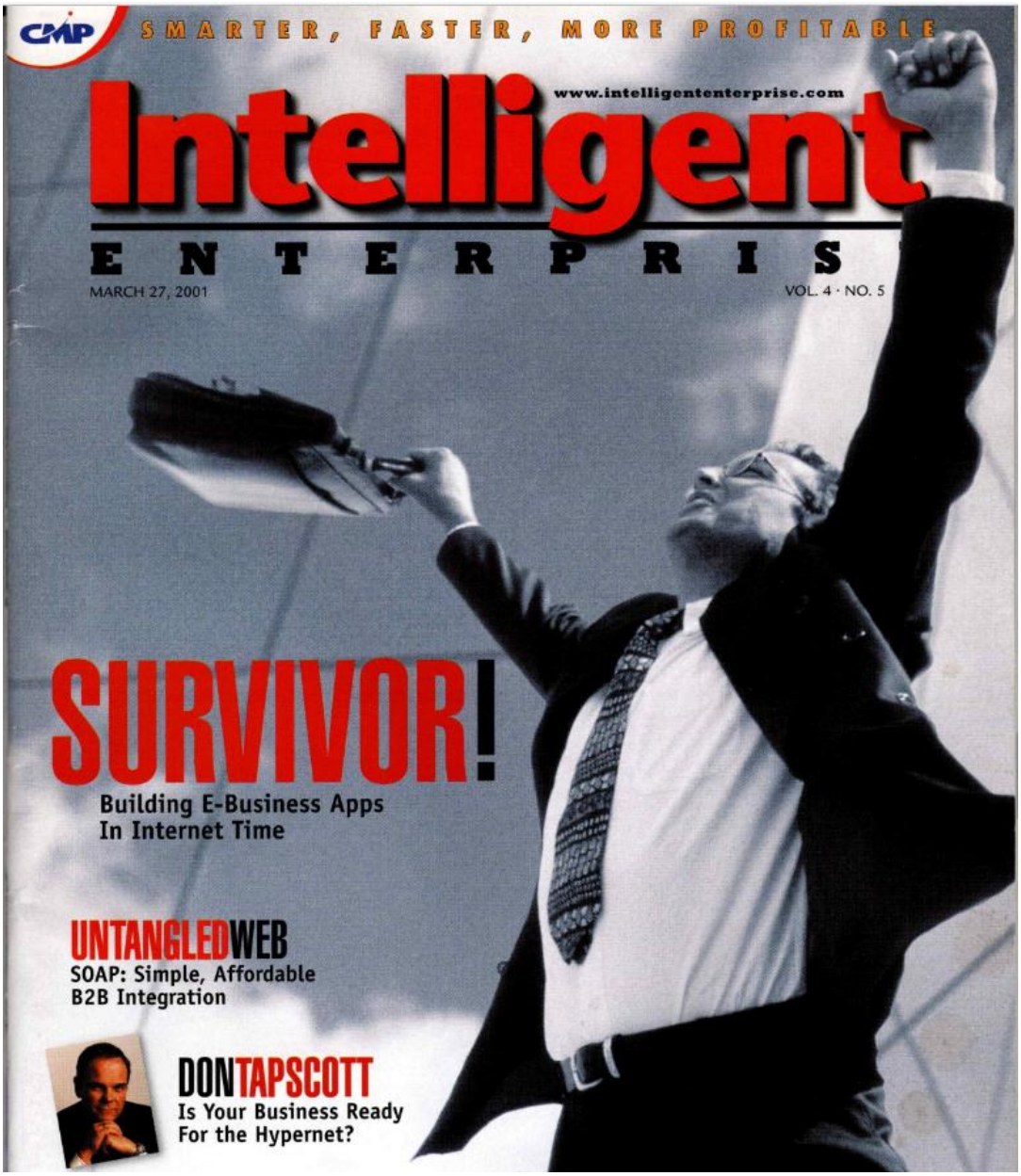
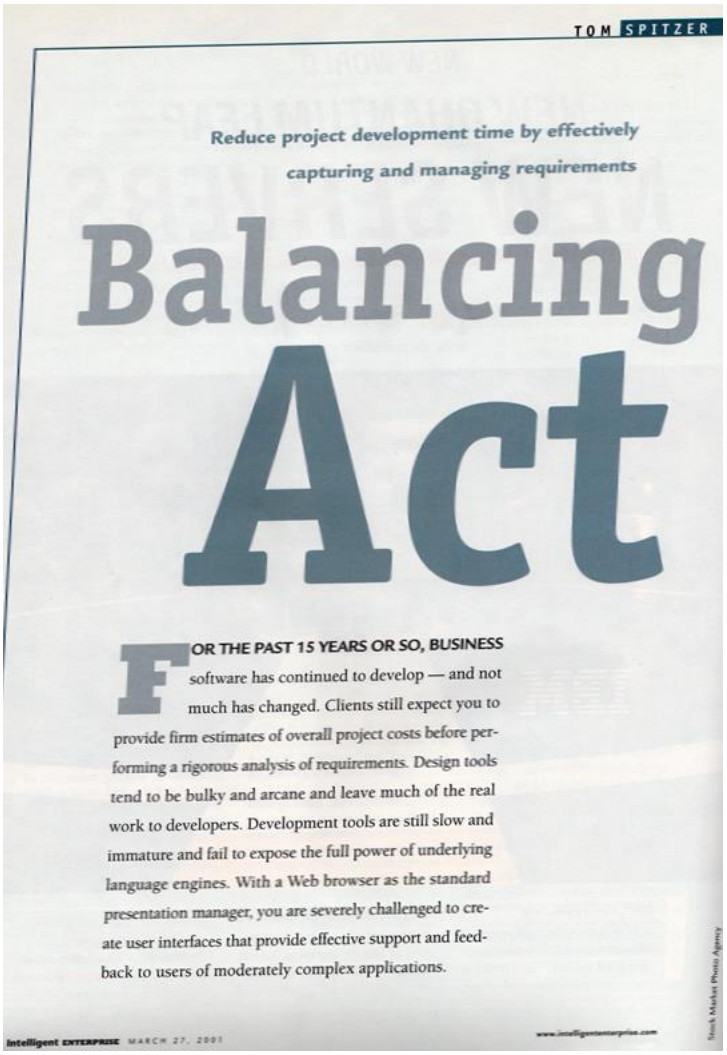
This article was the cover story in the March 27, 2001 issue of Intelligent Enterprise magazine

It is now December of 2024, and we can use Generative AI tools to write code. I think that it is effective requirements capture is all the more important, and less amenable than coding to the development of useful software applications.





TOM SPITZER

Reduce project development time by effectively capturing and managing requirements

Balancing Act

FOR THE PAST 15 YEARS OR SO, BUSINESS software has continued to develop — and not much has changed. Clients still expect you to provide firm estimates of overall project costs before performing a rigorous analysis of requirements. Design tools tend to be bulky and arcane and leave much of the real work to developers. Development tools are still slow and immature and fail to expose the full power of underlying language engines. With a Web browser as the standard presentation manager, you are severely challenged to create user interfaces that provide effective support and feedback to users of moderately complex applications.

intelligent ENTERPRISE MARCH 27, 2001

For the past fifteen years or so, we have been developing business software. Not much has changed. Clients still expect that you will provide firm estimates of overall project costs before you perform a rigorous analysis of requirements. Design tools tend to be bulky and arcane and leave much of the real work to developers. Development tools are still slow and immature and fail to expose the full power of underlying language engines. With a Web browser as the standard presentation manager, we are severely challenged to create user interfaces that provide effective support and feedback to users of moderately complex applications.

Among the most significant changes are an emphasis on short delivery cycles and the expectation of extremely large user populations brought about by near universal access over the Web. These considerations are especially critical when developing software that will be delivered as a service through an Application Service Provider (ASP). In particular, the scalability requirement has driven an increase in the complexity of the architectural foundation on which we are building applications whose functionality is actually relatively straightforward. As I will argue below, effectively capturing and managing requirements is the key to reducing cycle times. The right approach to software architecture maps software

components back to the requirements, and reduces testing to an integration problem. Designing an architecture that cleanly partitions component functions, and leverages standard models enables scalability and facilitates sharing processes and information.

While traditional enterprise software vendors are relatively familiar with this territory, heading in the ASP direction is a significant challenge for smaller application software vendors that serve the middle market. In the past, these were smaller companies whose products could exist as islands, and did not have to support extensive user populations. It was relatively easy to serve 5,000 businesses that had an average of 250 employees when you could require each of them to install a separate local instance of a client-server product. As a service provider that vendor is looking at a million user system!

Team Dynamics

Taken together, these changes have impacted the composition of project teams, the ways in which team members have to interact, and the structure of the projects on which they are collaborating. To produce Web based software of even moderate complexity requires a broad array of experts. Recruiting people with experience in larger corporate and consulting organizations has become extremely important to developers of ASP services. Today's effective project team needs at least the following members:

- individuals with strong database design and administration skills (not necessarily the same person!),
- somebody capable of selecting an appropriate technology platform,
- a requirements team capable of building a concise but comprehensive catalog of use cases that need to be supported,
- a usability expert capable of building information models that expose the product's functionality in a meaningful way to diverse user populations,
- developers capable of writing database stored procedures, middle tier business objects, and presentation code that includes or resolves to HTML, and
- an infrastructure expert who knows how to distribute all of these components in a manner that will maximize performance, scalability, and reliability.

It's no longer possible for one or two high performance individuals to build a product themselves. In fact, the market for those of us who made a living as knowledgeable generalists has tightened considerably!

Project Planning in Internet Time

When an organization launches a software project, the first thing that it wants is a schedule. The first thing that project teams want is a clear set of requirements. For them, starting a project by working on a project plan is extremely frustrating. If you don't know what you are supposed to build, its difficult to formulate a plan to build it. Conversely, management wants to know what "it" is going to cost and see a plan that shows "it" will be delivered on time, regardless of what "it" is! Emphasizing the time constraint heightens this tension, because without analysis nobody really knows if its possible to produce a meaningful set of functionality within the desired time frame.

In the "old" days, I was often able to structure engagements to include an analysis project and a subsequent development project. I could estimate the analysis project based on

number of users and a rough count of function points, and estimate the development project based on the results of the analysis. That approach no longer flies. With the possibility of infinite numbers of users, its impossible to interview a representative sample. Nor can we build a product in four months if we perform an exhaustive requirements analysis before starting any development. Instead, we have created a process in which we perform requirements analysis on a just-in-time basis, and have adopted an iterative project planning model to manage this process.

Take a look at a summary view of our project model displayed in Exhibit A. One of the significant departures from traditional project models is the extension of requirements analysis throughout the first half of the project cycle. In conjunction with analyzing requirements, we find it important to produce prototypes to serve as straw men for feature sets, and to provide a realistic simulation target for evaluating alternative deployment platforms. A second departure is the division of development work into between five and eight work packages, each scoped to run for about four weeks from design through unit test. Once we complete the Core Transactions work package scheduled for week 13, we have a product with sufficient functionality to support release to initial clients after another week or so of testing.

The Requirements Maze

Although its not news that effectively managing requirements is the key to managing schedules, its still a difficult thing to do. In the best case scenario, when you build a product or application without understanding the true requirements, your users or customers will let you know what's wrong or missing when you show it to them. In the worst case, they will abandon or ignore the product or service. If you survive, you have to add or change features in response to the new information, which will increase the cost and delay the effective release of the product. Slightly less damaging to the possibility of delivering on time is the tendency of the requirements to expand throughout the development process. As they expand, you need to invest time and effort into protecting the original scope or broadening the product.

When developing corporate applications, one can determine requirements by analyzing the corporation's business processes, and reviewing task requirements and business rules with prospective users and management representatives. To be successful, business software products (whether delivered as a product or a service) need to support all of the scenarios that a customer may encounter within a given application domain. Further, they need to provide sufficient flexibility to support practices and rules that can vary substantially from business to business. When gathering requirements for products, teams need to gather requirements from several potential customers, and then resolve inconsistencies between their practices and rules. As a result, I believe that specifying requirements for products is more difficult than for corporate applications.

Certainly the development team is going to be happiest and most productive if you create clear, comprehensive requirements before starting to design or build solutions. On the other hand, management gets extremely nervous watching the release date approach without much application code getting produced. Compressed within a four month project window, this creates an extremely stressful situation. We have been working with one of our clients, iTango Software, on tuning the development cycle to increase the comfort level of both parties. We are working to dovetail requirements gathering with the development schedule so

that we provide our development teams with the documentation they need, when they need it, but not sooner. Additionally, requirements analysts and designers are working closely to product requirements specifications in a format that feeds directly into the software design process.

Use Cases as a Requirements Vehicle

Use cases provide the most effective way to organize and prioritize requirements and deliver them to the development teams. Use cases provide a vehicle for establishing traceability between marketing requirements, design, development and testing. A "good" library of use cases clearly defines the various branches that may be followed through a task, and it clearly defines the relationships between the system under design and other systems it needs to interact with. Although it can be a sizeable effort, producing a good library of use cases ultimately enables faster completion of the product. Developers can take these use cases and quickly determine the responsibilities that each component must fulfill, and how each component must respond to other components. This is the essence of software design. Once the design is complete, available development tools do provide sufficient leverage to empower developers to build a great deal of functionality in a short time.

While use cases are a necessary part of the requirements specification, they are not sufficient. Use cases are ideal for capturing interaction driven functions, but its difficult to shoehorn articulation of business rules and requirements regarding performance, uptime, training costs, etc. into use cases. While we could do it, the results would be cumbersome. What we have done instead has been to create a catalog of business rules, and a catalog of requirements, and allow analysts to associate rules and requirements to the use cases that they support.

Since most people are unfamiliar with using a relatively structured approach to documenting requirements with use cases, rules, and requirements, it helps to provide a consistent structure for creating and accessing them. To do so, we created a database application with forms for entering these items and associating them with one another, and a set of queries and reports so analysts and designers can produce documentation on specific subsystems and functions.

On the first pass through the process, we aim for breadth rather than depth. Our objective is to determine the scope of the system by capturing many use cases quickly, but not fleshing them out with much detail. It took two people a couple of days to summarize the use cases for an employee benefits application that needed a supporting employee profile function that linked to several pre-existing human resource systems. We were able to turn this set of use cases over to our project manager to start building a project plan, and to the management team to make scoping decisions. A review of the use cases enabled us to assign "ABC" ratings of priorities to them. Next we added some meat to the "A" use cases and turned them over to the design and build team to start building object models.

Coding for Completion

Once the develop team has determined the responsibilities that each component must fulfill, and how each component must respond to other components, it can start writing code to support those responsibilities. Developing software with clear delineation of responsibilities between components, and limiting interactions between components to specified interfaces

provides a foundation for scalability. Adhering to standard architectural patterns that separate presentation from business logic and event management reinforces this effect.

Our development lead employs three strategies to produce a development schedule that fits within the overall time constraint, and that satisfies functional and non-functional (e.g. scalability and performance) requirements. First, he is ruthless about only building to support clearly documented requirements. If a requirement is not clear or comprehensive, he sends it back to the analyst for validation and clarification. Second, he enforces the organization of class code into highly focused methods, each of which fulfills a single documented responsibility. Third, he requires that all units of code include both internal test code and external testing procedures before they are considered complete.

Its critical that coding be completed within the construction phase, rather than allowing it to spill over into the testing process. I have been involved in (and admittedly I have managed) several development projects where a substantial portion of the development work actually took place during the testing process. Doing a substantial amount of coding in response to bug reports is a schedule killer, because its unplanned and unmanageable. Requiring that code deliveries include test harnesses extends the time necessary to produce a unit of code, but it's the only realistic way to plan and control the amount of time allocated for each unit on the project plan.

Specifying test cases as part of module design clarifies the intent of the class or module, and maps it back to requirements. Test cases flow naturally from use cases and supporting scenarios. Delivering test cases with the code that they test increases the confidence that the code supports the use cases. If developers write the test code before writing the class code, the test code will document what the class code will have to do, and will provide another opportunity to clarify any confusion about requirements. Since the test code focuses mainly on the external interface of the class, it will highlight required public methods and internal calculations. Another way in which self-testing code is also self-documenting is in that it provides other developers with examples of how to use the class.

Delivering code without internal and external test facilities simply defers the creation of unit tests to the test process, which is ill equipped to plan or manage this work. Further, it practically guarantees unplanned rework, since in the absence of verification its virtually certain that either a programmer or a tester will fail to fulfill or to understand a requirement. Producing code that must prove that it fulfills its design objectives ensures that development occurs when its supposed to, and that testing is not an adjunct to development, but a true verification of correctness and fulfillment of requirements.

Requirements Uber Alles

Although it may be counterintuitive, its more important than ever to understand requirements and adopt a structured approach to documenting and managing them. In fact, its worth considering making requirements gathering and validation a perpetual activity. Requirements change over time, now more quickly than ever. Businesses are buying and building systems intended to support relationship building between suppliers, customers, partners, and internal relationship managers. Companies continually analyze their markets, and can deploy marketing and business analysis resources in conjunction with one another to managing the evolving understanding of what services their customers, partners, and internal teams need to increase the value of their interactions.

Internet time means frequently extending the breadth and depth of the interactive services in which your organization participates. Whether you develop applications internally or you develop applications for others to use, the trick is to focus each release on a coherent package of services. To provide such focus, and provide an effective bridge between requirements analysis, design, development, and validation, try building a catalog of use cases to describe the requirements universe, and selecting the set of use cases to fulfill in each release.

Developing code that explicitly satisfies requirements articulated in specific use cases, and delivering that code with test cases that reflect those requirements is the best strategy for ensuring timely delivery of useful products and services. Organizing code modules around an architecture of loosely coupled components that fulfill clearly defined responsibilities yields two significant, but relatively unrelated benefits. First, it yields great flexibility in how it can be deployed, allowing the adoption of the deployment infrastructure that best fulfills scalability and reliability requirements. Second, it promotes ongoing evolution and adaptation through internal enhancements to existing modules, and addition of new modules that support services developed in response to ongoing requirements gathering efforts.

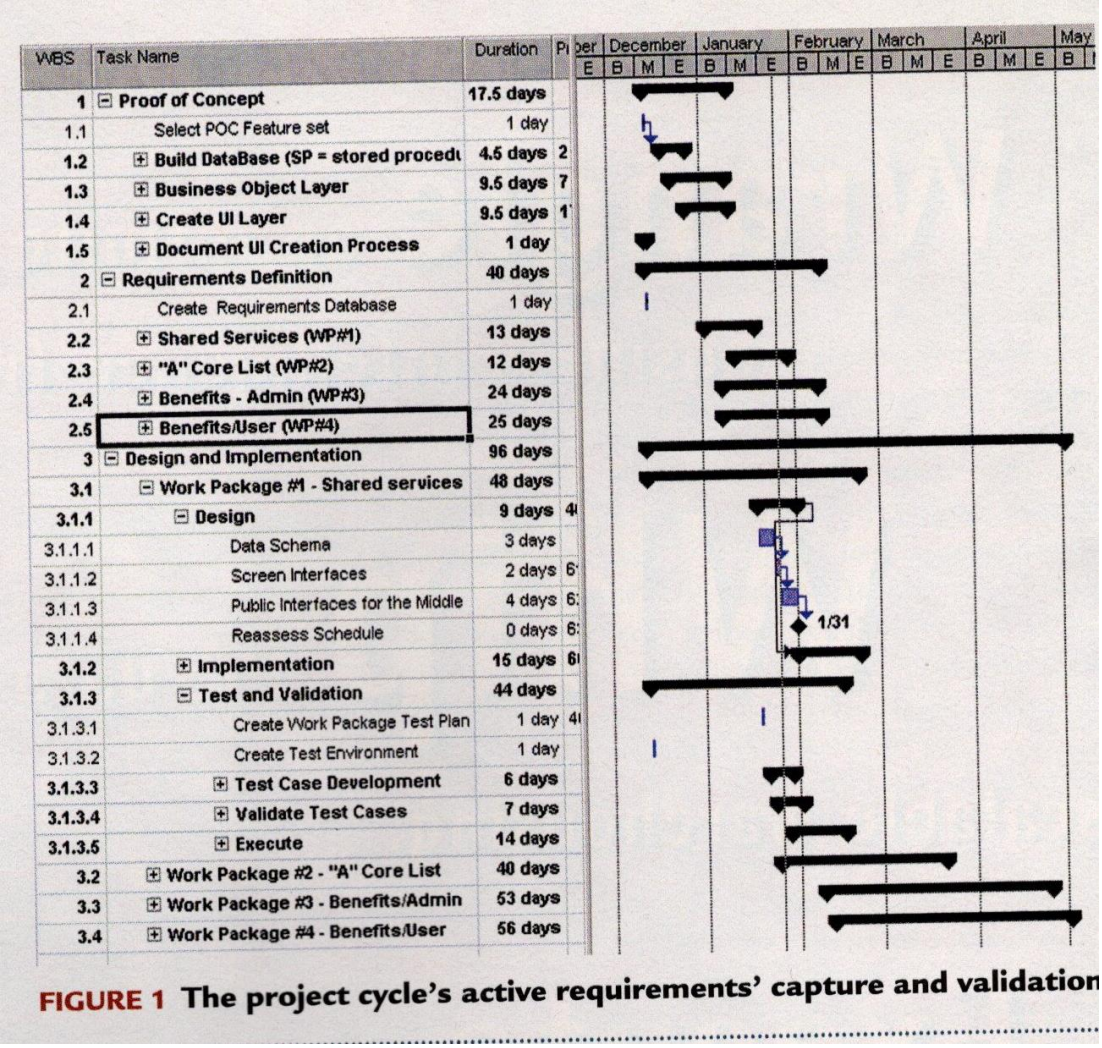


FIGURE 1 The project cycle's active requirements' capture and validation