WHITE PAPER

Best Practice with Structured Requirements

by Fergal McGovern, Product Manager, Optimal Trace Compuware Corporation

Audience:

- You are a business analyst and find it difficult to bridge the communications gap between business and IT users. This is causing misunderstandings about project scope and deliverables, often resulting in unmet expectations and, ultimately, project failure.
- You are a project manager and have recognized the need to adopt a more controlled and traceable approach to project development and scope management.
- You are an architect or QA lead and have recognized that your current process is sub-optimal. You may have already deployed a UML modeling solution, but it has proven difficult in terms of its deployment and the promised improvements have not materialized.
- You may have read some articles about requirements-driven software and the merits of the approach appear compelling, but you are still not quite sure about where to go from here.

Introduction

Analysts report poor requirements management accounts for as much as 71 percent of software project failures. The main cause is the gap between (a) what the business team wants and how it communicates this, and (b) what IT understands and delivers.

No matter how good a project development environment is, if the requirements captured in the first place are inaccurate or incomplete, then the project is destined for failure. The same fate awaits project plans that are structured around passive and unmeasurable module and task definitions rather than measurable, business-defined goals. It sounds obvious, but the current rate of project failure demonstrates that this is plainly a difficult task.

The main challenge with a software project is that the end result is, essentially, invisible. It is not like building a house, where design anomalies are often clearly visible. Communication in technical projects can be problematic, both between the customer/user and the business analyst and between the business analyst and the development/QA team. As a result, there is often inaccurate or poor understanding of the project scope amongst stakeholders. Project estimation relies on "gut feel" and there is often an overreliance on having particular technical people involved. Similarly, project progress cannot be measured easily or accurately due to inaccessible or overly technical milestones. This increased risk to the project often results in cost overruns, late delivery or, worst of all, outright failure to deliver the system required.

A structured approach to requirements capture and management resolves these problems and is the only way all stakeholders can be confident that all requirements have been understood and incorporated into the project plan.

Why traditional requirements techniques are not enough

The role of accurate and complete requirements in project success is not a secret. The problem is traditional requirement-gathering methodologies are failing. Why?

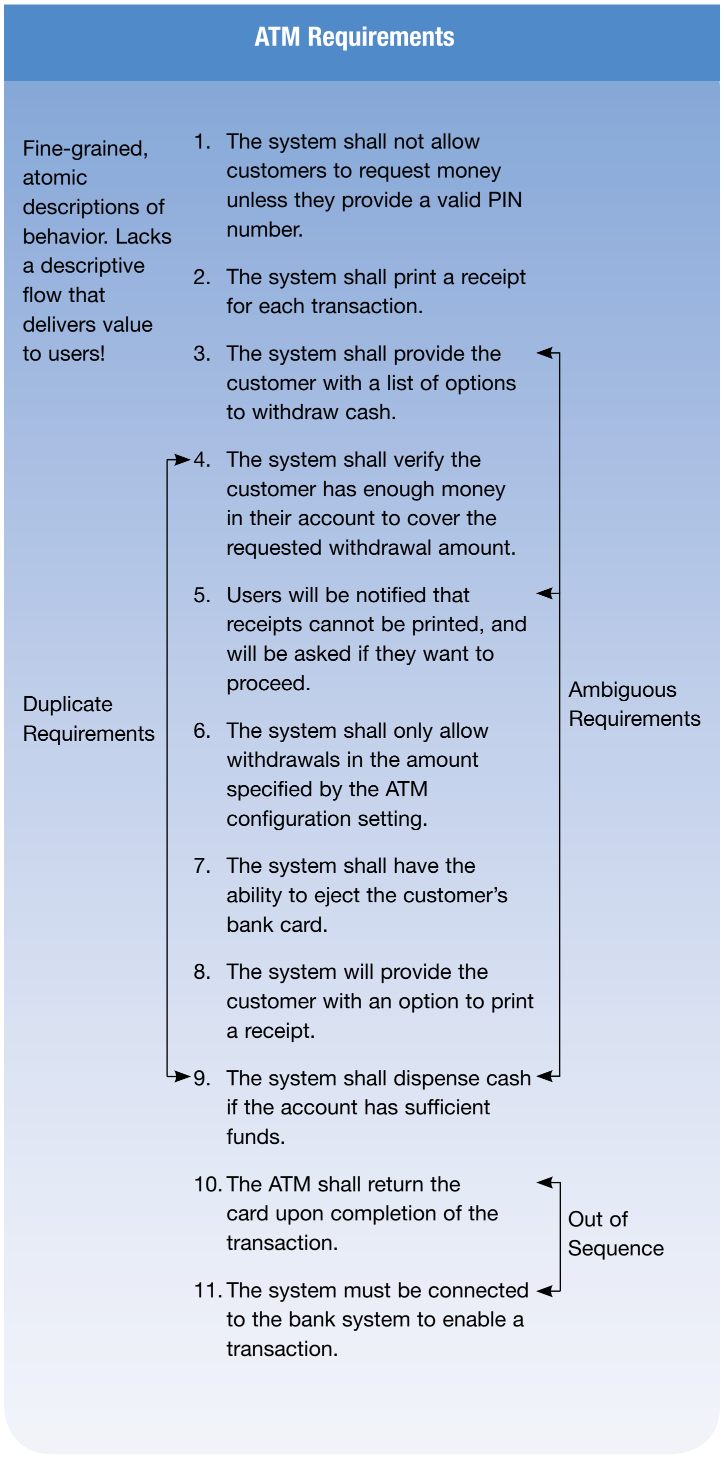
- They usually fail to describe what the system needs to do from a user's perspective, adding to the difficulty of communicating the project scope and deliverables. This is because they tend to focus on certain desired high-level end results, augmented with sometimes quite detailed and interspersed technical requirements.
- Traditional requirements also tend to be independent of one another. In other words, without being tied together by interdependencies and connections, the "story" of what the system does gets lost.
- They are hard to translate into designs. Since there is no story, developers have to guess what the system is really supposed to do.
- They are hard to test. Again, the absence of a complete "story" of how the system will be used means testers and QA have nothing to go on that will help them ensure the system will deliver as it should.

The example at right details traditional requirements for an ATM (Automated Teller Machine) project, demonstrating how traditional requirements gathering can create risk in a project.

The main problem here is the requirements are not sufficiently specific or business-oriented, incomplete or ordered in a way that makes identifying duplication or gaps difficult.

On the other hand, Structured Requirements bring a logical flow to the expression of the system's objectives, offering a step-by-step sequential account of everything that needs to happen in the system that all stakeholders can contribute to and understand. By their nature, they make it easier to identify gaps and interdependencies. Structured Requirements capture ensures a complete and accurate set of system needs is at the core of project planning.

The impact of a Structured Requirements approach on the same ATM example is shown here:



ATM Requirements

1. The system shall not allow customers to request money unless they provide a valid PIN number.
2. The system shall print a receipt for each transaction.
3. The system shall provide the customer with a list of options to withdraw cash.
4. The system shall verify the customer has enough money in their account to cover the requested withdrawal amount.
5. Users will be notified that receipts cannot be printed, and will be asked if they want to proceed.
6. The system shall only allow withdrawals in the amount specified by the ATM configuration setting.
7. The system shall have the ability to eject the customer's bank card.
8. The system will provide the customer with an option to print a receipt.
9. The system shall dispense cash if the account has sufficient funds.
10. The ATM shall return the card upon completion of the transaction.
11. The system must be connected to the bank system to enable a transaction.

Fine-grained, atomic descriptions of behavior. Lacks a descriptive flow that delivers value to users!
Duplicate Requirements
Ambiguous Requirements
Out of Sequence

Traditional requirements for an ATM project.

ATM Requirements

Preconditions

1. The bank customer must possess a **bank card**.
2. The network connection to the **bank system**, must be active.
3. The system must have at least some cash that can be dispensed.

Basic Flow

1. The Structured Requirement begins when the **customer** inserts a **bank card** into the card reader on the ATM.
2. The system reads the **bank card information** from the card.
3. Include requirement **Authenticate Customer** to authenticate the use of the **bank card** by the individual using the machine.
4. The system displays the **service options** that are currently available on the machine.
5. The customer selects to withdraw cash.
6. The system prompts for the amount to be withdrawn by displaying the list of **standard withdrawal amounts**.
7. The customer selects an amount to be withdrawn.
8. Perform "Sub-flow Assess Funds on Hand."
9. Perform "Sub-flow Conduct Withdrawal."
10. The system dispenses the requested amount to the customer.
11. The system ejects the customer's **bank card**.
12. The customer takes **bank card** from the machine.
13. The system records a **transaction log** entry for the withdrawal.

The Structured Requirement ends.

A Structured Requirements approach applied to the ATM project.

Structured Requirements ensure completeness, communicability and depth in a way that traditional or flat requirements cannot because they:

- describe how the system is used in a particular way by one or more users to achieve a desired result
- provide a complete "story"-like description of how the system should behave that is easy to understand by business stakeholders and members of the development team alike
- are expressed in plain language, not specialized notations, so that everyone can understand, offering a common way for all members of the project team to communicate about what the system must do
- express desired behavior, providing a more natural and complete way of describing what the users want to achieve and what the system needs to do in response.

Structured Requirements enable full traceability throughout the life cycle because they form the core of the project planning process, connecting the project plan with business objectives. In the process, they drive design specifications (flowcharts, UML models, etc.), user interface design (wire-frames, screenshots and storyboards), test planning (composed of test cases) and code modules (COBOL, Java, C++, etc.). Structured Requirements yield:

- a better channel of communication to the customer during analysis and build phases, for higher levels of customer satisfaction
- more accurate task and project estimation due to more complete and accurate requirements capture
- visibility into "high-risk" architectural issues earlier in the build phase, allowing you to take pre-emptive action
- a much improved ability to adjust for feature modification and reprioritization mid-cycle without jeopardizing the timeline. This commonly occurs in the event of a resourcing review (for example, through attrition or budget reallocations) or business reprioritization of features.

The net effect is a more controlled build cycle, reducing risk and cost.

Structured and non-functional requirements

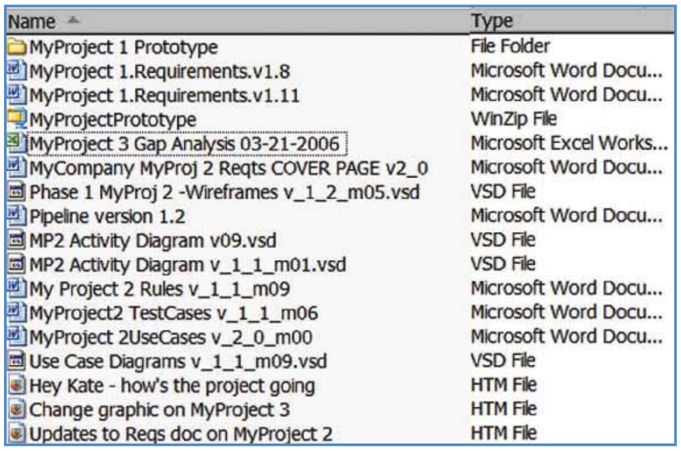
What exactly is a Structured Requirement?
Structured Requirements describe an objective or goal of a system. They are generated with the active involvement of the business user. Since they clearly reflect the behavior of the system in an understandable logical flow, it is easy for the user to understand and verify the process and ensure nothing is omitted. Structured Requirements also enable system architects and designers to understand the system objectives from the customer's point of view. They define the boundary and scope of the system, making it easy to determine what is in and what is out, accelerating customer buy-in and reducing disputes.

A Structured Requirement is a single "strand" of functionality that the system will deliver. When writing the specifications for a project, individual Structured Requirements will be formulated to satisfy each goal identified by the business users. These requirements will always be verb- or action-oriented and will strive to achieve a goal (Browse Orders, Purchase Goods, Login to System, Create/ Edit Orders, etc.). Therefore, a complete collection of Structured Requirements will express the entire functional behavior of a system.

Non-functional requirements
There are other system requirements that do not neatly fall into the Structured Requirements category. These are non-functional requirements (NFRs). These non-structured requirements generally fall into two categories: qualities, expressing aspects that the system must satisfy such as a number of concurrent users, security, etc.; and constraints, expressing "hard" realities such as the type of infrastructure the system must run on, coding language required, browser versions to be supported, etc. Typical NFRs include availability, performance, scalability, maintainability, manageability, security and flexibility. NFRs can either be associated with individual Structured Requirements or can be defined system-wide as part of the overall project.

What happens without Structured Requirements?

A conventionally facilitated session uses white boards and giant Post-it notes to describe a requirement. These are modified with strikethroughs and arrows pointing to repositioned steps posted on all walls of the meeting room. In this scenario, focus from requirement to requirement is achieved by referring to locations around the room—not the most streamlined way of



Name	Type
MyProject 1 Prototype	File Folder
MyProject 1.Requirements.v1.8	Microsoft Word Docu...
MyProject 1.Requirements.v1.11	Microsoft Word Docu...
MyProjectPrototype	WinZip File
MyProject 3 Gap Analysis 03-21-2006	Microsoft Excel Works...
MyCompany MyProj 2 Reqts COVER PAGE v2_0	Microsoft Word Docu...
Phase 1 MyProj 2 -Wireframes v_1_2_m05.vsd	VSD File
Pipeline version 1.2	Microsoft Word Docu...
MP2 Activity Diagram v09.vsd	VSD File
MP2 Activity Diagram v_1_1_m01.vsd	VSD File
My Project 2 Rules v_1_1_m09	Microsoft Word Docu...
MyProject2 TestCases v_1_1_m06	Microsoft Word Docu...
MyProject 2UseCases v_2_0_m00	Microsoft Word Docu...
Use Case Diagrams v_1_1_m09.vsd	VSD File
Hey Kate - how's the project going	HTM File
Change graphic on MyProject 3	HTM File
Updates to Reqs doc on MyProject 2	HTM File

Figure 1: Chaotic storage of project data.

managing requirements! Interdependencies between requirements are extremely difficult to manage and control in this scenario. Documentation is then synthesized based on a translation of scribbled notes, moderator recollection and refactoring of the Post-its. The business user is then expected to sign off on a document that is unfamiliar to them and often uses cryptic, technology-oriented notation.

In the absence of a Structured Requirements process, the management of requirements, and of the project as whole, becomes extremely difficult. For example, the graphic in Figure 1 shows a sample project repository containing details of a number of projects at "My Company." This is taken from a real-life example that shows how a poorly managed requirements-gathering process can make life difficult for everyone. Excel spreadsheets, graphics, Word documents and e-mails associated with different projects are mixed together with no consistent naming convention. Irregular spacing in the file names means the folder contents can't even be sorted based on project name. Equally, the e-mail subject line is the default file name, so it isn't clear to which project the e-mails necessarily refer. Tracking back through this mess to identify what has been agreed for each project is difficult enough. Identifying gaps, non-functional specifications and other basic requirements for the system in any complete way is made extremely complex. Even storing these artifacts in a versioning system will not ease the lack of transparency. Although it might support better management, this course of action will not provide any greater degree of connectedness or transparency into the quality/applicability of the requirements. This is requirements hell and, in truth, the whole project is hurting.

Advantages for project stakeholders

Business-based Structured Requirements as captured by Compuware Optimal Trace describe the functional, or behavioral, requirements of the system. These then feed into the design, development, testing, documentation and overall communication and delivery of the system. All stakeholders benefit: analysts, architects, user interface designers, developers, managers (project and product), testers, technical writers and, particularly, the business user.

Business users
Business users benefit by being included in the requirements-gathering process in an interactive manner. Non-technical language makes the process more understandable. Requirements are presented in step-based sequential tasks represented in a flow diagram that is complete, understandable and leaves no room for misinterpretation. Optimal Trace focuses the business user on the requirement at hand and can represent every requirement in the format most easily understood. The resulting documentation represents the exact content of input meetings, building trust and facilitating sign-off by stakeholders.

Business analysts
Structured Requirements provide analysts easy visualization of the steps involved in delivering on the business users' needs. By representing the system using natural language and graphics, in multiple document formats, Structured Requirements help all stakeholders feel confident their needs are being met. You can elicit multiple layers of abstraction and document them in an easy-to-understand format. You can explore system behavior by using simple, but deep, representations of the chain of events to be followed, ensuring completeness and early buy-in. This is a crucial factor in avoiding late-cycle changes. Similarly, adopting a Structured Requirements approach helps make modifications in a controlled way, ensuring all interdependencies are correctly managed. The business analyst can sort and prioritize project data based on metrics that make sense to each stakeholder, fast-tracking acceptance and validation. Business analysts can also use Structured Requirements to support collaborative working and enable stakeholders with a particular focus to isolate the streams of requirements of particular interest to them.

Project managers
Structured Requirements help project managers plan more effectively. You can group deliverables functionally, allowing delivery of full increments of functionality over time. These can optionally be delivered to the customer for verification and expectation setting, increasing their confidence the system will be delivered on time and in accordance with their expectations.

Project managers also benefit through enhanced scope management. You can easily add, delete or modify requirements and flows, understanding and managing the incremental impact of these changes throughout the life cycle. Requirements are structured naturally for scheduling, making it easy to map them to activities or tasks. Structured Requirements also let you baseline a given set of project data, making change control well-defined and transparent. Each set of Structured Requirements represents a phase in the consultation/requirements capture process. In the event that the customer decides to change direction during the project, then the earlier sets of Structured Requirements provide a verifiable "audit trail" of project variation, enabling quantification of the extent of change and identification of all amendments required as a consequence.

Architects
Architects benefit through the complete identification of all of the dependencies and alternate flows pertaining to the project, making it easy to visualize the solution and to deliver great code that works. All architectural elements or modifications included in the project can be clearly associated with (or traced to) a need to support a specific requirement or business goal. For example, a user function that retrieves content based on multiple data objects may need to be scaled to meet non-functional requirements such as sub-second response time for page retrieves. Without full knowledge of page elements and NFRs, such a design decision may be left undone. Architecturally significant or complex elements are incorporated early in the life cycle, enabling early resolution of architectural problems.

Structured Requirements form a natural mechanism for identifying initial architecture build-out. They give architects a useful mechanism for coordinating with the project manager about appropriate prioritization of high-risk aspects of a system as well as customer-critical functionality. Structured Requirements can map directly into the design and test environments, ensuring a seamless translation from business goals to requirements to development, test and delivery.

UI designers
Designers and those involved with user interfaces benefit by being able to easily represent the information required in screens, look-and-feel and any storyboards that form the dialog between the

system and the end user. Such artifacts can be associated with any relevant requirements so business users get an early impression of how the customer will experience the system.

In many cases, this ability to connect business needs, requirements and end-user experience avoids the need for time-consuming and problematic prototyping or simulation. (See associated Optimal Trace white paper "Show and Tell.") Generating project documentation in formats that are understandable to all stakeholders and include screenshots and other artifacts helps speed stakeholder buy-in and project validation.

Developers
Developers benefit since their implementation goals are directly molded from the stakeholders' expectations. Structured Requirements can be grouped collectively, mapped to interaction diagrams and easily refined into code. Ensuring every single piece of code developed is driven by an authentic business goal, eliminates the risk of scope creep and over-development. Similarly, linking the coding/development process to specific goals enhances the ability to reuse code (e.g., code for "Browse Orders" may well be reusable for similar projects in the future).

QA test leads
QA and test leads benefit from complete clarity on the business requirements driving the project. Structured Requirements describe exactly how the system needs to behave, giving the perfect initial baseline for test plan development and documentation (whether Pre-Conditions -> Environment state pre-test; Flow of Events -> description of inputs/outputs or Post Conditions -> Environment state post-test). As a result, QA and test personnel can improve quality and ensure the final product actually matches the original business goals. Associating elements like NFRs, custom properties, alternate scenarios, etc. directly into the test plan helps highlight suspect links as well as vital performance and operational requirements early in the life cycle. You can take action to ensure project completeness at the earliest possible stage.

Structured Requirements, therefore, offer definitive coverage of all requirements through to QA and test, ensuring testing and QA focus on the original business needs.

Technical writers
Technical writers benefit because Structured Requirements provide the scope of feature documentation requirements. This gives a natural organization for any documentation associated with the project, providing goal-based descriptions, system states for each requirement and user interactions described in text. The automatic generation of project documentation based on Structured Requirements, with Optimal Trace, eases the presentation and communication of system functionality from the start. Technical writing is easier, and documentation is more complete.

Summary

Structured Requirements, as enabled by Optimal Trace, bring a whole new level of accountability, efficiency and reliability to the management and communication of projects. This approach is vital for success in project delivery, fulfillment of customer expectations and successful management of resources and compliance.

Structured requirements throughout the life cycle

Analysis

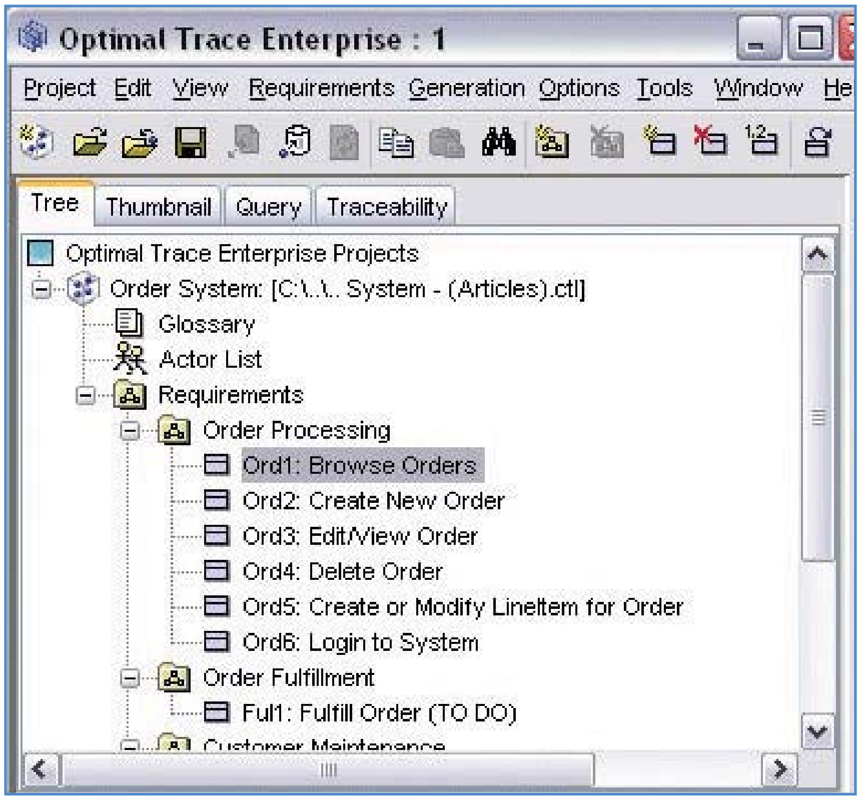
We know IT projects are usually driven by a business need: increase efficiency, support a sales channel, centralize a distributed business activity, etc. Since Structured Requirements focus on real, measurable business requirements emphasizing goals and objectives, they offer a real opportunity to link project delivery with the original business drivers of the project, addressing the single biggest cause of project failure in the process.

The truth is even with the best possible project development environment, the critical success factor will always be accuracy and completeness in capturing business requirements and goals and tracing them to the associated details. Let's take a look at how to get this right.

Engage with the customer or business sponsor early
It sounds obvious, but it is important to make sure the business sponsor or customer is involved at the outset in clearly stating the business goals for the project. This person needs to have a clear view of the ultimate objectives of the project and what a successful outcome is, especially from the user's perspective. They also need to make sure all business-related stakeholders are available to participate in the initial scoping and analysis phase.

Specify the main high-level goals clearly
Structured Requirements help the business stakeholders express their goals in plain language and in increasing detail during the course of the analysis. This clarifies the overall scope and depth of the project in a way that everyone can understand. It is vital the business stakeholders understand their roles in defining the project at this stage.

The business sponsor and his/her team work with the business analyst to identify the primary goals of the proposed system. Most project initiatives have very high-level business needs or drivers, for example, sell more product through a 24x7 delivery mechanism,


Optimal Trace Enterprise : 1
Project Edit View Requirements Generation Options Tools Window He
Tree Thumbnail Query Traceability

Optimal Trace Enterprise Projects
Order System: [C:\..\.. System - (Articles).ctl]
Glossary
Actor List
Requirements
Order Processing
Ord1: Browse Orders
Ord2: Create New Order
Ord3: Edit/View Order
Ord4: Delete Order
Ord5: Create or Modify LineItem for Order
Ord6: Login to System
Order Fulfillment
Ful1: Fulfill Order (TO DO)
Customer Maintenance

Figure 2:
Good quality business drivers and goals.

improve customer loyalty by increasing usability, speed delivery times for the invoicing system, etc. It is important to clarify these drivers before sign-off is agreed at the executive level. These primary drivers also are typically reflected as the "Project Vision" and tend not to change after project inception. They are, however, still too high-level to form a meaningful contract that can drive an application initiative once it begins.

Handing these kinds of drivers to a development team will clearly not work as they lack sufficient detail. Without being broken down into more specific goals and objectives, they can frequently lead to over-engineered frameworks capable of handling only very broad business scenarios. This often reflects a dynamic in the project team that is "technical-led." The lack of traceable detail in the requirements and the lack of visibility undermine confidence in meeting the specific business need.

Identifying and relating the primary drivers to a more detailed and specific level of goals is therefore critical.

It is this next level of goals that forms the basis, or "framework," for building out a full set of detailed and operational system requirements. This is where to focus.

What do such goals look like? Assuming the business has a driver to "Make the ordering process more efficient," we might have an associated set of goals:

- Browse orders
- Create new order
- Send invoice
- Setup promotion code
- Cancel/delete order
- Ship to customer
- and so on.

You can view these goals as "contractual" obligations that the system must fulfill from an end-user perspective. Indeed, looking at the proposed system from the viewpoint of the end user is an extremely effective way of arriving at the first set of appropriate goals. Each goal becomes a single Structured Requirement and, within Optimal Trace, these goals appear in the Tree area. (See Figure 2.)

Collaborating in stakeholder analysis sessions: To help elaborate what is needed during the analysis phase, you can use Optimal Trace on a laptop in a Joint Application Development session with business stakeholders or domain experts. This helps clarify from the outset the highest-level objectives that the system needs to satisfy, in other words, the business drivers and goals. In this context, developing a project in Optimal Trace and projecting the flows onto a screen is an effective way of establishing agreement amongst a team and working through any issues or clarifications.

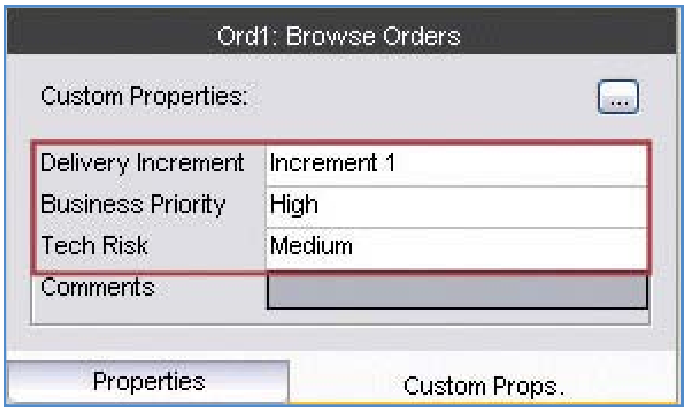
Ask yourself what the system must offer a user and this will quickly generate an initial set of Structured Requirements. If the system must be capable of sending an invoice to the customer, then we have a Structured Requirement called "Send Invoice."

What about systems that have no direct user interaction? Many system initiatives are infrastructure-oriented projects involving the need to "join up" or "integrate" existing back-office systems. Often, these initiatives are fulfilled by a service-oriented architecture providing value to third-party systems. Regardless of implementation architecture, the central tenet of the structured approach still holds true in this non-user-interactive context. For example, "Calculate State Tax for Inter-state Shipments" given with some parameters becomes a contractual obligation and may be satisfied in a service-oriented world by a relevant interface. It is critical, therefore, to understand the business rules and details of the Structured Requirement and to integrate them into the requirement itself.

Add detail to the prioritized Structured Requirements
Having identified the business drivers and their associated high-level goals, we now have the basis for building detailed Structured Requirements for the project. By taking each high-level goal and breaking it down into all of its associated flows, with each flow comprising a group of constituent steps, a set of complete Structured Requirements will emerge. In essence, each fully fleshed out Structured Requirement is a collection of related flows, each directed to achieve the specific goal or part of a goal.

Initially, you can focus on a few of the main goals, considering two things: business priority and technical difficulty. These factors tell us how to prioritize correctly.

Within Optimal Trace, you can stamp each Structured Requirement with the appropriate values for "business priority" and "technical risk." If working in a more agile world, you may also consider using the field "increment number" at the requirement level to specify the increment of delivery for that goal. (See Figure 3.)


Ord1: Browse Orders

Custom Properties:

Delivery Increment	Increment 1
Business Priority	High
Tech Risk	Medium
Comments	

Figure 3: Custom properties in Optimal Trace.

For example, assuming we have deemed “Creation and Editing of Orders” to be a high-priority, we might further flesh out that goal with a sequence of steps such as:

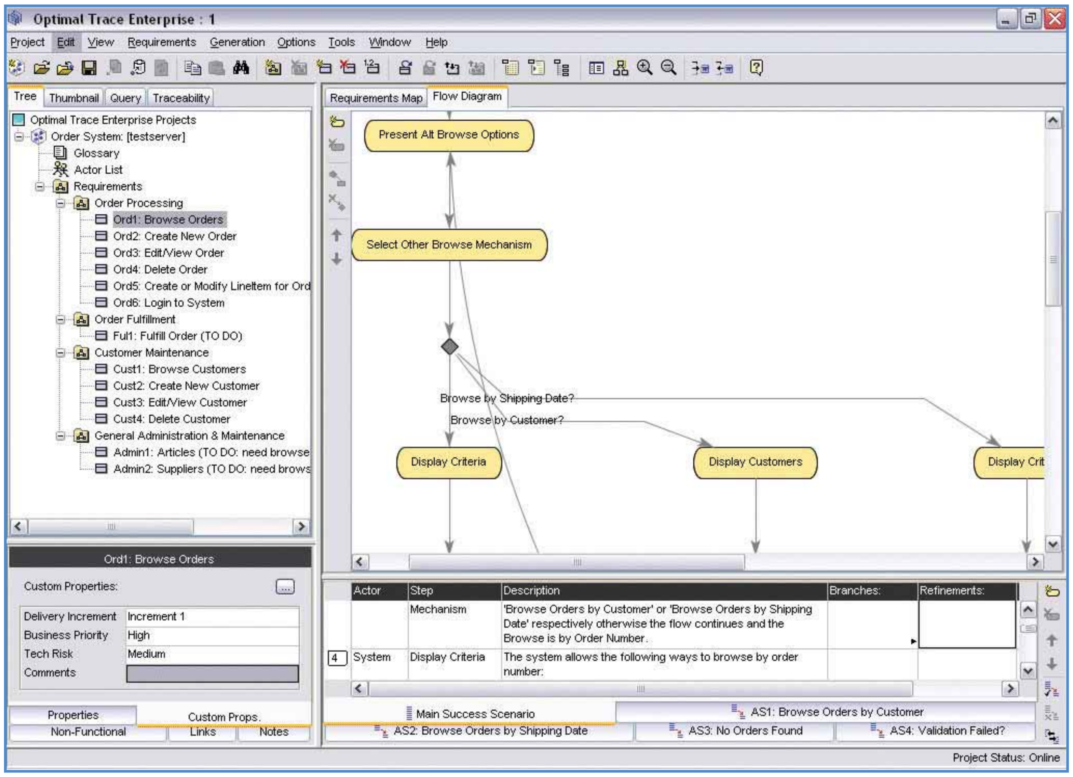
- Check if logged in
- Provide billing address
- Confirm product and quantity
- Calculate total cost including tax
- Validate information (quantity greater than zero, for instance)
- Provide payment instructions
- and so on.

In one sense, this set of steps constitutes a dialogue between a user and a system. Figure 4 reflects the dialogue represented in Optimal Trace.

Through direct association with the high-level system or business-user goals and expression in action-based language, you can easily communicate the Structured Requirements to the business and technical stakeholders. These are your first Structured Requirements.

You can see these early stages of system analysis are mission-critical. You have specified the initial high-level business goals and expanded each into a series of Structured Requirements. These in turn are fleshed out and broken down into sets of steps.

Applying this process to every high-level goal and drilling down in increasing detail through every strand of activity required by the system, you construct a complete and accurate picture of everything the system needs to do.


Optimal Trace Enterprise : 1
Project Edit View Requirements Generation Options Tools Window Help
Tree Thumbnail Query Traceability
Optimal Trace Enterprise Projects
Order System: [testserver]
Glossary
Actor List
Requirements
Order Processing
Ord1: Browse Orders
Ord2: Create New Order
Ord3: Edit/View Order
Ord4: Delete Order
Ord5: Create or Modify LineItem for Ord
Ord6: Login to System
Order Fulfillment
Ful1: Fulfill Order (TO DO)
Customer Maintenance
Cust1: Browse Customers
Cust2: Create New Customer
Cust3: Edit/View Customer
Cust4: Delete Customer
General Administration & Maintenance
Admin1: Articles (TO DO: need browse
Admin2: Suppliers (TO DO: need brows
Requirements Map Flow Diagram
Present Alt Browse Options
Select Other Browse Mechanism
Browse by Shipping Date?
Browse by Customer?
Display Criteria
Display Customers
Display Crit
Ord1: Browse Orders
Custom Properties:
Properties Custom Props.
Non-Functional Links Notes
Main Success Scenario
AS1: Browse Orders by Customer
AS2: Browse Orders by Shipping Date
AS3: No Orders Found
AS4: Validation Failed?
Project Status: Online

Delivery Increment	Increment 1
Business Priority	High
Tech Risk	Medium
Comments	

	Actor	Step	Description	Branches:	Refinements:
		Mechanism	'Browse Orders by Customer' or 'Browse Orders by Shipping Date' respectively otherwise the flow continues and the Browse is by Order Number.		
4	System	Display Criteria	The system allows the following ways to browse by order number:		

Figure 4:
Main project page in Optimal Trace.

Consider alternative flows
What about situations where things may not happen as planned in the main flow? Usually, for each goal, there may be a number of points where a deviation in the story can occur. These deviations may result in completion of the goal or failure to achieve it.

You need to be prepared for commonly overlooked scenarios like:

- What if the quantity requested is zero?
- What if the product is out of stock?
- What if the log file exceeds the disk size?

Considering alternative scenarios for different or unusual situations often serves as an early warning system for lack of completeness in the requirements. Some alternative scenarios may be business-oriented (e.g., “product out of stock”) while some may be more technical (e.g., “disk space exceeded”). Both types of scenarios should be fully considered, resulting in a completed contract for each goal.

Each alternative scenario or flow should be outlined and connected via a branch to the related step condition for the scenario. For example, if the quantity of items ordered is zero, branching from the step “Validate Information” might require the system to raise a query to Accounts highlighting an error in the records.

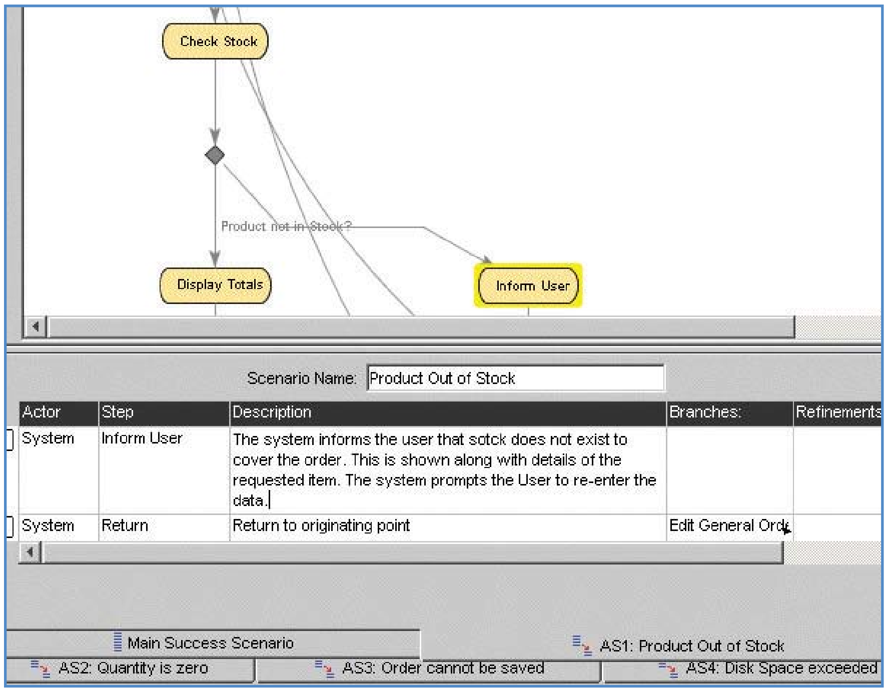
Figure 5 shows how it might look in Optimal Trace.

Note that alternate scenarios/flows may not end in success, since some scenarios may not meet the objective of the Structured Requirement. It is critical these are documented, as the system needs to take these into account. If they are not accounted for, scope can expand and project overruns can occur.

Address performance and quality characteristics
It is generally at this stage you should consider the NFRs. These relate to the qualities and constraints of the system and can relate to specific Structured Requirements or to the system as a whole. Issues such as availability, performance, scalability, maintainability, manageability, security, flexibility, etc. are managed through NFRs, and they are a vital part of the initial project scoping.

Ask business stakeholders questions relating to the overall performance of the system. What is the expected user load? What about speed of response when issuing the invoice?

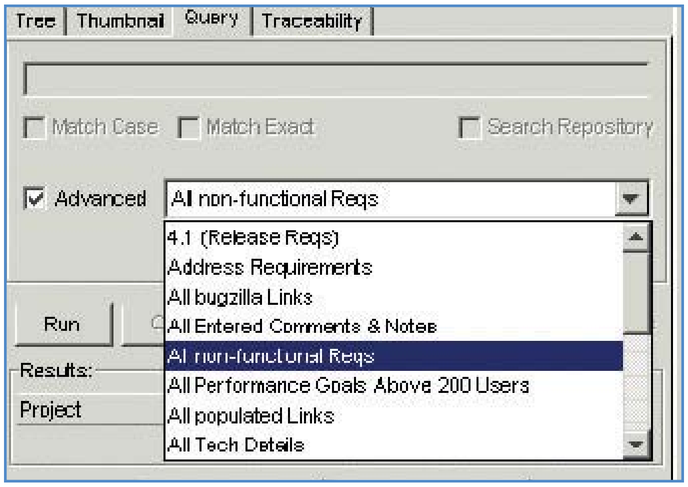
Such questions can be difficult to answer, especially for business-facing stakeholders, but leaving them unanswered introduces risk. If the business cannot say how many users are expected to simultaneously create orders, then we have a significant hole in our requirements project. Factors that may come into play during the lifespan of the system also need to be taken into account as much as possible. Not considering expected system qualities such as scaling projections at the outset can lead to high-risk and non-scalable systems. The bottom line is greater business risk.



Scenario Name: Product Out of Stock

Actor	Step	Description	Branches:	Refinements
System	Inform User	The system informs the user that sotck does not exist to cover the order. This is shown along with details of the requested item. The system prompts the User to re-enter the data.		
System	Return	Return to originating point	Edit General Ord	

Figure 5:
Alternate flows in
Optimal Trace.



☑ Advanced: All non-functional Reqs

- 4.1 (Release Reqs)
- Address Requirements
- All bugzilla Links
- All Entered Comments & Notes
- All non-functional Reqs
- All Performance Goals Above 200 Users
- All populated Links
- All Tech Details

Figure 6:
Categorized non-functional requirements in Optimal Trace.

In Optimal Trace, you can include the non-functional requirements associated with the structured requirement by highlighting the requirement in question and entering appropriately. If you wish, you can categorize the specific NFR as either performance, legal, etc. by using the "type" column. This is a standard column in the "Standard Software Development" template that ships with Optimal Trace.

You can then use a query to filter the project at any time for any performance requirements. If you are the system architect and receive no NFRs with your requirements, then it is vital to get clarity from the business analyst on what the business needs and all resulting system requirements. Visibility at this level is critical.

Get customer sign-off and baseline the project
At this stage, you need to get sign-off from the customer.

Generate the project documentation in Microsoft Word and create a project baseline. Baselines are useful as they:

- provide a clear account of what exactly was agreed upon with the customer
- give access to any changes that occur, whether they originate from the business or the technical side of the house.

The end result of this detailed work is a full set of project data that outlines all steps, branches, alternate flows, qualities and constraints. Screenshots and graphics can also be attached to the project. The result: Business stakeholders get a comprehensive account of the project that reflects the desired customer experience of the system mapped from the business goals. Optimal Trace can present this project data in multiple editable formats. Business stakeholders can see their requirements clearly reflected in the project plan and sign-off is easier and quicker.

Do's and don'ts of Structured Requirements

Do use verbs—goal-based

- Requirements are always active and verb-oriented.
- They are associated with a system goal.
- Write active requirements that have verbs. For example: "uc23: Personalize Content" rather than "uc23: Personalization."
- Thus the objective or goal of this requirement is to allow the user to personalize the content of the web page.

Do use definitive language—avoid fuzzy language

- Never use phrases such as: "the system might allow for," "xyz functionality may be provided," "we may consider providing," etc.
- Language must be definitive and unambiguous. If in doubt, place any ambiguous statements in the "open issues" area rather than in the requirement. Once resolved, it should be either in or out of scope and the requirements updated accordingly.

Do use structured narrative—keep the context visible, the value to the user clear.

- Compare flat paragraphs:
  "The order entry system has an interface to a back-end system and a terminal. It computes and displays the sum of the order items' cost..."
- With structured narrative:
  "The orderer (system or an entry person) identifies the name of the customer and the items on the order. The system displays the cost of the total order. If the items are in stock and the client has sufficient credit, ..."

In this example, you can see the flat paragraphs just give a listing of separate issues, with no real connectedness. The paragraphs don't represent a chain of events and certainly don't make it clear what exactly needs to be coded/tested/delivered. Alternatively, the structured narrative shows the individual steps that make up the deliverable. Each step is an element that can be coded and tested—and even delivered incrementally. This highlights the true benefit of Structured Requirements: the representation of an entire project in clear, discrete steps showing connections, dependencies and alternate scenarios. This clearly enhances the ability to successfully deliver a complete project.

Do express objectives in non-technical language

- Express requirements in a structured way that is understandable by all stakeholders.
- Requirements are best expressed in structured English, defined as specific steps or groups of steps within a process.
- For example, "build XML parsing infrastructure." This is understandable to your developers, but not to your customers.
- Whereas, "exchange purchase data with suppliers" makes sense to customers.

Do get specific on detail

- Specify detail—avoid late scoping surprises.
- Specify requirement detail in as far as is possible.
- For example, in the case of "Personalize Content," do not have the body of the requirement read: "The system offers the ability to allow the user to personalize her/his web pages." This is open-ended and will potentially lead to major scope creep.
- This also makes it difficult to estimate the amount of work required.
- Specify exactly what you understand to be the areas that can be personalized. For example, you might write: "The system offers the user the ability to customize the web pages based on: (a) Sporting interests, (b) Horoscopes or (c) Location."

Do avoid implementation details—no lock-in with architecture

- Technical detail belongs in the architecture and design documentation. Try not to allude to, or deal directly with, the technical detail of how the requirement will obtain its functionality.
- Never say how the system will achieve the requirement, always say what the system offers. This is trickier than it sounds.
- If you specify it here, that means you've locked yourself into a specific technical solution which subsequently may need to be adjusted/tweaked. The requirements form the scope portion of the contract. Any change at a later stage will require a change in contract.

Do specify alternative scenarios/flows

- Requirements with no alternative flows imply a perfect world. That's not likely in the real world. There are always exceptional situations.

Do avoid premature design and capturing design in Structured Requirements

- Each requirement yields tangible observable value to an actor.
- Don't over-abstract requirements.
- It is easy for developers/technical people when first writing requirements to over-analyze and attempt premature "reuse."

Common analysis questions
The following section outlines some questions that can arise during the analysis phase.

How big should the requirements be?
The following questions will help you decide how big or small your Structured Requirement should be:

- If you need to create a (functional/UAT) test case for this project, would you have to jump across many requirements to get a meaningful user-facing test case?
- If you are explaining the objective or business goal to a business stakeholder, do you have to jump across many requirements to explain the goal coherently?
- Do many or most of your requirements just have a main path only, with no alternative scenarios?
- Do you have a very large number of requirements, >100, for example?

If the answer is "yes" to one or more of these questions, then it is highly likely you need to combine the requirements and make a smaller number of core objectives/requirements.

What makes a good Structured Requirement?
The requirements associated with each goal must be measurable and understandable. In short, a good requirement is a step that provides value to a given user of the system. A second characteristic of a good requirement is that it is verb-driven and active in voice. Specifically the name should begin with a verb, for example, "Place Order." Another characteristic of a good requirement is it is not too granular and can stand alone. For example, when defining the requirements for an ATM, "Validate PIN" would not be considered a good requirement as it depends on other requirements to provide value to an external stakeholder. "Withdraw Cash" is a better requirement to set, and "Validate PIN" would be represented as a step within the requirement "Withdraw Cash."

For each goal, write a simple Structured Requirement, for example, what the goal delivers. The ideal outcome of the goal will follow from this. This is the easiest-understood aspect of the requirement, and other features represent refinements to this scenario. Then capture each user's intention and responsibility, from trigger to goal delivery. Identify and state what information passes between them and number each line. This results in a readable description of the system's function. You can also identify any qualities and constraints associated with the scenario (the non-functional requirements) and attach these to the scenario.

Structured Requirements Guidelines

- **Aim for approximately 30 externally facing requirements per project for manageability. Treat large projects as several internal projects.**
- **Each requirement should have a max of 10 or so scenarios for manageability. Break it out into sub-requirements if necessary.**
- **Each scenario should have at most 10 to 20 discrete steps.**
- **Estimate 2 to 4 man-weeks of pure development time per scenario in a normal external-facing requirement.**
- **Assume 1:1 ratio of development time to test time.**

What are packages and what is the best way to package Structured Requirements?
Within Optimal Trace, sets of Structured Requirements can be grouped into one or more packages. Packages can be contained in other packages and are typically used in a number of ways:

- Identifying delivery areas for project planning
- Identifying related business objectives associated with a given area of the system. For instance, if building a financials system, "Applications Receivables would likely be represented as a package."

How many scenarios and steps should I have for each Structured Requirement?
In general, for a customer-facing requirement such as "Create Trade," a main scenario will tend to have between five and 15 steps. Additionally, there will likely be anywhere from three to 10 alternative scenarios. Questions that prompt scenarios include:

- Are any data validation steps needed? If so, represent what happens when the validation fails as an alternative scenario.
- Are there different ways to achieve the goal or objective? For example, "Browse Trades" can be done by date or by trader, etc. These might be represented using alternative scenarios.

How do I identify gaps in scope and requirements?
Structured Requirements gathering involves breaking the project down into individual steps. Representing the entire project as a series of independent or connected steps with associated, alternate

scenarios and branches makes it significantly easier to assess whether everything has been included and to take any necessary action early in the life cycle.

Optimal Trace also ships with a pre-canned Complexity and Completeness Report that easily shows:

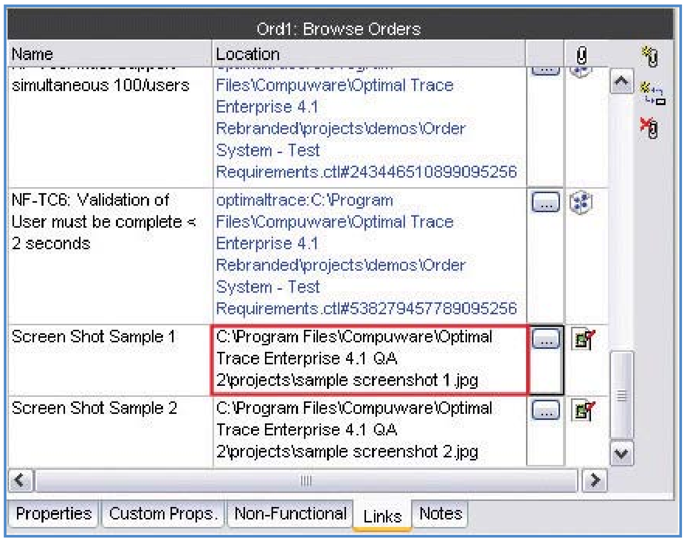
- intricacy, the depth of nesting (requirements within requirements), average number of requirements or steps per scenario or package, etc.
- completeness, the number of bad links, empty packages, Structured Requirements with no alternative flows or steps, goals with no associated non-functional requirements, etc.

How do I capture user interfaces/storyboards/wireframes?
Many systems have a visual interface that lets users interact with the system. Ensuring developers design the user interface to directly serve requirements is critical to project success.

Screenshots are certainly very powerful communication aids, especially to the business. They are very focused on the user experience and, as a consequence, there can often be a tendency to gravitate exclusively to them while ignoring the need to consider broader system issues. A screenshot/prototype-driven approach that excludes critical business and operational performance rules is a recipe for serious expectation management challenges and a condition often described as "Prototypitis," that is, a scenario whereby the business likes the prototype so much that it demands changes to it and rapid turnaround. The developers make the changes to the prototype. The customer reviews it and demands more changes. The "real" system never gets built. An extreme form of this is where the "prototype" system actually goes into production, akin to building a house with no foundations and no internal walls.

Therefore, it is vital to balance the screenshots with the appropriate set of Structured Requirements and associated non-functional requirements, all working together to provide a complete contract.

For each requirement that needs a user interface, attach a screenshot or sketch of what the screen layout might look like. Make sure you are not committing to build the user interface (or system) exactly like this. The screenshot should be clearly annotated to reflect this condition.


Ord1: Browse Orders

Name	Location
simultaneous 100/users	Files\Compuware\Optimal Trace Enterprise 4.1 Rebranded\projects\demos\Order System - Test Requirements.ctl#243446510899095256
NF-TC6: Validation of User must be complete < 2 seconds	optimaltrace:C:\Program Files\Compuware\Optimal Trace Enterprise 4.1 Rebranded\projects\demos\Order System - Test Requirements.ctl#538279457789095256
Screen Shot Sample 1	C:\Program Files\Compuware\Optimal Trace Enterprise 4.1 QA 2\projects\sample screenshot 1.jpg
Screen Shot Sample 2	C:\Program Files\Compuware\Optimal Trace Enterprise 4.1 QA 2\projects\sample screenshot 2.jpg

Figure 7: User-interface screenshots and other artifacts can be linked to specific requirements or the system as a whole in Optimal Trace.

This approach avoids the need to prototype, while giving the business user an indication of the possible look-and-feel of the system. It is a good idea to ensure all interface "mock-ups" have been sanctioned by the design/architect team prior to sharing them with the users. This is important, since sharing mock-ups has the effect of setting user expectations of final look-and-feel, whatever health warnings have been clearly attached to the storyboards! If it emerges that a suggested interface is unrealistic (say, for example, due to constraints that are imposed through NFRs) then this will affect the architecture of the system but, more importantly, will mean re-calibrating the users' expectations. That's something you want to avoid if at all possible.

In the case of very complex user interfaces, it can be useful to build a working prototype that allows for user validation while concurrently validating and building out the associated structured requirements. The key is to control stakeholder expectations and have the prototype serve the requirements rather than the other way round. This balance is critical.

Figure 7 represents how a storyboard or screenshot gets linked to a requirement in Optimal Trace.

How do I handle business rules?

Business rules mean different things to different people but can generally be placed into a number of categories:

- Statements of fact. In a telephone billing system, you might state: Rule 1 - The volume discount rate is 15 percent. Rule 2 - The number of hours after which the volume discount applies is 20.
- Statements of validation. In a retail banking system, you might state: Rule 3 - A customer always has a bank account. Rule 4 - A customer has the following details: First Name, Middle Name, Last Name, Title (one of Miss, Mr., Mrs., Other), E-mail Address, Address 1, Address 2, City/town, State/County, ZIP/Postal, Country.
- Statements of quality. In a consumer-facing online ordering system, you might state: Rule 5 - The system must support at least 1,000 concurrent users (connections). Rule 6 - The credit card approval process must be secure.
- Statements of sequence. In a payments system, you might state: Rule 7 - If the credit card payment is not authorized then notify the retailer. Rule 8 - If payment is not received (net 60 terms) then issue overdue notice.

In a Structured Requirements-driven approach, you represent business rules in a project as either:

- steps in a given requirement
- non-functional requirements associated with a Structured Requirement with custom property type specified as a bound value called "Business Rule"
- as definitions contained within a glossary or actor description
- or as custom fields at a requirement or step level.

The chart below summarizes this:

What is the best way to capture data requirements?
Following on from the previous section, data requirements and statements of validation are synonymous. Although we suggested that these be stored as steps or custom properties, there are in fact a number of alternative ways in which you can capture data requirements.

The exact approach you choose will depend on preference and applicability. In Optimal Trace, you can define a "data-oriented" document profile that generates documents that look and feel much more tabular and data-centric.

(See www.compuware.com/products/optimaltrace for online demos and more information on document profile customization.)

Approaches for capturing data requirements

1. Dedicated data package: This approach is to have a step with refinements to a dedicated "data" package. Each step in the dedicated data package represents a single data field with properties: "Label," "Field Type," "Field Length" and "Mandatory."
2. Step with custom property: This approach is where steps house the detail with specific custom properties, for example, "Fields," "Labels," "Field Types" and a "Mandatory" attribute. Each step houses all fields.
3. Step with no custom property: Expressing all the data in the description column only, and not leveraging step-level custom properties at all. This is the approach used in the demo project: "Order System - (articles).ctl" that ships with Optimal Trace.
4. Glossary: Using the glossary to specify entities and, optionally, data definitions as custom properties at the glossary level.

Business Rule Type	Representation
Statement of fact	Either as a step within a requirement of a glossary/actor entry or as a requirement's custom property
Statement of validation	Step within a requirement or as a requirement's custom property
Statement of quality	A non-functional requirement (NFR) with category set appropriately
Statement of sequence	Steps in a scenario (main alternative) within a requirement

How to represent business rules in Structured Requirements.

How do I deal with existing systems?
It is not uncommon to find "brown-field" sites where current systems require extensions or enhancements. Original documentation for these systems may be lost or badly out of sync with the actual deployed system. What do you do?

First, examine the way people are using the current system. This will likely involve walking through the current system online with existing system users and stakeholders originally involved (if available). Document these usage flows as Structured Requirements. Each distinct dialogue, and its objective, between the user and the system itself becomes a discrete Structured Requirement and each interaction as part of that dialogue becomes a step in the requirement. Keep these requirements externally facing.

Normally, the purpose of documenting in this way is to introduce enhancements or modifications to an existing system(s). For any requirements that are intended to be enhanced, you should describe all scenarios (both main and alternatives). The alternative scenarios will typically describe exceptional situations such as "Invalid Details Entered."

There may well be aspects of the system that were originally designed but are not currently used. Since we are only concerned with extending the current system, we can safely ignore these. Similarly, there is no need to describe in detail any requirements that will not be affected by the planned changes. A simple requirement with name and description suffices for these.

For those requirements that require changes, specifically identify the steps and introduce new alternative scenarios or new requirements that outline the intended change in behavior. These then describe exactly the change required and form the basis for the project plan. Iterate in the delivery cycle using collections of requirements and scenarios needing adjustment.

Project planning and delivery

Manage customer deliveries by requirement set
Ensure the customer expects delivery of portions of the system at a point that coincides with a finished iteration. If the customer is anxious to review progress, let him/her have real, working pieces of functionality. One of the key dangers with customer review cycles is offering an initial prototype that the customer perceives to be an actual system. The customer might want to use the prototype now, although it is, in fact, still very much a throwaway. Avoid this situation by delivering increments of functionality instead of prototypes. Doing so ensures the focus remains on the ultimate system delivery rather than on endless prototyping, in which you never build the real system.

Plan for incremental delivery
Project planning should center on incremental delivery. Project deliverables should be structured in terms of iterations, each iteration corresponding to a group of Structured Requirements. Four to eight weeks is a sensible delivery increment per iteration, depending on the system's overall complexity and project size. Even in very large system developments (with project teams of 15 to 100-plus), it is always a good idea to partition the overall project into sub-projects, each of which consists of several iterations. This reduces risk by providing a mechanism for accessing completeness, allowing visibility for the project manager and, if required, allowing the project manager to de-scope or reprioritize aspects of the system during the project life cycle.

As a rule of thumb, each project should contain three iterations. It is a good idea to consider micro-increments within an iteration. See "Surviving Object-Oriented Projects" for more details on this approach.[1]

In the absence of Structured Requirements, incremental delivery is extremely challenging. Why? It is very difficult to partition the delivery cycle so increments yield sets of tangible value to all stakeholders. For non-structured projects, increments are often assumed to comprise of sets of technical components. This invalidates one of the primary motivations for incremental delivery in the first place: Ensuring project managers can measure completeness and validate the system in segments, with each segment offering value to business stakeholders. Increments based on Structured Requirements allow this to happen; whereas, segmenting increments around technical components, as is the case with non-structured or traditional approaches, means interdependencies often lead to one large development cycle, in effect, one large increment. This leads to significant risk, as it is back to a pure-play waterfall approach with all of the associated disadvantages.

[1] *"Increments and Iterations," Alistair Cockburn, Addison-Wesley, 1998.*

Here is a suggested approach for grouping structured requirements into iterations:

1. Prioritize the requirements by business importance.
2. Prioritize the requirements by technical difficulty. You should involve the technical architects and have them review each structured requirement.
3. Group related requirements into packages.
4. Prioritize the delivery of each package.
5. Keep duration of iteration to approximately four to eight calendar weeks, if possible (this applies if you are in an agile or iterative-based environment).
6. If longer than eight weeks, then there is higher risk that the architecture and/or estimations are flawed.
7. Within the project plan, schedule the "most difficult" and highest-business-priority iterations first, as these will incur the most risk and are most important to the business.

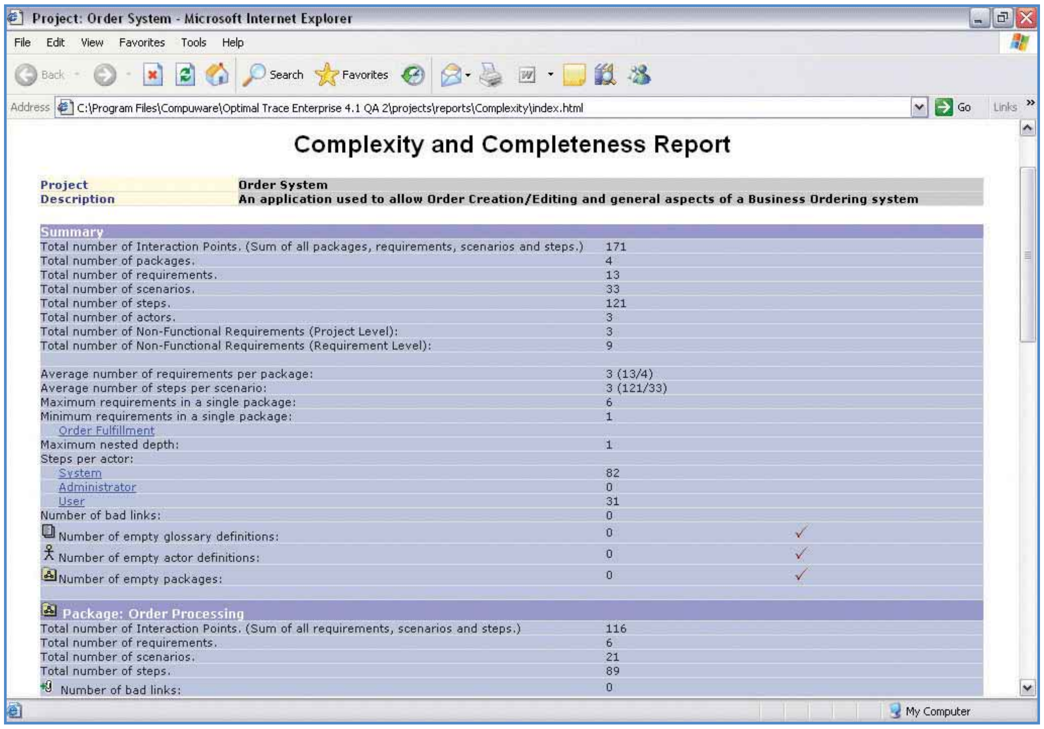
Estimate the project plan

When performing estimation, list all of the requirements and scenarios against components in a tabular format. (It is easy to do with Optimal Trace's automatically generated documentation.) The objective is to cross-reference requirements against the current components (if modifying an existing system) or against the anticipated new components (if this is a new system initiative).

If using Excel, depending on how many requirements and components you have, you can use one or multiple Excel sheets. If the latter is the case, then cross-reference each group of requirements to a given tab. Then mark against each component whether or not that requirement affects it. Correlate each affected component and estimate the required change in terms of hours. During the first increment of delivery, it is likely that these estimates will be refined and solidified, and so on throughout the project.

Optimal Trace provides a pre-canned report that helps estimation. It is called the Complexity and Completeness report (See Figure 8.)

As you start analysis on new projects, you can run this report and, by comparing the number of interaction points with production projects, you will be able to get an indicative sense of the relative complexity.



Project: Order System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address C:\Program Files\Compuware\Optimal Trace Enterprise 4.1 QA 2\projects\reports\Complexity\index.html
Go Links
My Computer

Complexity and Completeness Report

Project	**Order System**
Description	**An application used to allow Order Creation/Editing and general aspects of a Business Ordering system**

Summary		
Total number of Interaction Points. (Sum of all packages, requirements, scenarios and steps.)	171	
Total number of packages.	4	
Total number of requirements.	13	
Total number of scenarios.	33	
Total number of steps.	121	
Total number of actors.	3	
Total number of Non-Functional Requirements (Project Level):	3	
Total number of Non-Functional Requirements (Requirement Level):	9	
Average number of requirements per package:	3 (13/4)	
Average number of steps per scenario:	3 (121/33)	
Maximum requirements in a single package:	6	
Minimum requirements in a single package:	1	
Order Fulfillment		
Maximum nested depth:	1	
Steps per actor:		
System	82	
Administrator	0	
User	31	
Number of bad links:	0	
Number of empty glossary definitions:	0	✓
Number of empty actor definitions:	0	✓
Number of empty packages:	0	✓

Package: Order Processing	
Total number of Interaction Points. (Sum of all requirements, scenarios and steps.)	116
Total number of requirements.	6
Total number of scenarios.	21
Total number of steps.	89
Number of bad links:	0

Figure 8:
The Complexity and Completeness Report is one of a number of Optimal Trace's pre-canned report templates.

Common project planning questions

How do I handle change during and after the analysis phase?

Bear in mind that requirements evolve during the analysis phase and even after sign-off. So, for example, we may discover an early-stage Structured Requirement may need more detail later on in the analysis. Early-stage requirements typically comprise a simple name and description and some basic steps. Once you achieve agreement between the participating stakeholders, you typically define more detail and recognize and capture alternate scenarios. The final iteration of the analysis phase should result in requirements that have been prioritized, have value to the customer and have enough detail that they can be implemented by a development team. At this stage, all stakeholders are positioned to proceed and sign-off is significantly easier.

Important for business analysts and project managers, this approach to requirements management makes documentation extremely reliable. Disputes about what was signed off rarely arise because the project data is unambiguously set out at every stage in documentation in the format most suitable for each stakeholder.

The table below outlines some sample metrics recorded during a typical analysis phase.

Checkpoint questions to ask before sign-off and after each iteration:

- How many requirements have you defined?
- How many alternative scenarios exist?
- Have you only considered the main success scenarios or have you covered alternate flows too?
- How many steps are in each main success scenario?
- Have you identified non-functional requirements?

Even after sign-off and after the first iterations of delivery have begun, it is not unusual to discover aspects of the requirements that are incomplete or need further clarification or refinement. An acceptable level of change would be +/- 10 percent after sign-off. Typically, clarification queries will emanate from the development or QA team, which can now address the task of developing the test specifications and test plan.

Baselining in Optimal Trace after each significant change provides comparison points during the life cycle. Republishing the requirements documents in Optimal Trace along with the associated baseline comparison HTML report creates a tight communication loop. Finally project managers, analysts and all stakeholders should register for e-mail notifications on changes to specific aspects of the project of direct interest to them. This gives high visibility to all concerned stakeholders into aspects of the project of direct interest.

	Analysis Phase		
	Iteration 1	Iteration 2	Iteration 3
Number of high-level requirements	18	26	29
New requirements discovered	N/A	11	3
Requirements dropped or deemed out of scope	N/A	3	2
Avg. number of alt. scenarios per requirement	0	4	7
Avg. number of steps per requirement	2	12	20
Avg. number of NFRs per requirement	0	2	4

Sample metrics recorded during a typical analysis phase.

How do I control new releases and what is the best strategy for handling maintenance releases?
Optimal Trace projects can be baselined to create a "line in the sand" so that incremental changes to a system are easy to track and manage. Alternatively, new projects can be cloned from the original project, making it easy to kick off similar projects or new releases of an existing system. Optimal Trace also enables multiple projects to be linked to each other. In this way, projects (each equivalent to a release) can be cross-referenced and managed.

There are two options in Optimal Trace for handling maintenance releases:

1. Handle maintenance releases as separate projects.

2. Handle maintenance changes using baselines.

Which approach you use is normally dictated by:

- the degree of change
- the size of the maintenance release
- the number of roles involved (QA, development project manager).

As a rule of thumb, if the maintenance scenario involves a relatively high level of additional change to the original project, then having a dedicated project which optionally links back to the original project elements makes the most sense. In this context, anything above a 15 percent deviation within the most significant Structured Requirements justifies a new project.

How do I avoid over-engineering?
As outlined above, Optimal Trace's process of narrowing down each business driver into its constituent Structured Requirements and associated steps, alternate flows and branches ensures complete and accurate scoping of the project. This avoids the risk of vague or ambiguous requirements that lead to over-engineering.

By fleshing out the goals and requirements with the business user, your system will be fully expounded before design work is started. In this way, you fully capture the business users' needs and map them into the final system architecture, design and development without missing anything or extending the project beyond the necessary scope.

How do I avoid scope creep?
Iteration of the Structured Requirements with the business user yields a complete set of requirements for the system. Optimal Trace helps you manage scope creep easily by tracking all amendments and changes. Optimal Trace's automatic change tracking facility and generation of suspect links highlights the extent of change, one iteration to the next (enabling the business analyst or project manager to immediately identify trouble spots), and also indicate the impact that amendments and changes will have on the overall system. Optimal Trace can also automatically notify you of changes to particular aspects of the project by e-mail with links to the changes involved. All of this easy-to-use functionality gives full visibility of the system at all times, as teams collaborate and evolve the system requirements.

How do I avoid rework?
Inaccurate or incomplete requirements cause late-cycle problems that often require expensive reworking. By capturing a complete and accurate set of system requirements early in the development life cycle, you can put projects on the right footing as soon as possible. All project work is traceable to a specific step, requirement and business driver.

How do I control access?
It is important all stakeholders are given a role in determining the project requirements. Nonetheless, the management of large, complex projects depends on the ability to separate issues that must be addressed as opposed to the "nice-to-haves." Getting the buy-in of a restricted number of senior domain experts on the business users' side dramatically helps narrow down large projects to the essential elements that will deliver on the business need. Other, less directly involved stakeholders can easily review and submit proposed edits to the aspects of the project that concern them, online or off-line. This is the most powerful way to secure accuracy and buy-in throughout the project life cycle. Manage controlled access to the Structured Requirements and supporting documentation by limiting access rights to certain people and offering documentation created as PDFs more generally.

Optimal Trace's automatic document generation also allows users to communicate in the format most comfortable for them. This simplifies the process of communicating to stakeholders as project requirements evolve. Particularly helpful when dealing with large groups, approved edits can be reincorporated into the main project in a controlled way and with structured version control. Optimal Trace makes managing project edits and communicating project status a lot easier, and a lot more reliable. The use of custom properties and full logical complex querying means the project can be easily sorted (based on high-risk, priority, completed, etc). Optimal Trace gives a better, more flexible view of project data and as a result, prioritization, reporting and tracking are made much easier.

How do I relate requirements to design and build?
It is vital the system design and build are traced directly from the original business requirements. Business analysts and project managers can work with technical staff to map those requirements into appropriate components or system interfaces and help identify suitable business objectives within a given iteration. In building out the architecture and design plan, each requirement and its associated elements are mapped to the design plan. To assure maximum efficiency and project success, trace every piece of design work to a requirement or one of its elements (alternate flow, NFR, etc.) and vice versa.

An absence of traceability indicates there may be potential issues with the accuracy of the requirements, or the team is building something outside the system's scope. Tracing at this stage in the process is plainly an invaluable tool in identifying shortfall/overlap in the design plans.

Optimal Trace's integrations with many leading development tools enable a seamless mapping of requirements through to the development environment. By comprehensively describing the system with a set of Structured Requirements, associated NFRs and other artifacts (screenshots, etc.), customers and business users do not have to get involved with aspects of the system's design. This is a good thing, because UML modeling tools are primarily used by technical teams and can be intimidating to the non-technical user. Also, representations of the project in development tools do not completely reflect actual requirements. They help visualize the system but do not capture business rules or represent scenarios, sub-requirement elements or non-functional requirements. These modeling tools are effective for diagrams but are not text-friendly or useful for requirements management.

Testing and validation

One of the biggest challenges facing QA and test leads is the lack of effective tracing from business and system requirements to the actual test requirements, and vice versa. This potential for disconnect between QA and business stakeholders has implications for development but even more so for QA and test. It can be directly attributed to a lack of Structured Requirements coming out of the analysis phase.

A second challenge is the duplication of effort that is seen quite commonly with respect to the various activities of QA teams. Traditional requirements are usually assertion-based; these are statements of fact, like statements of business rules. For QA to interpret such requirements effectively and turn them into something that can be coherently tested and measured is difficult, time-consuming and creates duplication. Stating a set of project requirements completely in a verb-based, goal-oriented manner makes it immediately easier for QA/test to see what tests are required to ensure the system delivers on primary business drivers.

Testing and QA also tend to happen very late-cycle, long after the system "concrete" has been set. This is the classic tale with which we are all well familiar. The QA lead reports performance issues to project managers and gets short shrift because it is too costly and inconvenient to consider re-architecting so late in the cycle. These issues result from QA being siloed and not given the priority necessary for full, effective and timely intervention.

Complete capture and clear communication of project requirements in a structured way go a long way toward connecting analysis, development and test, introducing QA issues earlier in the development cycle and reducing duplication of work. A set of system requirements which defines functional and non-functional scope makes system validation and test much easier and more accurate.

Create a specific test project
Initially you should create a separate Optimal Trace project for your test plan. While relating to the requirements project, test projects are structurally different since they have a testing-oriented package structure and different custom properties. Create a test project from the QA Project Template that ships with Optimal Trace. Over time, you will also probably modify this template to add specific fields relevant to your own environment.

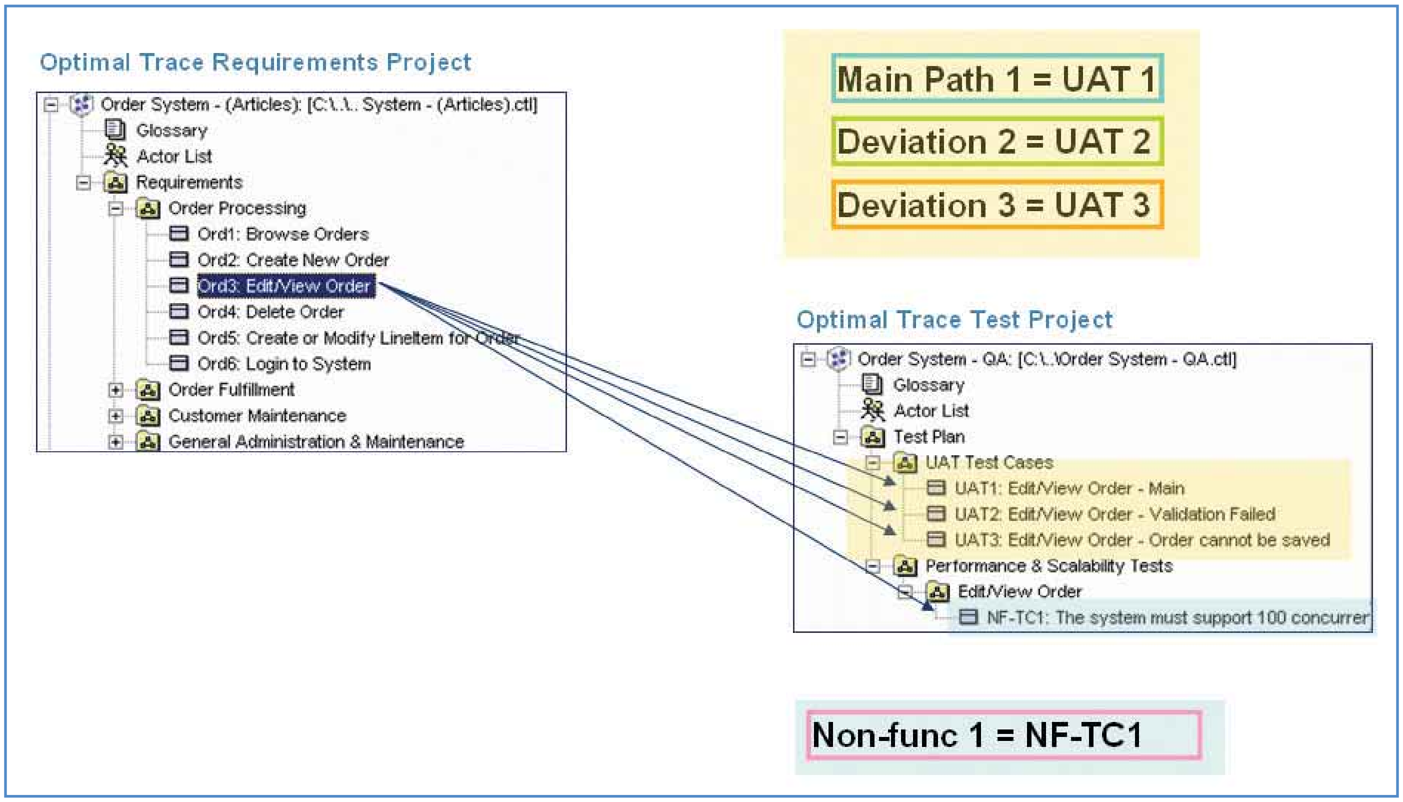

Optimal Trace Requirements Project
Order System - (Articles): [C:\..\.. System - (Articles).ctl]
Glossary
Actor List
Requirements
Order Processing
Ord1: Browse Orders
Ord2: Create New Order
Ord3: Edit/View Order
Ord4: Delete Order
Ord5: Create or Modify LineItem for Order
Ord6: Login to System
Order Fulfillment
Customer Maintenance
General Administration & Maintenance
Main Path 1 = UAT 1
Deviation 2 = UAT 2
Deviation 3 = UAT 3
Optimal Trace Test Project
Order System - QA: [C:\..\Order System - QA.ctl]
Glossary
Actor List
Test Plan
UAT Test Cases
UAT1: Edit/View Order - Main
UAT2: Edit/View Order - Validation Failed
UAT3: Edit/View Order - Order cannot be saved
Performance & Scalability Tests
Edit/View Order
NF-TC1: The system must support 100 concurrer
Non-func 1 = NF-TC1

Figure 9: Structured Requirements mapped directly to test cases.

Test projects will tend to have trace links back to the original business-facing project thereby aligning them with the original requirements project. QA stakeholders should also consider subscribing via e-mail notifications to changes on specific parts of the business project that they are testing. This makes it possible to review any changes applied and modify associated test cases. Optimal Trace's automatic generation of suspect links resulting from project changes also facilitates immediate reviews, allowing impact analysis between the test and requirements projects.

Develop test specifications based on requirements
Structured Requirements comprise scenarios and each scenario equals one functional test case. Always ensure the complete path of every requirement has an associated test case. In this way, the functional test specification comprises all scenarios.

Also, all non-functional requirements associated either with the project or with specific requirements should be written as a unit or system test. Test data should be associated with each resulting test. NFR qualities at the system level generally become system tests; NFR constraints become manual tests.

Generate the initial test specification
Optimal Trace can automatically generate the basis for a complete test plan based on the Structured Requirements, alternate scenarios, NFRs, etc.

To do this, Optimal Trace analyzes the set of Structured Requirements with all its steps, scenarios and descriptions and associated NFRs and from this generates a separate test project. This process automatically itemizes the unique paths within the overall Structured Requirements and reflects a complete set of test cases to enable thorough functional testing.

Similarly, a full set of non-functional requirements associated with the Structured Requirements ensures all system requirements are carried over to the testing environment for validation.

Looking at the demo project "Order System" that ships with Optimal Trace, Figure 9 depicts the "Edit/View Order" Structured Requirement as entered by the analyst and how it maps to a test project. Running Optimal Trace's automatic test case generation from this requirement will yield four tests. The three functional paths (one main path and two deviations) need to be tested and therefore yield three functional tests. Additionally a single NFR yields one non-functional or system test.

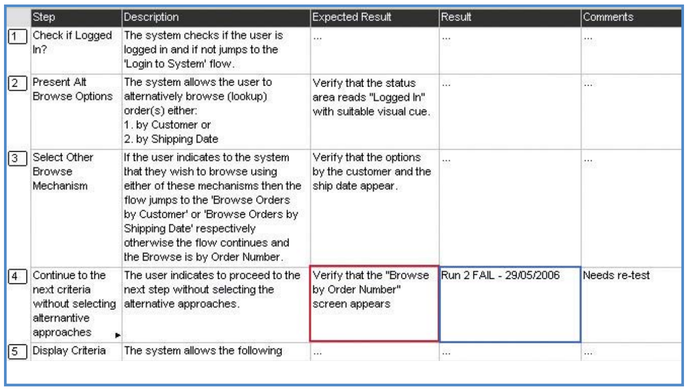
Add validation and expected result information
The test cases automatically generated from the Structured Requirements and associated non-functional requirements are the starting point for establishing the specification. We now need to look at them from a validation perspective.

Rather than treating the test cases as requirements (e.g., "The system needs to" or "The user will enter"), we express them in terms of what the system needs to validate. In other words, the requirements need to be inverted with validation as the focus. The test steps generated by the original Structured Requirements are augmented and enriched by the additional validation focus, considering test data, etc.

The real goal here is to prime the test pump, increasing the speed and efficiency by which tests can be generated, often even before any development has begun. This process allows the tests to be generated before any parts of the system are built by the development community. By putting the two sides of the equation, test and requirements, together earlier in the life cycle, you get better quality requirements, better quality test cases and aligned system delivery. This ensures a direct trace from the test process back to requirements and vice versa.

Optimal Trace users quite often generate tests very early in the development cycle, perhaps even in the first few weeks of analysis, to get the sense and scale of what is coming down the pipe and what may be missed during the early phase. This ensures as many as possible of the right questions are asked early in the process. Let's see how we do this for functional and non-functional tests.

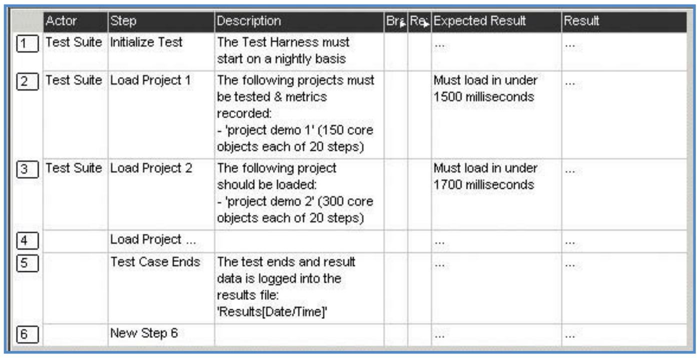
Add validation information to functional tests
To add validation information to the functional tests, you need to modify the description and add detail to the "expected result" field for each step. Modifying the description means we are clear as to what needs to be validated for that step. This takes the form of an instruction such as "Click New Order" or "Verify that the following fields are displayed." Detailing what you expect the result to be allows you to access pass/fail results (see Figure 10).



	Step	Description	Expected Result	Result	Comments
1	Check if Logged In?	The system checks if the user is logged in and if not jumps to the 'Login to System' flow.	...	...	...
2	Present Alt Browse Options	The system allows the user to alternatively browse (lookup) order(s) either: 1. by Customer or 2. by Shipping Date	Verify that the status area reads "Logged In" with suitable visual cue.	...	...
3	Select Other Browse Mechanism	If the user indicates to the system that they wish to browse using either of these mechanisms then the flow jumps to the 'Browse Orders by Customer' or 'Browse Orders by Shipping Date' respectively otherwise the flow continues and the Browse is by Order Number.	Verify that the options by the customer and the ship date appear.	...	...
4	Continue to the next criteria without selecting alternantive approaches	The user indicates to proceed to the next step without selecting the alternative approaches.	Verify that the "Browse by Order Number" screen appears	Run 2 FAIL - 29/05/2006	Needs re-test
5	Display Criteria	The system allows the following	...	...	...

Figure 10 : Validation information associated with functional requirements.

Add validation information to non-functional (system) tests
Each non-functional test needs to be fleshed out to provide the detailed steps that comprise the test. Similar to our functional tests, the description and expected result house the instructions and pass/fail criteria.

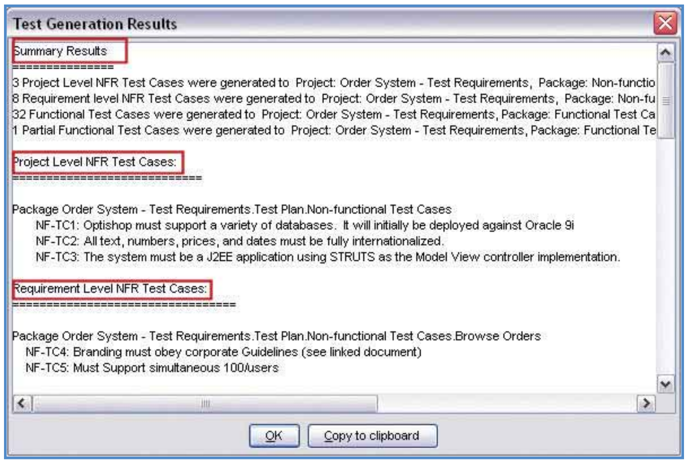


	Actor	Step	Description	Br	Re	Expected Result	Result
1	Test Suite	Initialize Test	The Test Harness must start on a nightly basis			...	...
2	Test Suite	Load Project 1	The following projects must be tested & metrics recorded: - 'project demo 1' (150 core objects each of 20 steps)			Must load in under 1500 milliseconds	...
3	Test Suite	Load Project 2	The following project should be loaded: - 'project demo 2' (300 core objects each of 20 steps)			Must load in under 1700 milliseconds	...
4		Load Project ...				...	...
5		Test Case Ends	The test ends and result data is logged into the results file: 'Results[Date/Time]'			...	...
6		New Step 6				...	...

Figure 11: Test result information stored in Optimal Trace.

Store test run results
If using Optimal Trace solely for your testing, without the support of automated test tools such as Compuware QACenter, you will want to store the results in the project. To do this, perform the following:

1. Create a custom column for each step.
2. A column in Optimal Trace is a custom property at step level called "Result."



Test Generation Results

```
Summary Results
===============
3 Project Level NFR Test Cases were generated to  Project: Order System - Test Requirements,  Package: Non-functio
8 Requirement level NFR Test Cases were generated to  Project: Order System - Test Requirements,  Package: Non-fu
32 Functional Test Cases were generated to  Project: Order System - Test Requirements, Package: Functional Test Ca
1 Partial Functional Test Cases were generated to  Project: Order System - Test Requirements, Package: Functional Te

Project Level NFR Test Cases:
============================

Package Order System - Test Requirements.Test Plan.Non-functional Test Cases
    NF-TC1: Optishop must support a variety of databases.  It will initially be deployed against Oracle 9i
    NF-TC2: All text, numbers, prices, and dates must be fully internationalized.
    NF-TC3: The system must be a J2EE application using STRUTS as the Model View controller implementation.

Requirement Level NFR Test Cases:
=================================

Package Order System - Test Requirements.Test Plan.Non-functional Test Cases.Browse Orders
  NF-TC4: Branding must obey corporate Guidelines (see linked document)
  NF-TC5: Must Support simultaneous 100/users
```

Figure 12: Test generation results from Optimal Trace.

Note this is the default in the shipped "Optimal Trace QA Project Template." (See Figure 11.)

Now on the test requirement itself, do the following:

1. Create sets of custom fields at a test case level, one set for each anticipated test run.
2. "Run 1 Data," "Expected Result 1," "Actual Result 1" and "Tester 1."

Common QA and test-related questions

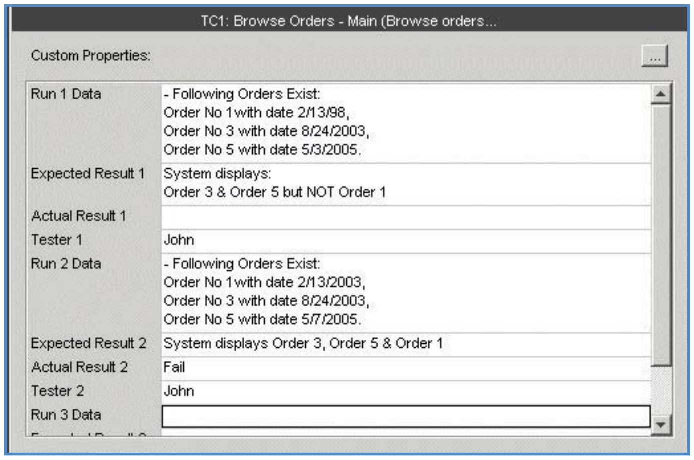
How do I verify and test requirements for quality and completeness?
The business goals, which have been deeply specified in the early analysis phase, can be used as the basis of user acceptance testing. All associated non-functional requirements must also be mapped to test. Structured Requirements make it easier to capture complete and accurate system requirements. If these are fully mapped to the test environment, then the test process will be properly focused on making sure the system fulfills the business goals and objectives originally intended.

How do I avoid testing the wrong stuff?
If the testing plan has been generated from the project's Structured Requirements, then you won't test the wrong aspects of the system. It is important to cover all aspects of the requirements, though, making sure to test NFRs, alternate scenarios, etc. That way, you'll be sure to have a complete test plan and stay on track.

What exactly does Optimal Trace test generation yield?
Test generation in Optimal Trace equates to:

- "primers" for the test cases, more needs to be done, data, etc.
- a set of unique paths for each Structured Requirement
- coverage of associated non-functional requirements (e.g., qualities and constraints)
- not crossing functional requirements (do not traverse "branches")
- not carrying over custom properties
- automatically tracing from test back to requirement or vice versa.

It is important to note you are not obliged to use Optimal Trace's test generation to create test cases; it is an accelerator. You could alternatively copy and paste directly from the requirements project.



TC1: Browse Orders - Main (Browse orders...

Custom Properties:

Run 1 Data	- Following Orders Exist: Order No 1with date 2/13/98, Order No 3 with date 8/24/2003, Order No 5 with date 5/3/2005.
Expected Result 1	System displays: Order 3 & Order 5 but NOT Order 1
Actual Result 1	
Tester 1	John
Run 2 Data	- Following Orders Exist: Order No 1with date 2/13/2003, Order No 3 with date 8/24/2003, Order No 5 with date 5/7/2005.
Expected Result 2	System displays Order 3, Order 5 & Order 1
Actual Result 2	Fail
Tester 2	John
Run 3 Data	

Figure 13: Test run results associated with a requirement: expected and actual.

Conclusion

The paper highlights the real benefits that a Structured Requirements approach will bring to your projects. A structured and logical flow framework for requirements capture and management ensures that projects can be expanded fully and accurately. The ability to add alternative flows and branches, and attach screenshots and other artifacts means each project presents a complete picture of the project and what it needs to achieve.

The ability to trace all project activity back through requirements to the original business drivers reduces risk throughout the life cycle. Rework is reduced, QA and test can focus on actual project deliverables and IT investment is always linked to enterprise-based benefits.

Most important of all, this approach draws all stakeholders together in a collaborative process that is easy to communicate, understand and adopt. This accelerates project sign-off and reduces disputes after the fact. Primarily, though, it ensures all stakeholders are involved at an early stage when their contributions are most valuable, radically improving the chance of the project successfully delivering on the original business drivers.

When adopting a Structured Requirements approach, it is vital to apply it thoroughly and diligently. The tips outlined here should help you in securing project success, whether you are involved in business analysis, project management, QA and test, coding or architecting. At a higher level, financial control, portfolio management and corporate compliance are all made easier as a result. The Structured Requirements approach enabled by Optimal Trace has something to offer everyone involved in IT investment.

Fergal McGovern is the originator of, and Product Manager for Optimal Trace, Compuware's leading Requirements Definition and Management solution. He has spent many years applying structured requirements techniques to IT projects, aiming for improved project delivery and success.

He has held senior roles with Ebeon, Iona Technologies and various Fortune 1000 companies. Fergal is a leading commentator and practioner on Requirements Strategy.

To learn more or demo our product, visit www.compuware.com/products/optimaltrace