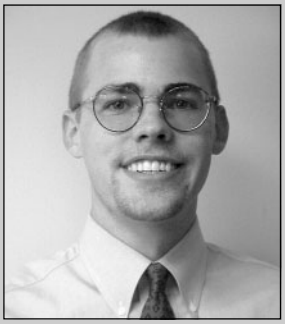
A Developer's Guide to Making Applications More Effective, Easier to Learn, and Simpler to Use

Jonathan Pokress



Jonathan Pokress, CPIM, is an independent consultant and president of Bluenote Consulting Group, Inc. (www.bluenoteonline.com), based in Charlotte, North Carolina. He has spoken on numerous occasions on both technical and managerial topics, including web-enablement with the SAP Internet Transaction Server, development methodology, and user interface design.

(complete bio appears on page 85)

Let's face it. R/3 isn't an easy system to learn or use, at least for most end users. Few of us who have interacted with users over the years would contest this, and most would probably attribute it to the large amount of functionality that R/3 provides. While this is fair — after all, R/3 reflects the unavoidable complexities of our businesses — I propose it is also because, when it comes to designing applications, we sometimes forget our most important role as developers: to be user advocates. It's not our fault. From school onward, we've been rewarded based on whether our applications work, not whether they are effective, learnable, and usable.

So let's change that. Let's make our applications more effective, easier to learn, and simpler to use. There are only two things we need to do to make things right:

- Identify the production applications (and business processes!) that users *currently* struggle with and try to simplify them.
- Use the knowledge derived from our real-world experiences to build better applications going forward.

Sound "pie in the sky"? It isn't. This article will arm you, the developer, with the working knowledge and resources you need to achieve both goals. By bringing together best practices, guidelines, and tools gleaned from both established industry experts[1] and my own experiences, this article will:

[1] Over the last few decades, independent and institutional research from Jakob Nielsen, Apple, IBM, Xerox, Microsoft, ISO, the SAP Design Guild, and others has made great strides in understanding the issues that affect system usability. This article draws heavily on this body of work, aiming to collect, distill, enrich, and organize it for the benefit of R/3 application developers in particular.

Is R/3 Hard to Use?

One example of improving usability is to develop custom R/3 transactions to consolidate information onto one screen that might otherwise span several. However, this begs the question: Is the very need to develop such transactions evidence that SAP has produced an overly complicated product? I propose that this is not the case.

In Release 4.6, SAP has addressed many of the usability deficiencies found in earlier releases. Still, most people I've spoken with still classify R/3 as hard to use. When struggling to post a delivery, for example, it is quite frustrating to run into errors that make no sense to someone like me — a developer often possessing incomplete functional knowledge — and leads to the feeling that SAP could have "made things easier."

But let's be fair. SAP has a very tough job to do. Imagine developing a software package from scratch that accommodates just about every major business process within a modern corporation, in which each piece has to interact properly so that no data is lost. Add on that the package must accommodate both large and small organizations across different industries, and that the software must allow entire modules and functions to be enabled, disabled, or supplanted by external systems. It's remarkable SAP has accomplished so much in the face of these challenges.

Now to the point. When people, myself included, say that R/3 is hard to use, I feel that we're not so much stating that *R/3* is complicated, but rather that *modern business* is complicated. This understanding may not make running transactions any easier, but at least we know where to more accurately place the blame.

- Clarify what makes an R/3 application "easy to use"
- Detail how you can immediately improve the usability of your new applications
- Help you identify currently used applications that can, and should, be simplified
- Explain how to simplify these target applications, including when to use each technique presented
- Provide the tools you need to be successful[2]

[2] The R/3 Usability Toolkit is a set of resources I have assembled from various sources, including my own experiences, and contains a reference of best practices, a questionnaire, and a checklist. This toolkit is available for download from **www.SAPpro.com**.

By the time you finish reading this article, you will have the knowledge and tools you need to make a difference in your own applications. Let's roll up our sleeves and get started.

Understanding the Elements of Usability

A great deal of research has been done on the topic of usability over the last two decades. Researchers have proposed several models to help us better understand the components of usability. But let's skip the theory and jump straight to the good stuff. Empirical data reveals that well-designed

applications have the following 13 elements[3] in common:

1. **Transparency:** Applications should tell users what the system is doing (and has done) at all times, and without excessive delay. Without sufficient transparency (e.g., if a user is not aware that the system is done processing or isn't sure if data has been saved when exiting), tasks seem to take much longer than necessary and data errors become more likely.

2. **Functional Clarity:** In easy-to-use applications, all titles, labels, messages, and other communications use contextual language that users can relate to. Data, functions, and options also have a logical location and order. Think of how many support calls result from vague error messages or illogically placed buttons or functions!

3. **Safety:** Users should be able to move freely within or among your applications. The application should always, however, indicate when data might be lost and (via confirmation prompts, for example) allow users to proceed or cancel the operation as appropriate. Users should also be allowed to enter data in a variety of formats without the system malfunctioning or misinterpreting the data and, when possible, be able to undo their actions. Recall how frustrating it can be when a screen forces you to correct a date format, for example, before allowing you to exit.

4. **Consistency:** Organization, navigation, and terminology should be consistent in applications across the entire system and, when possible, across all systems with which a user will interact. Have you ever noticed a particular R/3 transaction that just didn't "feel right"? Or, if you've ever performed translations, you know how much more work it is when custom programs use slightly different abbreviations for the same term (e.g., *Mat #* versus *Mat no.*).

5. **Flexibility:** Application screens should provide sufficient guidance for inexperienced users without encumbering experienced users. The application should also accommodate varying levels of experience. For example, users new to R/3 (or your particular application) usually find screens with many items difficult to comprehend and frequently turn to online help. In contrast, more experienced users usually want a lot of data and functionality per screen to minimize navigation, and rarely use online help. Try to strike a balance so both types of users can use your application efficiently and effectively.

6. **Efficiency:** Application screens should include only what is necessary. Commonly used functions and data should be placed early within the application to minimize overall navigation. It's important to figure out which business processes, functions, and data will be executed the most and use this as a guide for minimizing the required number of mouse-clicks, tabs, etc.

7. **Visual Clarity and Aesthetics:** An application's interface should promote understanding and the perception of quality. All objects, options, and instructions should be readily accessible, so that the user doesn't need to remember the location or meaning of any elements. This topic is a field in itself, and, in my experience, is one of the most underrated among technologists. An application's look-and-feel directly affects how quickly users can learn it, how useful it is, and how its quality is perceived (which is a reflection on you, the developer). How would you feel about a car with the engine of a Ferrari but the design and aesthetics of a Yugo?

8. **Utility:** The application's data and functionality should address user needs across geographic or divisional boundaries, and anticipate those of likely future users. The application's design should incorporate state-of-the-art UI components to provide the maximum functionality with minimum code, and its final form should be validated against the original design. Perhaps you share my experience that teams often do not spend

[3] The elements proposed here have been compiled, tailored, and expanded from a variety of authoritative sources to apply to R/3 application development. See the resources sidebar on page 85 for more details.

enough time figuring out what to build before they start building it. The result is almost always a product that never quite meets user needs and evolves through a series of Band-Aids.

9. **Constructiveness:** System status messages should be written in terms that are meaningful to users, pleasant in nature, and respectful of user intelligence and experience. They should describe the problem fully and suggest a course of action. Ask yourself how many standard R/3 messages you've come across that are incomprehensible or offer no information on next steps. How did it make you feel? Now ask yourself how many you've written the same way.

10. **Responsiveness:** The application or report should respond to user requests and actions in a reasonable amount of time and, if possible, should allow the user to abandon a process with a long wait time.

11. **Fault Tolerance:** The application should seek to prevent, minimize, and recover from execution errors or the loss of data. Code defensively — always ask yourself questions like: What can go wrong with this code? What happens if no data is retrieved during a SELECT? What happens if a save fails? When an error does occur, the application should, whenever possible, allow the user to recover instead of aborting. Once you are nearly done developing a piece of code, it is a good idea to review all LOOPs, SELECTs (especially those with a FOR ALL ENTRIES clause), CASE statements, etc., and ensure that you have handlers in place just in case the "worst" happens, which it usually does once the application is in production.

12. **Accessibility:** Users should be able to quickly and easily access concise documentation for their specific tasks or questions, empowering them to solve problems and explore the application's functionality on their own. It is important to keep the following considerations in mind: How frequently is the documentation accessed? Are color-blind people disadvantaged by any specific color schemes you used? Also, unless the application is entirely self-explanatory, is both step-by-step and task-oriented documentation easily accessible? Do users know where to find it?

13. **Privacy:** Users should feel confident that private data will remain private, and that no one will be able to "steal their identities" when accessing or posting data. Ask your friendly neighborhood security administrator for the best way to lock down applications — by placing authorization groups in the programs' attributes or assigning authorization objects to transaction codes, for example.

Understanding the elements of usability is an important first step toward building better applications for your users (these 13 elements are also summarized in **Figure 1** for your convenience). My R/3 Usability Toolkit, presented next, organizes its recommendations and heuristics[4] around this important framework.

How to Develop More User-Friendly Applications

While the 13 usability elements are important, they don't provide what developers really need: specific recommendations, metrics, and tools with which to develop better applications going forward. I created the R/3 Usability Toolkit for exactly this reason. The toolkit consists of the following three components:

- ☑ **Usability Best Practices Reference:** This reference is a Word document containing general guidelines and specific best practices for developing easy-to-use R/3 applications. For consistency, the recommendations are grouped by the usability elements just introduced (e.g., "Functional Clarity") and subgrouped by topic (e.g., "Menus"). A table of contents is included so that

[4] *Heuristics* are specific guidelines and rules of thumb. The R/3 Usability Toolkit contains a large checklist of heuristics I've specifically tailored to R/3 development. My hope is that this checklist will enable you to spot about 80% of usability issues before going live with a new application.

Figure 1 Critical Elements of Application Usability

Usability Element	Description
Transparency	Applications should tell users what the system is doing (and has done) at all times, and without excessive delay.
Functional Clarity	In easy-to-use applications, all titles, labels, messages, and other communications use contextual language that users can relate to. Also, data, functions, and options have a logical location and order.
Safety	Users should be able to move freely within or among your applications. The application should always, however, indicate when data might be lost, and (via confirmation prompts, for example) allow users to proceed or cancel the operation as appropriate. Users should also be allowed to enter data in a variety of formats without the system malfunctioning or misinterpreting the data and, when possible, be able to undo their actions if necessary.
Consistency	Organization, navigation, and terminology should be consistent in applications across the entire system and, when possible, across all systems with which a user will interact.
Flexibility	Application screens should provide sufficient guidance for inexperienced users without encumbering experienced users.
Efficiency	Application screens should include only what is necessary. Commonly used functions and data should be placed early within the application to minimize overall navigation.
Visual Clarity and Aesthetics	An application's interface should promote understanding and the perception of quality. All objects, options, and instructions should be readily accessible, so that the user doesn't need to remember the location or meaning of any elements.
Utility	The application's data and functionality should address user needs across geographic or divisional boundaries, and anticipate those of likely future users. The application's design should incorporate state-of-the-art UI components to provide the maximum functionality with minimum code, and its final form should be validated against the original design.
Constructiveness	System status messages should be written in terms that are meaningful to users, pleasant in nature, and respectful of user intelligence and experience. They should describe the problem fully and suggest a course of action.
Responsiveness	The application or report should respond to user requests and actions in a reasonable amount of time and, if possible, should allow the user to abandon a process with a long wait time.
Fault Tolerance	The application should seek to prevent, minimize, and recover from execution errors or the loss of data.
Accessibility	Users should be able to quickly and easily access concise documentation for their specific tasks or questions, empowering them to solve problems and explore the application's functionality on their own.
Privacy	Users should feel confident that private data will remain private, and that no one will be able to "steal their identities" when accessing or posting data.

Demystifying Common User Complaints About R/3

To truly improve application usability, developers need to do something we rarely get (or take) the time to do: listen, observe, and interact with users, who are generally the best barometers of an application's design. Drawing conclusions from user feedback can be tricky, though, since the feedback usually lacks specifics. Thus, to be effective, you need to learn to convert vague comments into specific items to investigate. Here are four common examples.

Example #1: "The System Is Too Complicated"

Possible causes include:

- ☑ **Overload:** The application might be presenting too much data or functionality at one time. Try to restrict screens to no more than 20 fields, 4 pushbuttons, and 5 toolbar buttons.*
- ☑ **Sub-optimal delivery:** The application might be presenting data or functionality in a way that is distracting or difficult to comprehend. Remember to group related data and functionality.
- ☑ **Excess:** The application has unnecessary functionality or data. Remember the 80/20 rule when designing main application screens — i.e., design your main application screens with the data or functions that are applicable to 80% of your audience, and move the highly specialized features to separate screens. Alternatively, consider using authorizations to display only functionality or data that is relevant to certain user groups.

Example #2: "I Don't Understand Where to Get the Information I Need"

Possible causes include:

- ☑ **Conflicting model:** The positioning or grouping of functionality or data doesn't match the user's mental model (e.g., the user identifies "net weight" as a shipping-related field, but it is placed on a tab labeled "Accounting" to accommodate FI users).
- ☑ **Unclear navigation objects:** When navigation objects like menus, tabs, and buttons don't clearly summarize their content or function, users are forced to rely on recall to locate information or functionality. Use clear, meaningful labels to avoid the inevitable "Where do I do this?" phone calls.
- ☑ **Non-intuitive navigation:** The user cannot remember where to find data or functionality, or perhaps the user cannot access it from his or her current location in the application. Try to keep all main navigation points globally visible throughout the application (e.g., through tabs or toolbar buttons).

* These limits are appropriate when designing for experienced users, who can digest far more data and functionality than inexperienced users. Reduce these limits when designing for novice users.

you can find principles relating to specific topics quickly and easily.

- ☑ **Usability Checklist:** The checklist is a practical, comprehensive list of things to double-check for new (and existing) applications. It will help you catch common, high-impact usability issues that can be easily avoided. The checklist is a Word template so you can pull it up and make entries without modifying the original.[5]

[5] When you make changes to a Word template document and attempt to save it, Word asks you for a filename and saves it as a new document instead of modifying the actual template.

Example #3: "It's Too Hard to Do One Simple Thing"

Possible causes include:

- ☑ **Multi-hop process:** Business processes that involve more than one transaction force users to spend a lot of time navigating around between different screens. Also, users often have to memorize data from one screen for use in another. Consider simplifying the process by developing a single wrapper transaction.
- ☑ **Needle in a haystack:** Is the user looking for a specific piece of information in a sea of data? Especially when designing reports, consider whether users are likely to use the data in aggregate or in parts, and aim to help them get the information they need as quickly as possible.
- ☑ **Low performance:** The transaction or report in question might be taking a long time to save or run. Observe the user's actions and propose ways to mitigate the problem if performance cannot be remedied by a technical change (e.g., have the user try specifying more data on the report's selection screen).
- ☑ **Unclear errors:** Check if the user is running into undecipherable error messages or locks and address the problem.
- ☑ **Inability to locate data or functionality:** Refer back to Example #2.

Example #4: "The System Is Telling Me Something, But I Don't Understand What I'm Supposed to Do"

Possible causes include:

- ☑ **Lack of clarity:** The message or visual cue doesn't clearly express what happened. Be sure to explain errors completely and succinctly, and use terminology that is easy for users to understand.
- ☑ **No details:** The message or visual cue tells you what happened, but doesn't clearly explain what to do next. This is common, and you have no doubt experienced it yourself. Provide appropriate details on the necessary next steps.
- ☑ **User inexperience:** The user is told to perform a task that he or she doesn't know how to do. This situation is unavoidable to some extent. To mitigate it, make sure that documentation is available and that users know where to get one-on-one help.

- ☑ **Usability Questionnaire:** Sometimes there is just no substitute for picking users' brains, especially when investigating usability issues with already-deployed applications. Unfortunately, end users aren't always that good at precisely identifying or expressing the sources of their dissatisfaction (for pointers on interpreting user feedback, see the above sidebar "Demystifying Common User Complaints About R/3"). The questionnaire is designed to help you pinpoint areas for further investigation, and can even reveal that the problem is not technical in nature (e.g., some problems are political).

> ✓ ***Note!***
>
> *I created the R/3 Usability Toolkit by combining research from a variety of authorities with my own development experience, and then tailoring it to the specific needs and working environments of R/3 developers. It is by no means a complete work, and I encourage you to expand the tools as your own knowledge and experience grows.*

Next, I'll explain how and when to use each tool in the toolkit and provide tips for setting you on the road to success. For your convenience, **Figure 2** provides a summary of each tool, and of the discussion to follow.

The Usability Best Practices Reference

An essential part of the toolkit, the reference is probably the tool you'll use first and expand with your own ideas. Containing over 200 specific do's and don'ts for building more usable (and functional) applications, the reference is also useful as a basis for developing an IT policy document (e.g., peer-review quality assurance sheets).

To get the most out of the reference, download and read through it while this article is still fresh in your mind. Then, print it out and tack it up on your office or cubicle wall and review it periodically. You can also review specific guidelines when doing a specific task, like writing status messages.

As you will notice when you read it, there are a few themes that run throughout the Usability Best Practices Reference. In the following sections, I present six of the most important themes for you as "lessons," citing what I feel are the most commonly violated best practices for each, and pointing out some useful tips to keep in mind.

Lesson #1: Present Data and Functionality in Digestible Chunks

Frequently violated best practices include:

- ☑ Especially for inexperienced users, try to keep the amount of information on each screen manageable (i.e., a maximum of 20 fields, 4 pushbuttons, and 5 toolbar buttons).
- ☑ To avoid visual overload, display no more than 10 icons on a screen at once for novice users and 15 for expert users.
- ☑ To help streamline the user decision-making process, screens should display only essential information.
- ☑ For toolbar icons, buttons, and menus that rely upon an item being selected, keep them disabled until that item is selected. Where this is not possible, if no object has been selected and the function is executed, issue a "Please select a line and retry" message.

> ✓ ***Tips***
>
> - *Experienced users will be able to learn, use, and benefit from a complex transaction like VA01 (Create Sales/Returns Order) more readily than inexperienced users.*
> - *If a field is needed by only 20% of users, ask yourself if there is a way to detect these users (e.g., via a particular security authorization) and hide the field for all other users.*
> - *When copying fields from a structure or table onto a screen you are designing, it's often tempting to include extra fields that are not really necessary. Avoid this impulse.*

Figure 2 A Summary of the R/3 Usability Toolkit

Tool	Description
Usability Best Practices Reference	What it is/does: • Summarizes guidelines and best practices for developing easy-to-use R/3 applications • Organized around the 13 key elements of usability When and how to use it: • Read it today and periodically reread it • Expand it with your own best practices • Share it with your teammates and managers
Usability Checklist	What it is/does: • Helps you spot common, high-impact usability issues that can be easily avoided When and how to use it: • Designers should skim through it after assembling a prototype to spot issues • Developers should use it after development is complete • Testers should use it during the usability testing phase • Expand it with your own items
Usability Questionnaire	What it is/does: • Helps you assess users' perception of usability and level of satisfaction • Helps you determine what users don't like about a suspect application and solicit suggestions for improvement When and how to use it: • For new applications, distribute it to test users to get initial feedback, and then, a month or more after go-live, distribute it to a small cross-section of users (five or so) as a barometer • For suspect existing applications, distribute it to users periodically after go-live • Expand it with your own items

Lesson #2: Communicate Clearly and Efficiently

Frequently violated best practices include:

☑ Field labels should be brief, familiar, and descriptive.

☑ Make sure that button and tab labels, text elements, and selection texts employ end-user terminology. Interview a cross-section of users, propose several alternatives, and go with the text that is most widely understood.

- ☑ After development, review all possible messages by looking at the application's message classes. Ensure that the messages employ end-user terminology or cross-application R/3 terminology that users will learn during training.
- ☑ Label fields consistently throughout the application.
- ☑ Use color with discretion — only to draw attention, communicate organization, indicate status change, or establish relationships.

> ✓ ***Tips***
>
> - *R/3 applications often use cryptic abbreviations like "CoTL" or "S" (especially for column headers). Provide more descriptive values or more comprehensible tool tips for your users.*
> - *Many applications use colors carelessly, which makes the interface difficult to learn and use. In R/3, this is particularly true with reports. When designing reports, use background/font color sparingly, consistently, and with purpose.*

Lesson #3: Design Strategically

Frequently violated best practices include:

- ☑ When analyzing user needs, remember to identify both current and potential end users.
- ☑ After completing your analysis, including stakeholder interviews, validate your final design parameters and assumptions with users before beginning development.
- ☑ After validating user needs, begin designing the application by creating a prototype that reflects the final product as closely as possible. Only when users actually see what the final product will look like are they really able to conceptualize what might be wrong or needs improvement.
- ☑ Take into account users' likely environments when designing a solution. For example, consider the input device they will be using (PDA, laptop, PC), which determines screen size, color scheme, keyboard functionality, etc.

> ✓ ***Tips***
>
> - *Other user groups will inevitably fall in love with what you did for one group. Avoid building in any assumptions that would make it impossible to roll out the same code to these envious users.*
> - *Always validate business assumptions before you start coding! For example, let's say that a developer meets with users and records the assumption that each delivery item will have only one batch, only to find at a subsequent meeting that this assumption is not true in certain cases. Or suppose a developer records an assumption that all boxes will contain barcodes, and upon review with other warehouse staff discovers that a particular brand of goods has no barcode. Catching these errors before development will save you a lot of time and effort (and headaches) down the road.*

Lesson #4: Develop Prudently

Frequently violated best practices include:

- ☑ For international applications, whenever possible, use field labels that have already been translated.
- ☑ Leverage the most recent user interface elements available in your R/3 system whenever possible.
- ☑ Use radio buttons when only one object can be selected at a time.

☑ Use checkboxes only when more than one object can be selected at a time from a list.

✓ ***Tips***

- *Keep in mind that a wireless scanner might be of more use than a desktop PC for some warehouse workers, and therefore a keyboard may not be used at all.*
- *If "Mat no." has already been translated as an abbreviation for Material number, use that term instead of a new label like "Material no.," which would then also have to be translated.*
- *For reports use table controls instead of step loops and ALV instead of plain lists to provide maximum functionality with the minimum code.*

Lesson #5: Develop Defensively

Frequently violated best practices include:

☑ In reports, encourage users to specify search criteria to ensure finite runtimes.

☑ Be sure to provide a WHEN OTHERS clause for all CASE statements. It will happen in production sooner or later!

☑ Safeguard applications from unauthorized entry by adding authorization groups to programs and authorization objects to transaction codes.

☑ Provide data on a "need-to-know" basis.

☑ Make sure all SELECT, INSERT, UPDATE, and MODIFY statements are immediately followed by an IF SY-SUBRC <> 0 statement.

☑ For SELECT statements with FOR ALL ENTRIES specified, make sure that the internal table used is guaranteed to have at least one entry (see "Tips" below).

✓ ***Tips***

- *If XI_VBAK were blank, the following code fragment could potentially return <u>all</u> records contained in table VBAP:*

```
SELECT * FROM VBAK INTO XI_VBAK
           WHERE. …
  SELECT * FROM VBAP
           FOR ALL ENTRIES IN
                XI_VBAK
           INTO XI_VBAK
           WHERE …
ENDIF.
```

It should instead be coded as follows:

```
SELECT * FROM VBAK INTO XI_VBAK
           WHERE. …
IF NOT XI_VBAK[ ] IS INITIAL.
  SELECT * FROM VBAP
           FOR ALL ENTRIES IN
                XI_VBAK
           INTO XI_VBAK
           WHERE …
ENDIF.
```

- *If a user needs to see only his or her plant's data, use security settings to implement this restriction.*

Lesson #6: Help Users Help Themselves

Frequently violated best practices include:

☑ Tell the user exactly what he or she should do to correct an error.

☑ Prevent users from making errors whenever possible.

☑ Provide training documentation in PDF format on a centrally located, easily accessible, shared drive or on a web site familiar to all users.

✓ ***Tip***

Examples of bad error messages:

- *"You entered a bad date."*
- *"Please save it and retry."*
- *"The order needs to be released."*
- *"You forgot to clear the item."*
- *"Why don't you consider saving first?"*

Examples of good error messages:

- *"Invalid date 2202. Please check your entry."*
- *"Please save the document and retry."*
- *"Please release the order and retry."*
- *"Please clear the item and retry."*

The Usability Checklist

As long as you're in the printing mood, print the Usability Checklist and pin it up next to the Usability Best Practices Reference. Review the items in each section and see if there are any "checkpoints" you can add. Taking ownership of the document in this way will help you quickly become familiar with it. The checklist, like the reference, is also a good template for creating a team-wide peer-review program.

To make a true impact on your applications, it's important to use the checklist *early* in the design process and at key points during and after development. Here's who should be using it and when:

- **Designers (during the application design/ prototype stage):** Whoever is designing the application should skim through the checklist after assembling a prototype to spot any issues. If you currently don't create prototype applications before building them, start. You'll be amazed at how many more major issues you (and your users) will identify and resolve up front instead of after going live! Plus, painting or drawing a screen-by-screen prototype will let you validate usability early on instead of during testing when changes are costly.

- **Developers (just after development but before testing):** After you've finished developing the application, quickly run through the checklist to spot any major issues. Be as critical as possible. Making changes at this point is easy and cheap.

- **Testers (during usability testing):** Testers are accustomed to testing functionality, but rarely have a step-by-step plan with which to validate an application's usability. Distribute the checklist to testers and have them execute it just as they would a functional test plan. Emphasize that the ease with which users can learn and use the application is as much a predictor of its success (and total cost!) as the functionality.

When used thoughtfully, the checklist will become a natural part of your team's application design process. I am confident it will help you improve end-user satisfaction, and relieve much of the unnecessary pre- and post-go-live burden that trainers, the help desk, and developers face when deploying applications with avoidable design issues.

The Usability Questionnaire

While the Usability Questionnaire was initially developed to help you identify problems with applications *currently* in production, it can also be an effective tool for spotting issues in new applications that might otherwise go undetected (i.e., the checklist does not address the issues of suitability to the task or user satisfaction, while the questionnaire does). There are three main phases in which you should use the questionnaire:

What If I Don't Have Time to Use the Full Usability Checklist?

I have to admit that, at 22 pages, the Usability Checklist can be a bit time-consuming (although you will be able to spin through it quickly after a few uses). During development you'll inevitably want to perform a "quick check" in lieu of the full test. In anticipation of such a need, here's an abbreviated list of essentials to check:

- ☑ Is the user kept advised of which data is saved and which isn't?
- ☑ Is the user kept informed of where he or she is and where he or she can go?
- ☑ Are all button, menu, and field labels likely to be clear to users?
- ☑ Is screen information organized in a natural and logical manner?
- ☑ Can the user exit (abandon) the application without having to correct field validation errors?
- ☑ Does the application employ locking to ensure only one user at a time can modify a record?
- ☑ Is the application's use of data, functionality, and labels consistent with other R/3 applications?
- ☑ Does the application accommodate both novice and expert users?
- ☑ Is the information on each screen necessary and sufficient? Is both primary and supporting information included?
- ☑ Is the application designed to minimize mouse-clicks and navigation?
- ☑ Is the application visually pleasing and does it convey a sense of quality?
- ☑ Does the application's visual design promote understanding and avoid overload?
- ☑ Do the application's functionality, data, and design embody the identified needs of its users?
- ☑ Do all messages use language and phrases familiar to users?
- ☑ Do all messages indicate what happened and provide details on what to do next?
- ☑ Will the application's performance match user expectations when exposed to production data?
- ☑ Does the application do everything it can to avoid the loss of data from errors or unanticipated results, and does it recover quickly?
- ☑ Will users be able to access the application easily? Consider availability, input devices, and monitor size, for example.
- ☑ Does the application accommodate people in different locations and those with disabilities (e.g., color-blindness)?
- ☑ Is the application safeguarded from unauthorized users?
- ☑ Is the application designed so that only authorized individuals can see sensitive data?

- **During usability testing:** Print the questionnaire and distribute it to testers along with the checklist during usability testing, but ask them to complete the questionnaire *after a few "clean" test cycles*. After all, we don't want to gauge user reaction to a buggy version of the code, right?

> ✓ ***Tip***
>
> *Also, for best results, try to recruit testers that represent a cross-section of your user population (e.g., across divisions and with varying levels of R/3 experience). This way, if the feedback is consistently positive, you can be confident your application will be well received by the masses, or you can identify subgroup-specific issues and thus focus your training and support resources where they are needed most.*

- **After go-live when things have stabilized:** About a month or two after go-live, distribute the questionnaire to a small cross-section of actual users (about five or so).

> ✓ ***Tip***
>
> *Resist the temptation to distribute the survey via mass email because it will dilute its effect. A personal invitation to a small group, delivered and picked up in person, is the most effective way to ensure meaningful results without creating survey burnout.*

- **Periodically after go-live for suspect applications only:** If through the proverbial grapevine you learn that users are complaining that an application is no longer useful or is hard to use, identify a small survey group that might be able to confirm and elaborate. If you've had success with the questionnaire in the past, consider using it as a starting point for your investigation.

Like all surveys, the success or failure of the questionnaire can depend on technical, social, and political factors. To maximize your chances for success:

- Consider whether a one-on-one conversation would work better for gathering information. Bear in mind your company's culture and how your relationship (or your manager's reputation!) with users might influence their responses.
- Be sure to secure the permission of your own manager and of those responsible for the users you wish to survey before distributing the questionnaire.
- Assure users that the survey is confidential and anonymous (if it is).

> ***Warning!***
>
> *As with any survey, do not overuse the questionnaire! Users will quickly become so disenchanted with the idea of completing yet another survey that you are unlikely to obtain meaningful responses.*

The Usability Questionnaire is designed to help you hone in on the *nature* of user complaints, which you can then use as a basis for further investigation. The basic premise is that users are good at expressing how they *feel*, but not at identifying specific application design aspects that contribute to *why* they feel this way (refer back to the sidebar "Demystifying Common User Complaints About R/3" on pages 68-69). This is your job. Based on research,[6] I have divided the questionnaire into five sections that reflect the primary drivers of the user experience. **Figure 3** provides a summary for your reference.

[6] The Usability Questionnaire is based in part on *Measuring Usability with the USE Questionnaire*, by Arnold M. Lund (STC Usability SIG Newsletter, Vol. 8, No. 2).

Figure 3 The Five Primary Sections of the Usability Questionnaire

Section	Description
Ease of Learning	• These questions gauge how quickly users learn your application's features through training and subsequent exploration, which has a direct impact on their impressions of its usability. • You can aid learnability by using terminology familiar to users and leveraging users' mental models (i.e., models of how data is related) when organizing data and functionality on the screen.
Ease of Recollection	• This section targets how easily users remember what your application can help them do (or not do), and how to do it. • You can aid recollection by labeling data, menus, and buttons thoughtfully and consistently, and by using visual metaphors (e.g., tabs, toolbar buttons, or menus) to remind users of the features and data that can be accessed from where they are in the system.
Effectiveness	• These questions focus on the degree to which your application helps users accomplish the task at hand completely and effectively. • You can make your applications more effective by researching user needs before coding, identifying and validating business assumptions you'll be making, and soliciting input from a cross-section of potential users during the design phase.
Ease of Use	• This section gauges the ease and speed with which users can accomplish specific tasks with your application. • Use the Usability Best Practices Reference and Usability Checklist to avoid common user interface flaws and improve application consistency.
Satisfaction	• This section provides a high-level snapshot by assessing the degree to which users enjoy interacting with your application. • If you've done a good job on each of the previous sections, you should expect good scores on this section. Good marks on the first four combined with bad marks here might suggest that external factors are negatively influencing your application's success (e.g., overall system performance problems, user bias, political issues, etc.).

As with the Usability Best Practices Reference and Usability Checklist, the Usability Questionnaire can become a powerful part of your team's toolbox if used consistently and thoughtfully.

Now that you've got a handle on the principles of usability, along with an understanding of how to develop more user-friendly applications, what about the ones you already have? How can you apply what you've learned so far to your existing applications? That's what we will examine next.

How to Simplify Previously Deployed Applications

While a commitment to building more effective and usable applications going forward is an important

step, to truly help users we need to identify and simplify transactions, reports, or business processes currently in use that are inefficient or hard to use. I call these efforts *usability improvement projects*.

But should developers be the ones to do this? Isn't it somebody else's job? In my experience, we developers are actually perfect for the job, for the following three reasons:

- We usually remember the details of the applications we've designed.
- We know which usability improvements are technically possible (e.g., ALV, dropdowns, etc.).
- After reading this article, we hopefully know what contributes or detracts from an application's usability in the first place.

Hunting for and solving usability problems may be easier than you think. Over the next sections, I will walk you through my suggestions for this process, which involves the following six steps:

1. Secure your manager's approval to investigate.
2. Identify potential targets for simplification.
3. Determine likely sources of usability issues.
4. Evaluate potential solutions for simplification.
5. Propose improvement projects.
6. Prioritize, plan, and execute the improvement projects.

Let's take a closer look at each of these steps in turn.

Step 1: Secure Your Manager's Approval to Investigate

The first step is to ask your manager for some time to talk with users, trainers, and configurators to see if there are any applications or reports that are not serving their intended purpose, that are inefficient, or that are hard to learn or use. Mention that, following the process described here, you will investigate any issues you discover and distill your findings into one or more improvement proposals should you find anything worth pursuing.

> ✓ ***Tip***
>
> *When proposing improvement projects to your managers, you may meet with some resistance along the lines of, "We can't even meet our current workload, let alone review old stuff." While in some cases this is true, I propose that such workload overload is in many cases a result of deploying poorly designed software. Before publicizing your proposals, prepare a response in which you describe the ways in which your suggested improvements will actually reduce current support workloads and, in addition, simplify future rollouts (e.g., through reduced training and support, streamlined processes, better code reuse, etc.).*

Step 2: Identify Potential Targets for Simplification

Once you've got the support of the appropriate managers behind you, you're ready to hunt for issues. Here are three great places to begin looking:

- Trainers and support personnel
- Configurators
- Users

Trainers and Support Personnel

Trainers are a hidden gold mine of experience when it comes to the usability of your company's applications. Through daily interaction with users in multiple locations and with varying backgrounds, trainers can tell you, for example, exactly which buttons users don't remember to push or which messages make no sense.

> ✓ ***Tip***
>
> *Users typically speak more candidly to trainers about what's troubling them than they do to managers (or even developers) since they aren't superiors, and because they have an established rapport. Ask trainers to help you investigate issues with users.*

The first step in approaching trainers is to identify those responsible for each module or business unit[7] and introduce yourself. Let them know you are looking to make their jobs easier by reducing usability issues, and ask the following questions:

1. Which five business processes, tasks, transactions, or reports are the most difficult for users to learn, and why?

2. Which five business processes, tasks, transactions, or reports are the most difficult for users to use, and why?

3. Which five business processes, tasks, transactions, or reports require the most time to execute, and why?

4. Are there any transactions or reports the trainer feels are poorly designed?

Once you've collected the answers to these questions, look for patterns. Some applications are used across business processes and may show up more than once. Some applications will appear on all three of the "hardest to learn," "hardest to use," and "most time to execute" lists. Reduce the lists to (at most) 10 applications or issues to investigate further.

While support personnel often don't remember details about problem applications, they can be very helpful in identifying particularly troublesome applications by helping you review records of past support tickets.[8]

> ✓ ***Tip***
>
> *The key to success is to select those issues that, when solved, will result in high-dollar savings due to positive effects on a large user base or high transaction volume.*

Configurators

Configurators typically know which business processes are particularly long and complex, and are often willing to demonstrate them for you in a test system. Visit with the configurators responsible for each module and business area and have them walk you through the top five business processes in their area that they feel are the most challenging for users to learn or use. Write down the steps that they follow to demonstrate the process, along with the data used, so that you can reproduce the demonstration on your own. For each process, ask a few key users (both expert and novice) to verify the list.

> 💣 ***Warning!***
>
> *Since they rarely interact directly with users, configurators are generally less able than trainers to identify usability issues. It's usually best to have them list potential problems for you to investigate on your own through interviews with trainers and users. However, some configurators do work with users one-on-one during go-lives, in which case they are likely to have the experience and insights you need.*

[7] It is a good idea to solicit the blessing of at least one manager in each business area in which you want to talk to trainers, users, or configurators. You can also ask your manager to do this for you if he or she has a relationship with anyone you wish to speak to.

[8] When reviewing support tickets, pay attention only to those relating to usability issues (e.g., "User misunderstood functionality"). Applications with a large number of functional malfunctions may not indicate a true design flaw from a usability perspective.

Users

Users are often very good sources for identifying problem applications but, as mentioned previously, aren't very good at identifying *why* the problems exist or how to solve them. It's usually best to interview a few key users only *after* identifying suspect applications. Nevertheless, users can be valuable as your first point of investigation if you think this will work best in your company.

In this case, to ensure your efforts will have maximum impact, focus your attention on those groups that use R/3 the most.[9] Interview each user to determine which transactions, reports, or multi-transaction/report business processes they feel are the most difficult to use, learn, or execute. If you've previously consulted with trainers or configurators, verify their suspicions and ask users to elaborate.

Step 3: Determine Likely Sources of Usability Issues

With 10 or so targets in mind, you now need to determine the source of each target's usability issues. The following four sources are the most common:

1. Design flaws
2. Generalization problems
3. Unavoidable complexity
4. External issues

Source #1: Design Flaws

Design flaws — such as buttons with unclear names, too much data on a single screen, or the inability to abort an application without correcting a field validation error — can usually be uncovered using the Usability Best Practices Reference, the Usability Checklist, or the results of the Usability Questionnaire. Have configurators demonstrate the suspect application or process, and use the R/3 Usability Toolkit, along with some old-fashioned common sense, to spot items that might be confusing to users, overly time-consuming, or difficult to remember.

Source #2: Generalization Problems

Is the application cluttered with data or functionality not needed for a particular user's given task? This is a common scenario with standard R/3 transactions since they are designed for such a wide audience and set of functions.

For example, suppose a shipping clerk simply wants to enter a truck ID into a sales order. In this case, transaction VA01 (or VA01N) — Create Sales/Returns Order — is not well suited to the task. To enter the single piece of data, the clerk would have to navigate through multiple screens and large amounts of data and superfluous functionality. While the transaction may be well designed in general, its usability could be improved greatly for the shipping clerk, who might benefit from a transaction variant that brings up the truck ID entry screen as quickly as possible and hides irrelevant data. Alternatively, you could develop a simple, one-screen application that accepts a truck ID and inserts it into the sales order behind the scenes via a hidden BDC (batch data communication). If you feel it would benefit your users, you can even deploy this transaction via the web or as part of a Portal offering.

Source #3: Unavoidable Complexity

Sometimes applications unavoidably reflect the complexity of the business processes they model. When you attribute a usability issue to this cause, you are saying that all users need all the data (or functionality) for nearly all the tasks for which the application is used, and that this data or functionality is presented in the most effective, consistent, and palatable way possible. As you might imagine, this finding is rarely justified.

[9] Configurators usually know the number of users they support, so ask if you're not sure.

Source #4: External Issues

Sometimes the source of a usability issue is not even related to software. You can often detect the presence of these "external" issues by observing users while they interact with the application in a real-world setting. Here are a few examples to illustrate:

- Suppose that pickers in the warehouse tell you that transaction ZPIC is difficult to use. You observe them in the warehouse, and notice that the screen on the wireless device they are using lacks contrast. In the warehouse's low-light conditions, they can hardly see their entries, and therefore consider the application hard to use.
- Let's say a group of shop-floor users report a production confirmation transaction as hard to use. Upon visiting them, you notice that it's difficult for them to use the keyboard to enter serial numbers and quantities. You determine a barcode scanner would allow them to work entirely without a keyboard.
- Imagine that a quality-inspection lab crew reports a result entry transaction as hard to use. Your investigation reveals that they only recently began using R/3 and had no R/3 training.

Most usability issues stem from Source #1 (design flaws) or Source #2 (generalization problems). Since Source #3 (unavoidable complexity) and Source #4 (external issues) occur only occasionally, if at all, and are highly dependent on specific scenarios and intangibles, I will not cover their resolution here. As the first two can be addressed simply by software changes, I'll discuss what you can do for these two cases next.

Step 4: Evaluate the Potential Solutions for Simplification

The range of solutions available depends on whether the usability issue stems from a design flaw or a generalization problem. Here, I will outline the potential solutions available along with when to use which (for a detailed summary of these potential solutions, see the appendix on page 86).

Solutions for Usability Problems Stemming from Design Flaws

If one or more design flaws are at fault, consider these potential solutions, and in this order.

☑ **Solution #1: Review the configuration and/or security model.** Developers frequently roll up their sleeves and set off to solve problems that can actually be addressed through configuration or security changes. This is true both with standard R/3 and custom transactions (especially those coded by other developers). Some transactions allow fields or tabs to be enabled or disabled, fields or buttons to be relocated between screens, and labels to be changed via configuration or by tweaking security profiles. Be sure to talk with configurators, review the application's configuration documentation yourself, and even review the application's code in search of hidden features before assuming development is required to correct any issues.

☑ **Solution #2: Review OSS Notes for updates and/or report the issue to SAP for repair.** Chances are, if your users are experiencing trouble with a standard R/3 application, others are too. Be sure to check for any OSS Notes that might address the issue.

> ✓ ***Tip***
>
> *Also research whether the issue has been or will be addressed in a subsequent R/3 release.*

☑ **Solution #3: Make improvements to the application yourself.** For custom applications, this is usually much easier, although you still have to consider how the improved version will be deployed

without interrupting production or causing confusion among existing users. For standard R/3 applications, consider the availability of user, menu, and screen exits (recently renamed "Business Add-Ins"), or consider modifying the standard R/3 object[10] by applying for a modification key via SSCR.[11]

☑ **Solution #4: Use one of the "specialization" techniques.** In some cases it's better to address major design flaws by creating a separate, "specialized" transaction for use by a group or subgroup of users, rather than modifying the original (e.g., if a small subgroup of users has needs distinct from the main group, or if you wish to make large design changes to a standard transaction). Fortunately, this approach does not always involve developing code. I will discuss the specialization techniques in detail in the next section.

Solutions for Usability Problems Stemming from Generalization

For generalization problems, you can simplify complex applications/processes through "specialization." Consider the following options in the order in which they are presented here.

☑ **Solution #1: Use transaction variants to specialize complex transactions**. First introduced in R/3 3.x, transaction variants provide a way to customize standard R/3 transactions by hiding fields and functions, skipping screens, and defaulting field values. You define a variant using a simple recording tool (transaction SHD0) that walks you though each screen in the target transaction, prompts you for settings, and assigns a transaction code to the variant, which you provide to users instead of the standard code. Variants are an easy, low-commitment way to tailor R/3 transactions to the specific needs of a group, or to provide a user with multiple, specialized views of a single R/3 transaction.

☑ **Solution #2: Develop custom ABAP transactions to streamline, enhance, or combine transactions and reports.** If major flow changes are needed to simplify an application's design, or if you wish to combine several transactions or reports into a single, streamlined application, consider developing a custom ABAP transaction. The transaction can interact with users in a specialized, optimized, and personalized way, and then load the data into R/3 via a synchronous batch data communication (BDC) against one or more standard R/3 transactions. The downside to this approach, of course, is the necessary development time and the introduction of yet another custom application that will require training and support. Nevertheless, in many cases the cost can be justified (e.g., if a minute or two can be eliminated from a business process executed 2,000 times daily). See the sidebar "Why Develop Custom Transactions?" on the next page for more on why developing custom transactions is a good idea.

☑ **Solution #3: Develop web-based applications to leverage the usability strengths of the web paradigm.** Learning R/3 in the first place can be difficult for users and expensive for companies. While training is a wise investment for those likely to use R/3 frequently, the costs can outweigh the benefits for those who will use it only occasionally. This has led many companies over the last few years to build and deploy web-based R/3 applications.[12] Consider a

[10] Modifying standard R/3 objects has become much easier and less intimidating with the introduction of the Modification Assistant in 4.5B. Prior to the Modification Assistant, it was difficult to distinguish modifications from standard code, and reapplying them during an upgrade or support package import was a manual process. For this reason, teams rarely allowed modifications. If you have 4.5B+, modifying R/3 objects is now an easy and viable option for implementing large-impact changes. For more detail on modifications, see the article "The Basics and Beyond: Manage Modifications Effectively with the SAP Modification Assistant, Modification Browser, and Object Adjustment Tools" on page 89 of this issue.

[11] To modify standard R/3 objects you need an SSCR (SAP Software Change Registration) key, which you can request from the SAP Service Marketplace at **http://service.sap.com/sscr**.

[12] The primary motivations here are: (1) most users are familiar with browsers and Internet applications; (2) Internet applications tend to be graphical and intuitive; and (3) information from disparate systems can be combined on a single web page. In addition, the introduction of the SAP Internet Transaction Server (ITS) and, more recently, the SAP Web Application Server (Web AS) has further encouraged the development of web applications. With R/3 Enterprise, the Web AS is actually integrated directly into R/3.

Why Develop Custom Transactions?

When organizing R/3's functionality, SAP needed to keep things generic to accommodate a variety of industries and, perhaps more important, to accommodate end users with widely varying roles within individual companies. The downside to this is that the user experience is muddied by functionality irrelevant to his or her needs. For example, a shipping clerk whose role requires modifying one or two fields in a delivery will use the same generic Change Delivery transaction (VL02N) that is used by a customer service representative. The shipping clerk will see tons of tabs, fields, menus, and buttons totally unrelated to the task at hand.

This user interface complexity and the fact that many business processes require a user to visit several different screens across multiple transactions are the two largest issues of R/3 usability. Ideally, users would have a small number of transactions to accomplish their job functions. The transactions would provide them with the information (or ability to input information) they need, when they need it, on one or two screens.

This need can be addressed in a number of ways. One is through strategic, well-designed custom transactions that consolidate information retrieval and data entry in a role-specific way. If deployed to the web through technologies such as the SAP Internet Transaction Server (ITS) and SAP Web Application Server (Web AS), this role-based (or even user-level) personalization of the user interface is often easier to achieve.

web-based solution if your users are new to R/3, only use it occasionally, or use a very limited scope of its functionality.

☑ **Solution #4: Develop wireless applications to leverage the usability strengths of different platforms.** Using the same technologies that enable web-based applications,[13] applications that run on wireless devices (modified for industrial use) provide many users with easy access to R/3 data or functionality from desktop-unfriendly locations like warehouses. Many modern devices contain a web browser designed to access a slimmed down version of a web-based application. If your users find it difficult and inefficient to use desktops, a web-based wireless application can be a powerful solution.

☑ **Solution #5: Develop custom fat-client applications.** When particularly frustrated with R/3 (usually when it comes to reporting), companies often introduce fat-client applications — such as PC-based C, C++, C#, Excel, Access, or Visual Basic applications — to store and forward information, or to download R/3 data for local reporting. While this technique gets the job done in the short term, it is an antiquated approach (at least for enterprise computing), it is very expensive to support and enhance,[14] and, personally, I discourage it. When a custom application is needed, develop it as a client/server application instead — for example, as an ABAP program or web-based application. One exception to this is if you are using desktop applications for offline data entry or access, in which case fat-client applications are an acceptable solution.

[13] In addition to the ITS and Web AS, several third-party products (including those that utilize the terminal-based SAPConsole interface) exist for building wireless solutions. Despite a limited amount of marketing in this area, the ITS and Web AS are equally, if not more, powerful platforms for building wireless solutions, and in my opinion are far more strategic for R/3 shops in the long run.

[14] One of the main reasons is that it is very difficult to roll out PC-based software and manage software versions across thousands of computers. Also, working with data locally sometimes introduces complex synchronization issues.

Step 5: Propose Improvement Projects

Once you've determined strategic usability targets, identified likely causes, and assembled a rough plan of attack for each, present your findings to the managers who approved your interviews. It's best if you can also present your ideas to the people you interviewed. This way, if the majority see value in your proposals, there will be social pressure to make these "optional" changes "required" changes. If you present to only one manager, you run the risk that he or she will shoot down your ideas because of the "no time" argument and all will be for naught. This scenario may sound cynical, but it's best to be prepared. We want to be sure to prevail for the common good, right?

Step 6: Prioritize, Plan, and Execute the Improvement Projects

The last step is to work with your managers to prioritize, plan, and execute the improvements. Be sure to follow up (e.g., using the Usability Questionnaire) after you've implemented an improvement to verify that your actions have made a difference. This is the most rewarding part!

Tips for Planning Simplification Projects

Simplification projects can do a lot of good, but you must take care that you are actually helping things instead of creating a lot of extra work without yielding significant improvement. The following are some key tips to keep in mind as you set out to improve your own applications.

☑ Trainers are a great sounding board for usability improvement ideas. A good rule of thumb is to convince trainers that your improvements will make things substantially easier — for users, for the support staff, and for future training. If you can convince them, then you have good reason to feel confident that your improvements will succeed.

☑ When proposing improvements to a live application, consider how changes will affect current users and your teammates. Sometimes trainers become justifiably anxious when developers mention making interface changes to live applications.

☑ It sounds obvious, but be sure to check OSS Notes for solutions to usability issues with standard R/3 applications. While SAP usually doesn't recognize or release patches for usability issues, it's worth a try.

☑ Transaction variants are a great low-commitment, low-maintenance way to create several simple transactions from a monster-like transaction, such as VA01, without custom code. You can even provide a single user, for example, with five or so variants for the same monster transaction, each used for a specific task.

☑ When contemplating a simplified web-based or wireless application, consider where your users will primarily work — in the SAP GUI, in a web browser, on a device mounted on a forklift, etc. Also, consider which environments they are most familiar with. Due to their simplicity and pervasiveness, web-based and wireless R/3 applications are especially effective for delivering R/3 functionality to groups of users that are new to R/3 or are in remote locations.

☑ Modifying standard R/3 code is, of course, the least preferable option, but should be considered when reasonably small changes would yield large usability improvements. However, many R/3 shops have a blanket "no modifications allowed" policy, fearing large upgrade and maintenance costs. In my opinion, such policies are overcautious, and merely shift the costs of maintenance to production, training, and support teams as users struggle unnecessarily with unwieldy applications. Applying for a

Resources

- ☑ Jakob Nielsen and Robert Mack, *Usability Inspection Methods* (John Wiley & Sons, 1994, ISBN 0-47101-877-5)
- ☑ Jakob Nielsen's Usability Heuristics web site (**www.useit.com/papers/heuristic**)
- ☑ Elaine Weiss, *Making Computers People-Literate* (Jossey-Bass, 1994, ISBN 1-55542-622-0)
- ☑ Arnold M. Lund, *Measuring Usability with the USE Questionnaire* (STC Usability SIG Newsletter, Vol. 8, No. 2)
- ☑ Deniese Pierotti, Xerox Corporation, *Heuristic Evaluation - A System Checklist* (**www.stcsig.org/usability/topics/articles/he-checklist.html**)
- ☑ Usability Special Interest Group, *Usability Toolkit* (**www.stcsig.org/usability/resources/toolkit/toolkit.html**)
- ☑ SAP Design Guild, *Simplifying for Usability* (**http://sapdesignguild.org/resources/simplification**)
- ☑ The ISO web site (**www.iso.com**)

modification key at **http://service.sap.com/sscr** is simple and quick, and your modifications can almost always be backed out.

Now, go forth and make things better!

Conclusion

As developers, we instinctively know that usability is important. The ease with which an application can be learned and used, as well as how it is perceived, is a critical but often underrated determinant of its success (or failure). My aim in this article was to provide you with a foundational knowledge and the tools you need to build better, more usable applications, and to make a positive difference in the daily lives of your users. You can download the R/3 Usability Toolkit from the "Download Files" section at **www.SAPpro.com** to help you on your way.

The importance of becoming a "user advocate" cannot be overstated. If you agree, share this philosophy with your colleagues. In my experience, modern corporate IT environments sometimes settle for levels of mediocrity unless someone injects ideas and enthusiasm. If you've read this far, you might just be the one for the job in your own organization.

Jonathan Pokress, CPIM is an independent consultant and president of Bluenote Consulting Group, Inc. (www.bluenoteonline.com), based in Charlotte, North Carolina. He has spoken on numerous occasions on both technical and managerial topics, including web-enablement with the SAP Internet Transaction Server, development methodology, and user interface design. Bluenote assists clients in the areas of R/3 simplification, development of web and wireless solutions, quality assurance, and mentoring. You can reach Jonathan at jpokress@bluenoteonline.com.

Appendix : Potential Solutions for Mitigating Usability Problems

Solution	Pros and Cons	When to Use It
Solutions for usability problems stemming from design flaws		
Solution #1: Review the configuration and/or security model for options.	**Pros:** • Easy to implement. • Low commitment. • Look-and-feel can be customized by location or user subgroup. **Cons:** • Usually can be used only for trivial interface improvements (e.g., show/hide fields or tabs based on user authorizations), if even that much control is possible.	• For both standard and custom applications: Always consider this option first. Even custom applications are sometimes driven by one or more configuration tables or customize themselves based on a user's security authorizations. • Research the configuration/security options available: Ask configurators, review documentation, visit **http://help.sap.com**, or search the code for authority-check statements or references to key configuration tables.
Solution #2: Review OSS Notes for updates, and/or report the issue to SAP for repair.	**Pros:** • OSS Notes are easy and inexpensive to implement. • SAP develops and supports additional code and code changes required for improvement. **Cons:** • SAP rarely releases code fixes related to usability issues.	• For standard R/3 applications only: Checking OSS Notes takes very little time and can save you tons of coding.
Solution #3: Make improvements to the application yourself.	**Pros:** • Major improvements are possible with relatively little effort or expense. • Easiest solution for custom applications that have not yet been rolled out, and is usually easiest for previously deployed custom applications as well. • Many minor improvements can be made to a standard R/3 application's interface or functionality via user, menu, and screen exits (now Business Add-Ins) or direct modification (with an SSCR key). **Cons:** • For applications currently in production, you need to cut-over to the new version, which involves some risk. You also may need to retrain users. • Modifications to standard R/3 applications are costly to support and make upgrades and maintenance more difficult (this is less so with the introduction of the Modification Assistant in 4.5B).*	• For custom applications: This is the best way to address functional and interface issues. Cost is lowest for issues caught and addressed before the application goes live. • For standard R/3 applications: Use this method only when the needed changes are minor (e.g., one field to add or one menu or button title to change). Major modifications, even if implemented as user exits, can become expensive to maintain and can hinder upgrades or support package imports.

* For details on the Modification Assistant, see the article "The Basics and Beyond: Manage Modifications Effectively with the SAP Modification Assistant, Modification Browser, and Object Adjustment Tools" on page 89 of this issue.

Solution	Pros and Cons	When to Use It
Solution #4: Use one of the "specialization" techniques (see the generalization solutions below).	**Pros:** • For both custom and standard applications, users may benefit from highly specialized applications. • When specialized, custom transactions are developed to "wrap" standard, complex transactions, you have complete control over when and how the user interface changes. During upgrades, only the touchpoints between the specialized application and standard R/3 will need to be updated. **Cons:** • If you develop one or more custom, specialized applications (if transaction variants won't do the job), be aware that they can be expensive to develop, maintain, and upgrade. Training, support, and delays resulting from usability problems are also costly, however. When considering this approach, be sure to enumerate the costs of doing nothing versus the costs of developing and maintaining specialized applications.	• For custom applications: Use a specialization technique when a complex custom application needs to be simplified for a group of users, or when a small subgroup needs only a smaller or slightly different set of data or functionality than the majority. • For standard R/3 applications: Use a specialization technique if the needed changes are major (flow or logic changes) or numerous. Major changes are costly to maintain and can hinder upgrades. Many usability issues with R/3 applications can only be addressed via specialization. • For both standard and custom applications: Consider deploying highly specialized transactions to users instead of, or in addition to, the standard, either to simplify things for novice users or to provide a quick, limited-purpose application for experts (although experts may also benefit from the more complex application in certain circumstances).
Solutions for usability problems stemming from generalization		
Solution #1: Use transaction variants to specialize complex transactions.	**Pros:** • Low-commitment, low-maintenance — you can create, delete, and maintain variants using standard visual tools. • No custom development required. • Screens can be skipped, fields made display-only, and values defaulted. • You can even provide a single user, for example, with five or so variants for the same monster transaction, each used for a specific task. **Cons:** • Aside from skipping screens by pre-populating data, difficult to change an application's flow.	• Great for providing a limited scope of data or functionality to new users or users only needing a piece of a complex transaction (e.g., VA01).
Solution #2: Develop custom ABAP transactions to streamline, enhance, or combine transactions and reports (can execute originals via synchronous, hidden BDC).	**Pros:** • Complete ability to merge data and functionality from one or more transactions or reports into a specialized, easy-to-learn-and-use application built around the specific needs and experience levels of one or more groups of users. • You can supplement standard R/3 data and functionality with your own. • User interface is stable across R/3 upgrades. Only interfaces to standard transactions are changed. **Cons:** • Involves developing additional software, so development and support costs can be significant. • Custom applications will not automatically support functional enhancements available during upgrades, thus requiring additional coding.	• Consider this approach only when it cannot be accomplished through less-costly means. • Consider this approach if major modifications to standard R/3 applications would be required. Compared to modifying a standard R/3 application, a custom application might be less costly to support, cause fewer upgrade difficulties, and provide more flexibility in terms of the functionality and data you can offer. • Consider this approach when the needs of a small subgroup of users are vastly different than the majority who use the more complex transaction.

Solution	Pros and Cons	When to Use It
Solution #3: Develop web-based applications to leverage the usability strengths of the web paradigm (using the Web AS, ITS, JSP, ASP, etc.).	**Same pros as generalization solution #2, plus:** • For users new to R/3 but familiar with web applications, R/3 functionality can often be deployed without the significant training, support, and licensing costs (in some cases) associated with a traditional R/3 rollout. • Web-based applications can be made available via an extranet, providing highly restricted access to R/3 data and functionality to business partners or employees via the Internet. • Web applications can easily integrate R/3 and non-R/3 data in a single, seamless presentation. **Same cons as generalization solution #2, plus:** • May involve setting up additional servers (e.g., the Web AS or ITS) unless you upgrade to R/3 Enterprise, which has the Web AS built-in. • Developers need to learn web-development skills (i.e., web design, HTML, JavaScript, etc.). • Establishing an extranet can be costly and must be done with caution (e.g., security concerns).	• Consider developing web-based applications if a population of users has no experience with R/3 and will only need access to a small amount of R/3 functionality and/or data. • Consider this approach when you need to provide external parties with access to R/3 data or functionality, and the Internet would be a convenient, user-friendly, and cost-effective way to do so. • Consider this approach when you wish to seamlessly integrate data from multiple systems into a single application with a consistent look-and-feel.
Solution #4: Develop wireless applications to leverage the usability strengths of different platforms.	**Same pros as generalization solution #3, plus:** • Wireless devices can be easily transported, providing mobile users (e.g., warehouse or production floor personnel) with the data and functionality they need, exactly when they need it. **Same cons as generalization solution #3, plus:** • Wireless devices tend to be expensive and battery life becomes a concern.	• Consider this approach when users are mobile and would be able to do their jobs more quickly or more accurately by having R/3 data or functionality at their fingertips.
Solution #5: Develop custom fat-client applications (e.g., using C, C#, C++, or Visual Basic).	**Same pros as generalization solution #3, plus:** • Desktop applications tend to be very fast (except when they must constantly access backend data). **Cons:** • Desktop applications (especially C, C#, and C++ applications) can be more expensive and time-consuming to develop than ABAP applications. • Upgrades and maintenance of desktop applications is typically much more costly and difficult than it is for ABAP applications, mainly due to the number of systems that need to be upgraded.	• Avoid developing custom desktop applications to meet enterprise processing needs whenever possible. Instead, choose one of the other client/server options above. • An exception is when users must do a lot of offline processing or data entry, a mode in which both web and traditional SAP applications don't work as well. This need is rare, though, since most modern enterprise processing relies extensively on online access to enterprise data sources.

