seem desirable when you see it in operation? Does the system suggest unintended ways it might be used? If so, do these unintended operations work in a reasonable fashion and give correct answers?

Passing the acceptance tests does not necessarily mean the software is acceptable. That's just the first hurdle to acceptance. The example scenarios envisioned by the Three Amigos before development should cover the expected functionality pretty well, but no one has perfect foresight. It's always best to bring critical thinking to bear after development, also.

Bringing It All Together

Paula, Tom, and Alan were wrapping up their backlog grooming. Taking a digital photo of their work on the whiteboard, Alan expressed his appreciation to his collaborators. "Thanks for helping me build this list of unanswered questions about the promotional offer feature. I'll take these over to marketing to discuss them. If I can, I'll bring someone from marketing to our next meeting. Until we get better clarity on this, we've got other important features to build. This one isn't quite ready."

Alan reflected on how much easier the process had become since he no longer had to write all the details alone. Paula was much better at math and could easily turn business descriptions into algorithms on her own. Tom had devised some really interesting examples and, between the three of them, they'd been able to agree on which results would be correct in some unusual cases.

Paula felt ready to get started on the new work. She had examples that would clearly indicate when the software was behaving correctly, even when external data feeds were down. She relaxed knowing the start of the next iteration wouldn't be wasted on merely trying to understand the requirements.

Tom was relieved knowing that most of the end-of-iteration functional testing work was done. It would be a quick and straightforward job to put the examples they'd explored together into a form that could be run as an automated test. He was no longer bored at the beginning of the iteration and under extreme pressure at the end when the final functionality was completed. Instead, Tom now had time to do what he considered the fun part of testing—looking for unusual behavior when the system had unexpected inputs. "Alan, when you've got some time, I'd like to show you something I discovered today. The Product Category Profitability report screen seems confusing to me when there are data gaps in both the store results and the purchasing history feeds. Perhaps we can make a simple change to clear it up, or maybe we should take it to the accounting department to define some future work in this area."

What a difference when we all collaborate! Software development is still hard, but not quite as hard as it was. And it's not nearly as lonely. **{end}**

gdinwiddie@idiacomputing.com