Implementing Dashboards for a Large Business Community

A Practical Guide for the Dashboard Development Process

Doug Calhoun and Ramesh Srinivasan

Doug Calhoun is systems analyst, claims technology—data and information delivery at Allstate Insurance Company.
dcal9@allstate.com

Ramesh Srinivasan is manager, claims technology—data and information delivery at Allstate Insurance Company.
rsri2@allstate.com

Abstract

Dashboards are becoming more prevalent as business intelligence tools, and the reason is obvious: well-designed, accurate dashboards can quickly communicate important business indicators and trends and provide actionable information.

However, creating and implementing a successful dashboard involves a great amount of work. It often requires implementing tight controls while allowing the flexibility needed to test and learn with the business.

This article outlines tips for how to integrate these seemingly divergent processes as well as how to ensure the data accuracy, ease of use, and optimal performance that make a truly successful dashboard.

Introduction

The use of dashboards as a primary business intelligence tool is expanding quickly. When supporting a business unit fueled by data, how does an application team build dashboards that will provide great business value and be sustainable?

There are many methods for doing this, as we will explain in this article. However, there are also certain fundamental principles that may seem obvious, but can be difficult to implement:

- Engage business users, not just at the beginning and end of a project, but throughout the entire process. Make business users your partners.

- Involve the entire application team throughout your project's life. A factory-like approach of handing off tasks from phase to phase will not work well.
- Although design updates may require an iterative approach with business users, the number of components needed should drive the team to define phases and key deliverables early in your project to keep it on track.
- Sophisticated user interfaces are great, but in the end, it's really about the data. Ensure that everyone is in agreement about how to define the data from a business point of view, and create a plan for how to validate it.
- Ease of use is critical. Make sure your business partners get hands-on opportunities as often as possible.
- Design your technology based on the number and types of users. Performance and capacity should be considered when designing and building dashboards, much as they are with more traditional transactional systems.

This article is not intended to serve as a guide to visual design. That topic has already been studied extensively and successfully. We will discuss best practices for the *process* of creating a successful design.

In addition, the word *dashboard* is used here as a general term for data visualization tools showing at-a-glance trends and other indicators. It is not meant to signify the timing or refresh rate of the data, and could be used interchangeably with "scorecard" depending on how a business unit chooses to define it. In the business intelligence world, "dashboard" has become the most common term, so it will be used here with assumed broader connotations.

Another concern is process methodology. Many companies primarily employ a waterfall life cycle, which can be a difficult fit for a business intelligence implementation. However, a purely agile methodology for dashboards can also lead to trouble, as there are complexities with development and testing that require a certain level of more traditional phase-gating.

Essentially, the dashboard needs to be treated both as an application (with all the functional testing required) as well as a mechanism for providing data, including iterative testing and prototype updates. A certain level of flexibility in your development process may be required to achieve a happy medium and ensure a successful rollout.

Sophisticated user interfaces are great, but in the end, it's really about the data. Ensure that everyone is in agreement about how to define the data from a business point of view.

Depending on the size of your company, you may also need to leverage the assistance of other technology groups to implement. Where appropriate, involve groups such as your business intelligence community of practice or center of expertise; data, solution, and/or BI architecture; database administrators; all associated infrastructure/server administrators; change and release coordinators; and any other applicable groups you believe should be enlisted. Do this early.

All of this may require "innovating your process," which might sound like a contradiction in terms to process methodologists but may have practical application to your work. The best practices below will guide you in the direction that best fits your project's needs.

Starting the Project

If you are embarking on a dashboard project for the first time, there are several rules of thumb you should follow at the project's outset.

First, as with any project, you will need to define team roles and lay the groundwork for how the project life cycle will work. At the same time, you will need to identify and engage all stakeholders and ensure both groups agree on expected outcomes.

It is unlikely that you will be working on a dashboard project without a business case behind it, but getting a request from the business and truly engaging users as partners are two very different things. Although it can be easy to take orders and make assumptions, keeping the key business partners involved throughout the entire project life cycle, and beyond, is absolutely essential.

Finally, you will need some vital information before you begin. Some questions are obvious from a technical point of view. What are the data sources? Will data be stored separately? What tools will be used? What environment(s) must be built? Other questions are just as vital, but may not be so obvious. For example, is the project feasible, especially as an initial effort?

We recommend you limit the scope of an initial dashboard to a simple, straightforward first effort that has high business value. This way, a quick win is more possible, success can be attained early, and business trust will be earned as you "learn the ropes."

You will also need to be sure that the project is appropriate for a dashboard or other visualizations. For example, if the business primarily wants to track how hundreds of their individual workers are performing, a dashboard is likely not the right vehicle. However, if they want to track how their offices are performing over a period of time, using standard, well-known measures within the company, then a dashboard may be the best option. (You can still consider getting to the individuals' detail, which we'll discuss shortly.)

The main lesson here, and throughout the early phases of your project, is to ask questions and keep on asking them! If something does not make sense or seems impossible, work with business users until you reach a mutually satisfactory agreement.

Once the project looks possible, list all your assumptions—whether business related, technical, or process/project based. You'll need this list to build an order-of-magnitude estimate, define the technical space you will be working within, and help business users understand their role during the project (and how crucial it is). Having everything in order even before detailed requirements are determined will give both you and business users confidence. After all, before you start involving them in detailed requirements meetings, they're going to want some idea about when to expect a finished product.

Limit the scope of an initial dashboard to a simple, straightforward first effort that has high business value.

Finally, as you devise this plan, treat the dashboard as a full-blown application. Although the dashboard is built in the business intelligence space, it has both the complexity of a dynamic user interface (with the myriad possibilities of errors on click events), as well as the need for absolutely exact, gold-standard data. Both the data and the functionality will need to be tested thoroughly, as if you were developing a transactional application. If you release the slickest, most attractive dashboard your business has ever seen but the data is wrong or a button doesn't work, user confidence will quickly erode. Your dashboard may be pretty—pretty meaningless.

Consider the metrics and aggregations needed and what types of structures will be required to support your project. Depending on your company's standards, you might be using denormalized tables, dimensional tables in a warehouse (or a combination of these), an integration of detailed and aggregated data, OLAP cubes, or many other possible sources and targets. As with any BI solution, you need to choose the appropriate data model.

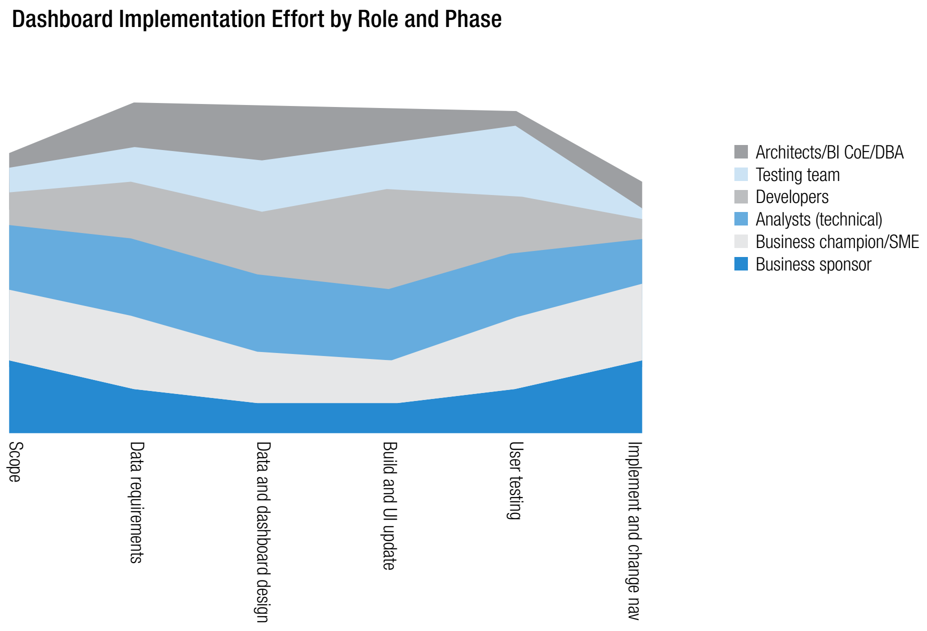

Figure 1. Both business users and technical staff should be involved throughout a project's life cycle.

The point here is that performance is paramount for user adoption.

Document and agree upon functional requirements and data definitions while offering the flexibility of iterative testing and tweaking that a business intelligence solution should provide. It is critical to lock down the logic behind the displayed metrics early in the project. If that changes or is vague to everyone, there is little chance you'll deliver a successful dashboard.

Gathering Requirements

Involving business users in your work is crucial—and most clearly needed—early in a project, especially during requirements gathering and scoping. You may need to remind yourself to keep your business partners actively involved, because it's vital to your project's success!

Multiple meetings will certainly be necessary, but make sure to keep users actively engaged via various methods, including whiteboarding at first, dashboard prototyping later, and sharing early data results. This will not only help hone the requirements, but also allow business users to feel they are truly partnering on the project. This will ensure that they know and trust what they will be getting.

In addition, the entire development team should be involved from inception through implementation to ensure nothing gets lost in translation through the work. See Figure 1 for a gauge of both business and technical involvement through a general project life cycle (regardless of the specific methodology used).

The following best practices can help you avoid pitfalls during requirements gathering, even when the relationship with the business is good.

Know your user. It is possible that your business partner may represent only one part of the larger group using the dashboard, or may be assigned to a project and may not be an ultimate end user at all. Some users may have different business needs from your primary business partner. Make sure that you define all the groups of users who will have access to the dashboard, and ensure all of their voices are heard. This is not as easy as it sounds, but is worth the effort.

Define a use case for every component you build. There is no point in creating a dashboard component unless there is a direct use for it that can be easily defined and documented. Documenting the requirements is crucial to ensure business users get what they have asked for and so developers and testers have a clear guide about what they must build. You want to ensure that the use cases, and the data shown, will stay meaningful over time for each component; it is not a good idea to introduce new or rarely used metrics with a dashboard solution. Finally, require sign-off for all use cases, business requirements, and scope documentation you create. The scope should be limited to the business metrics and granularity of the data at this stage; visualization requirements can be developed later.

Define all the groups of users who will have access to the dashboard, and ensure all of their voices are heard.

Know your data sources and plan your approach. You must understand both where the data initially resides and, if you use an extract-transform-load (ETL) or similar process, where it will eventually reside. If storing the data, you will need to know how it should be stored, how long the data will need to be available for access, and how often it needs to be refreshed. Especially if using ETL, three-quarters of your work will be spent on the analysis, load build and testing, and validation of the data. Even without ETL, our experience is that the majority of the time should be spent working with the data rather than building the front end. Given the visual nature of dashboards, it is easy to assume that the bulk of your work is spent building attractive, user-friendly interfaces. This is simply not the case with successful implementations, especially when so many easy-to-implement dashboard tool suites are available.

Include only trusted, known metrics whenever possible. Exceptions may arise, but if metrics are well known, the exceptions will be much easier to validate. The sources of the data must also be trusted, and business users should be included in selecting data sources.

Know your refresh rate. Will the dashboard be loaded monthly, weekly, daily, hourly, or a combination of these frequencies? The fundamental dashboard design approach will depend on your answer to this question. Use cases will drive your design. Make sure you have thorough discussions about what is really needed versus what would be nice to have, because the more often the dashboard will be refreshed, the more support (and cost) it will require after rollout.

Identify all filters and selections. The earlier in the project's life you can do this, the better. This information has a major influence on your dashboard design and will affect decisions about performance and capacity. If a large combination of multi-select filters can be selected for one component, there will be a multitude of data combinations to validate and possibly many thousands of rows to be stored. Technologists can be tempted to impress their business partners, but be careful not to promise something that is not scalable or sustainable.

Understand what levels of aggregation and detail are required. An early requirements exercise should involve the filters and dimensions that will be used as well as how they should be aggregated. Time periods are a common dimension as are office or geographical territories. On the flip side, sophisticated business users will inevitably want to know the details behind what is driving their trend or that one outlier metric. Not having a method of either drilling down to (or otherwise easily accessing) the detail behind the aggregation will frustrate users after the post-implementation honeymoon period has ended. Determining aggregation/detail needs should be part of the discussions during requirements gathering, but remember to balance your requirements with development difficulty and desired timelines. If detailed data is provided, it should be accessed directly via the dashboard, whether through sub-reports or drill-down capabilities in the components themselves, depending on your tool set.

Identify how much history you need. Some graphical trends will require year-over-year comparisons. Beyond that, it may be worth considering how long any data that no longer appears on the dashboard should be retained. If it does need to be retained for compliance or other purposes, an archival strategy should be considered, or possibly a view built on top of the base data. The more the dashboard can be limited to querying only the data it needs to display, the better it will perform.

Define the data testing and validation process. It is never too early to address how you will ensure data quality through a validation process. Defining specific responsibilities and expectations, and what methods will be used for validation, should happen even before design. This will also ensure that business users will be ready when they are asked to begin testing. The validity of the data is the most critical factor in the dashboard's success and continued use.

Integrate business users. There are several ways to involve business users in requirements gathering and refinement besides letting them dictate while you take notes. These options include:

- Prototype early and often. Prototyping can start with simple whiteboard exercises, and many dashboard tools now lend themselves to quick prototyping so business users can see and play with something similar to the final product deliverable. This hands-on method is excellent for rooting out requirements gaps, although it should not replace formal documentation.

- Use real data wherever possible when prototyping to give business users a better context. It also helps you to identify and correct data issues early.

- Integrate developers. Requirements gathering should not be done solely by analysts. If there are separate individuals responsible for coding, they must be involved at this stage so they truly understand the value and meaning of what they will build.

- Set expectations for production support. Agree upon a process for communication of user questions or any defects users discover. Depending on the user, this can be done many ways, although users at the executive level will likely prefer a direct communication path with the team's manager(s). Additional suggestions appear in the post-implementation section later in this article.

- Define milestone deliverables. Regardless of the software development methodology you use, defining milestone deliverables is critical for instilling and retaining business confidence in the project. It is also necessary to ensure the development team is progressing as expected.

An early requirements exercise should involve the filters and dimensions that will be used as well as how they should be aggregated.

Milestone due dates should be communicated early and deadlines met. If a deadline is at risk of being missed, share this information (as well as the reasons for the problem and the recommended course of action) with business users so new dates and deadlines can be agreed upon or so the team can remove items from the project scope or adjust resource levels and assignments.

Required deliverables from the business requirements-gathering phase may include:

- Scope lockdown, with documentation of what is in scope and out of scope.
- Final prototype with business sign-off. (Note: This remains a working prototype, and all team members must understand and agree that the design may change later in the project if practical.) The highest-level sponsor of the project should be part of this sign-off, as well as further sign-offs of the actual product prior to rollout.
- Detailed requirements definitions, including images from the prototype. Such documents can tie the business definitions of the metrics to the way they will be displayed. Such a connection will bring clarity both to the business client and to the developers/analysts building the solution.
- Technical conceptual design. This high-level document defines all data sources and targets, what delivery mechanisms are being used, and the general business use case(s) for the dashboard.

Defining milestone deliverables is critical for instilling and retaining business confidence in the project.

Designing and Building the Dashboard: Soup to Nuts

When dashboard design has begun, all layers should be considered in relation to one other. For example, if the dashboard will be connected to aggregated tables designed for performance, those tables, the way they are loaded (or otherwise populated), and any performance and capacity concerns should be considered. This is just as important as designing the dashboard functionality.

In general, the dashboard design should:

- **Ensure a single, consistent view of the data.** This can apply to the visual look and feel as well as how often the components on a screen are refreshed. The user should not have to think about how to interpret the dashboard; the data presentation should be clear and intuitive.
- **Keep everything in one place.** If detailed data or supplemental reports are needed, use the dashboard like a portal or ensure a centralized interface keeps the data logically consolidated. Also, make sure the same data source is used for both detailed and aggregated data on the dashboard.

  Keep in mind, however, that business users may expect that a snapshot of the dashboard will not change. For example, a monthly metric could possibly vary slightly in the source data, but re-querying every time for the dashboard view with different results could erode confidence and even skew expected trends. Have a conversation with business users early on to discuss such scenarios and determine whether storing point-in-time dashboard snapshots will be required.
- **Understand the usage scenario.** Knowing the size of the user base, as well as the types of users and when they will be accessing the dashboard, can drive design. You should understand the usage volumetrics early in your project and plan accordingly. You must also ensure that any maintenance windows do not conflict with peak-time use. Environment sizing, capacity, and performance will all be critical to ensure a stable tool.
- **Address multiple environments for development.** If your environment has the necessary capacity, build development, test, and production environments. It's worth it.

- **Plan to validate data accuracy as early as possible, and ensure your design and project plan allow this.** To avoid rework, it is crucial to make every effort to get the data perfect and acquire sign-off in a lower database environment during user testing. This will ensure that the data acquisition process is free of bugs. At the same time, ensure that you validate using data from the production source system(s), because the data will be well defined and likely have an independent method of validation.

- **Roll out with historical data available.** Plan on migrating all validated data to production tables along with the rest of the code. Implementing a dashboard with predefined history and trends will ensure a great first impression and enhance user confidence.

In addition to these areas of focus, consider several design best practices for both database/data quality and dashboard interfaces.

Database-Level Best Practices

Ideally, your dashboard will be running in a stable database environment. This environment may be managed by your team or may be the responsibility of another area of your company. Either way, your dashboard is meant to provide data for quick and meaningful analysis, so treating the data and the tables in which it resides is critical. Some best practices include:

- Using ETL or other data acquisition methods to regularly write to a highly aggregated, denormalized table. This will ensure optimal performance, as dashboard click events need to be fast. A good goal is to ensure that no click event takes more than three seconds to return data to the dashboard.

- Use predefined and well-maintained dimensional tables wherever possible. This ensures consistency and eliminates redundant data structures.

- Store the data using IDs, and reference static code or dimensional tables wherever possible. This way, if a business rule changes, only one table must be modified, and no data has been persisted to a table that is now outdated.

- Design and model the data so the front end can dynamically handle any business changes at the source level. This eliminates the need to update the code every time business users make a change, and maintenance costs will be much lower. The development team will then be able to work on exciting new projects rather than just keeping the lights on.

- Detailed data should be kept separate and not reloaded anywhere, if possible. However, it should be available in the same database so the aggregate and related detail can easily coexist.

- Unless absolutely necessary, do not store calculated values or any data that is prone to business rule changes. If persisted data becomes incorrect, it can be a huge effort to re-state it. Calculated fields can be done quickly using front-end queries or formulas (if designed properly).

- Create a data archival strategy based on business needs for data retention and how much history the dashboard needs to show.

- Ensure that any queries from the dashboard to the tables are well-tuned and that they will continue to run quickly over time.

- Likewise, ensure that any middle-tier environment used for running the dashboard queries is highly stable and can take advantage of any caching opportunities to enhance performance.

Dashboard-Level Best Practices

Spending a great deal of time on getting the dashboard data modeled, stored, automated, and correct will, of course, all be for naught if the dashboard front end is not intuitive, does not perform, or otherwise does not have high availability. To address this, take these steps throughout the life cycle:

- Check the dashboard usability by bringing in end users who were not involved in the initial project. Observe how quickly and easily they can meet their objectives, and remove all bias as you watch. You will need to plan for their participation well in advance, and this work should be done early in your testing (make sure to have production data at this point) so there is time to react to their input.

- Within the dashboard code, implement dynamic server configuration so all dashboard components can automatically reference the proper environment for the database, middle tier, and front end itself. This reduces reliance on hard-coded server names and can prevent deployments from accidentally pointing to the wrong location.

- Users will want to use Excel regardless of how well-designed your dashboard is. Make sure an Excel export option is available for all the data shown on the dashboard and any included reports.

- For every dashboard component, include a label referencing the specific data source as well as the data refresh date. This simple step resolves confusion and will greatly reduce the number of support questions you receive post-rollout.

- Do everything possible to avoid hard-coding filters, axes, or any other front-end components that change based on predictably changing business. The data and the front end both need to be flexible and dynamic enough to display information based on a changing business. The dashboard should not have to display invalid or stale data for a time period while the development team scrambles to implement a production fix. That would inevitably lead to a drop in user adoption and reduced confidence in the dashboard's validity.

- Test plans should include scripts for testing the overall dashboard load time as well as specific load times for all click events. This will afford the time needed to tweak code for optimal performance.

- Near the end of testing, simulate a performance load test whether you have automated tools to do this or you have to do it manually with multiple users. The purpose is to ensure no part of the underlying infrastructure has an issue with load.

- Test boundary conditions to avoid unforeseen defects later in the project's life. For example, what happens when a multi-year trend goes into a new year? Will the *x*-axis continue to sort properly? Define all conditions like this and find a way to test each one.

Do not store calculated values or any data that is prone to business rule changes.

Running the Project (and Subsequent Projects)

Considering the myriad of complexities involved in implementing a dashboard, from ensuring correct data is available when expected, to designing a usable and innovative front end, to working with the business through multiple and complex requirements, costs can be high and timelines can easily be missed if the project is not carefully managed.

The following procedures will help ensure a successful dashboard release, all in the context of the best practices explained so far:

Create and use an estimating model. The model should cover all aspects of a dashboard release (from data to user interface), all the technical roles and resources that will be involved, and be sufficiently detailed to break down time in hours by both phase and resource type. A model that can be defined by selecting answers to requirements-based questions will be the easiest for your analysts to use, such as: How many metrics and components will be displayed? How many data sources will be used? Does data for validation exist?

The model should be refined after each large project by entering the answers to these questions and determining how closely the model's hours match those actually spent.

Data validation is your top priority. Plan and allocate the time with your business partners and understand what data sources you will use for validation. If there is no independent source, you and your business users must reach an agreement about how validation will be performed.

Share real data as soon as it becomes available and the team has reasonable confidence in its accuracy. There is no reason to wait to share data, regardless of how early in the process this occurs, because the earlier data defects are identified and resolved, the more smoothly the subsequent processes will go.

As we've mentioned, we recommend you implement your project with historical data loaded. If this is planned, ensure that business users are aware and secure their pledge to spend adequate time comparing and validating the historical data.

Define phases of work and identify key deliverables for each. Regardless of the development methodology your department uses, you must align milestones to specific dates to ensure the project does not get out of control and to keep business users confident in your progress.

Depending on your business client and their expectations, you may need to blend agile and waterfall methods. Although this will not satisfy ardent practitioners of the methodologies, a blended approach can allow for the iterative testing and discovery that this type of work requires while ensuring adherence to a strict timeline, which a release of this complexity also requires.

Implementations are complex, so make a detailed plan. The manager or lead of the project should define all the steps needed, assign dates and responsible parties, and build a T-minus document/project scorecard. These tasks should be completed during the initial stages of the work, soon after any intake approval and/or slotting, and the document should be reviewed with the entire team at least once a week to ensure the project is consistently on track.

Depending on your business client and their expectations, you may need to blend agile and waterfall methods.

Escalate all identified issues and risks early and often. If your department already has a process for bringing issues and risks into the open and to the attention of those who can mitigate them, use it. Otherwise, create your own process for the project. Enlist all stakeholders and technology leaders for support, and do this proactively.

Review, review, review. Plan multiple design and code reviews, and assume at least a draft and final review will be needed for each major piece of work. Devote ample time to design review, because waiting until the dashboard is built may make recovery impossible if a fundamental design flaw has gone unnoticed. Formalize a method for tracking and implementing all changes identified during reviews.

Keep the development team engaged. For example, if the development team includes offshore resources, record key meetings using Webinar technology. This can serve as both knowledge transfer and training material later. Make sure everyone knows about the recording and ensure that no legal or compliance issues will arise.

Even though your work may be completed in phases, dashboards can rarely be efficiently delivered if a "factory" approach is used (in which requirements are passed to designers, and designs passed to builders, without everyone being involved). When a developer is far removed from business users working on a dashboard project, this can lead to project failure.

Create an internal process for ticket and defect handling, and implement bug fixes in small, bundled releases.

Implement a formal user-acceptance testing process. Once the development team has completed all internal testing of data and functionality, plan time (we recommend two to three weeks) to allow the business team to complete their tests. Testing should include as much data validation as possible. We recommend you formally kick off the testing phase with business users and employ a documented process for submitting defects and questions to the development team. Make this easy for your business partners. They should focus on testing, not on how to submit their test results.

Require sponsor/stakeholder approval before rollout. This will give your dashboard legitimacy to the ultimate end users and is invaluable for those early weeks when adoption may hang in the balance. This approval should include a presentation during which the sponsor can view and provide feedback about the dashboard, with sufficient time allotted to make adjustments. As mentioned, we recommend you conduct sponsor reviews of the dashboard throughout the project, including during prototype design.

Post-Implementation (You're Never Really Done)

After the dashboard is implemented, team members are often tempted to relax. There may also be new projects demanding focus.

Do not become distracted or complacent, because there are certain post-implementation steps that will ensure both that the few critical months after rollout go smoothly and that the development team does not become bogged down by production support or answering business questions.

First, build post-implementation work into the initial plan. Sustainability and support should be factors in scope and technical design.

For larger rollouts, consider best practices for the sponsoring business group and the technology team to handle presentations. This way, both business and technical questions can be answered accurately, all key partners are included, and accountability is shared.

Post-rollout sponsorship and change navigation coordination are crucial. The business unit will likely be responsible for communications and training, but the technology team can and should influence this.

If possible, ensure you have a method to collect usage metrics. If you can identify usage by user ID, that is even better because delineation between business and technology usage can be made and groups can be identified for training if usage is lower than expected.

The development team can suggest and implement innovative ways to communicate with users:

- Add a scrolling marquee to the dashboard or use some other technique for instantly communicating important messages. This component should be database driven, and the technical support team should have write access into a table separate from the dashboard's main tables. This way, announcements such as planned downtime or key data load dates can be easily delivered to all users.

- Add an e-mail button that goes directly to the dashboard support team. This may not be a popular choice for all technology teams, but dashboards are often used by upper-level managers who have no desire to call in tickets through a service center. They prefer direct and immediate access to the group that can resolve a problem.

- Create an internal process for ticket and defect handling, and implement bug fixes in small, bundled releases. Communicate the fixes to all users.

- Build context-sensitive help directly into the dashboard. Dynamically displayed help can greatly increase usability as well as cut down on support questions. Help text should discuss how to use the dashboard components, but should primarily emphasize the business rules and definitions of the metrics themselves. The dashboard should be built intuitively enough so users are not confused about its use. A document attachment is a viable alternative to context-sensitive help depending on the user base and what would best suit them.

Continuously funnel large and small enhancements into subsequent releases to maintain momentum. No matter how well you succeed with the initial deployment, there will certainly be areas for improvement.

Be open to user suggestions, even regarding component design or more traditionally technical items. Business end users are often great innovators you can learn from, and with advancements in Excel and business intelligence tools, they are becoming more technically skilled.

Learn from each deployment, and continue improving and documenting your own best practices.

Summary

Dashboards can be incredibly useful tools for business users, offering at-a-glance indicators of key company measures, the ability to view trends over time, and filtering capabilities to drill down to areas that could impact key business objectives. Dashboards can offer great visual flair *and* a quick method to identify and understand a positive or negative business impact.

However, implementing dashboards is not as magical as a flashy user interface might make it seem. The accuracy of the data, timeliness of its updates, and performance of user interactions are all as important as the visual design. To successfully implement all of these things, careful planning is required, as is a strong partnership between business sponsors and the entire development team.

The dashboard should be easy to use but can be difficult to develop. By utilizing the methods we've discussed, you can improve both the process and results of dashboard development at your enterprise. ■