MSFC-460

IAC-07-D3.4./D3.5/E5.5.08

LESSONS LEARNED FROM DEPLOYING AN ANALYTICAL TASK MANAGEMENT DATABASE

Mr. Daniel A. O'Neil[1]
daniel.a.oneil@nasa.gov
Ms. Clara Welch[1]
Clara.Welch@nasa.gov
Mr. Joshua Arceneaux
Booz Allen Hamilton
Houston, TX 77058,United States
arceneaux_joshua@bah.com
Mr. Dennis Bulgatz[2]
bulgatz@ama-inc.com
Mr. Mitch Hunt[2]
mitch.hunt.007@gmail.com
Mr. Stephen Young[2]
young@ama-inc.com

[1]National Aeronautics and Space Administration (NASA),
Huntsville, AL 35812, United States
[2] Analytical Mechanics Associates (AMA),
Huntsville, AL 35816, United States

ABSTRACT

Defining requirements, missions, technologies, and concepts for space exploration involves multiple levels of organizations, teams of people with complementary skills, and analytical models and simulations. Analytical activities range from filling a To-Be-Determined (TBD) in a requirement to creating animations and simulations of exploration missions. In a program as large as returning to the Moon, there are hundreds of simultaneous analysis activities. A way to manage and integrate efforts of this magnitude is to deploy a centralized database that provides the capability to define tasks, identify resources, describe products, schedule deliveries, and generate a variety of reports. This paper describes a web-accessible task management system and explains the lessons learned during the development and deployment of the database. Through the database, managers and team leaders can define tasks, establish review schedules, assign teams, link tasks to specific requirements, identify products, and link the task data records to external repositories that contain the products. Data filters and spreadsheet export utilities provide a powerful capability to create custom reports. Import utilities provide a means to populate the database from previously filled form files. Within a four month period, a small team analyzed requirements, developed a prototype, conducted multiple system demonstrations, and deployed a working system supporting hundreds of users across the aerospace community. Open-source technologies and agile software development techniques, applied by a skilled team, enabled this impressive achievement. Topics in the paper cover the web application technologies, agile software development, an overview of the system's functions and features, dealing with increasing scope, and deploying new versions of the system.

INTRODUCTION

Defining mission and system requirements for large aerospace projects involve a myriad of trade studies conducted by geographically distributed teams. Ensuring efficient use of resources requires a tremendous coordination effort. Within NASA's Constellation Program, the Architecture Trades and Analysis (ATA) Office defines the analysis cycle goals, objectives, and to integrate the analyses across the Constellation Program (CxP). Before 2007, this process, known as the Integrated Design and Analysis Cycle (IDAC), used a Microsoft Word document form called a Task Description Sheet (TDS) to define a study task. Collecting, coordinating, and reporting on the TDS forms required several people who found it difficult to track the status and dependencies among the studies using Microsoft Excel. Between October 1st and December 7th, of 2006, a small development team with representatives from NASA, Booz Allen and Hamilton (BAH), and Analytical Mechanics Associates (AMA) defined requirements, designed, developed, and deployed the Constellation Analysis Integration Tool (CAIT) database. Using CAIT, the ATA office can collect, coordinate, integrate, and report on all of the study activities within the program. In February of 2007, the Directorate Integration Office (DIO) within NASA's Exploration Systems Mission Directorate (ESMD) defined requirements to expand the scope of the database from managing tasks for one program to managing and reporting tasks throughout ESMD. This paper describes the process applied by the development team and the lessons learned about developing an enterprise task management database.

Acronyms

AMA	Analytical Mechanics Associates
ATA	Architecture Trades Analysis
BAH	Booz Allen and Hamilton
CAIT	Constellation Analysis Integration Tool
CxP	Constellation Program
DAC	Design Analysis Cycle
DIO	Directorate Integration Office
ER	Entity Relationship
ESMD	Exploration Systems Mission Directorate
FOSS	Free Open Source Software
GNU	GNU's Not Unix
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDAC	Integrated Design and Analysis Cycle
MS&DA	Modeling, Simulation, and Data Architecture
MVC	Model Viewer Controller
RoR	Ruby on Rails
SQL	Structured Query Language
TBx	To Be Determine, Resolved, Supplied, (TBD, TBR, TBS)
TDS	Task Description Sheet
RoR	Ruby on Rails

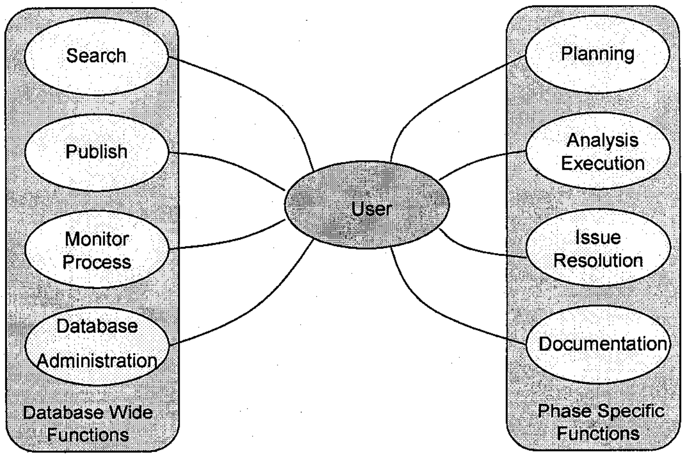


Figure 1: Functions identified in the requirements document

Application Objectives

With the creation of the CxP program office in 2006, the ATA office executed the first iteration of the IDAC for the CxP, IDAC 2. The IDAC 2 process involved study team leaders creating hundreds of TDS in Microsoft Word and storing the files in various directories of a web-based product data management system. Access to the directories where the TDS forms were archived were limited to organizational personnel; consequently, people who did not have access to the TDS archive directory relied on copies that were e-mailed to them. During the course of a study, a TDS author might change the content of the form and update the file in the archive or content that the form "linked to" might change without anyone being made aware of the changes. Personnel who were working with the copy of the original might not receive the updated version of the TDS. Also, given the flexibility of the Word form, several TDS authors modified or added fields to clarify their tasks; this action led to inconsistencies among the TDS files.

FUNCTIONS AND FEATURES

Built upon a a MySQL database, The CAIT application contains 89 tables, storing everything from users to Analysis tasks and the links between them. The Ruby On Rails (RoR) framework makes it easy to change the database to Oracle, SQL-SRV, or other, as the database connection logic is largely stored in the RoR model files. With CAIT, study managers create Task Description Sheet (TDS) to define the work, resources, products, and schedules. Table 1 presents the elements of a TDS.

Introduction	Description, Title, Points of Contact
Timeline	Current Status, Projected Completion Date, Priority
Links to External Items	Requirements (system and mission) Risks (potential problems and impact) Models (analytical tools used) Issues (current problems) Review Boards Resources (analysts, facilities, etc.)
Analysis Data	Inputs Outputs Board Deliverables
Supporting Information	Design Reference Missions Methodologies Mission Phases Purpose System Elements

Table 1: Elements of a TDS

Recognizing the issues associated with managing so many tasks through Word documents and Excel files, the ATA requested the Constellation Modeling, Simulation and Data Architecture (MS&DA) office to create a centralized database for managing tasks associate with the IDAC. Joshua Arceneaux, from BAH, served as the "voice of the customer" by writing a requirements document that explained the database functions and fields needed to capture sufficient data for IDAC management. Figure 1 identifies the functions specified in the ATA's database requirements document.

Why does a program conduct studies? Some studies resolve problems or provide needed detail for requirements. Other studies determine methods for mitigating risks associated with a design. A task within CAIT can link to a specific TBx associated with a requirement or a particular risk.

Linking tasks to people, products, tools, requirements, risks, and issues provides the most powerful function of the CAIT database. Data produced by one task can be an input to another task. Multiple tasks may involve the same human resources and requirements. A TDS record specifies dates that particular review boards receive the results of the task. Reports within the database identify the dependency among the tasks.

Filters implement a search function for finding records based on criteria ranging such as organizations, teams, status, scope, and key-words. Scope indicates whether the focus of a study is a component, system, multiple systems, or a complete exploration architecture. Report generators use the filters to produce status and dependency reports.

Administrative functions provide the capability to create user accounts, assign people to teams, and baseline TDS records. A database administrator can lock or "baseline" records, which prevents constant tweaking of the tasks throughout the IDAC process.

DESIGN PHILOSOPHY

Benefits derived from a custom database include process driven design and maximum flexibility to respond to the changing needs of an organization. When an organization procures a commercial application for managing a process, it is bound by the concepts and terminology embodied in that application. If the application is general purpose enough to accommodate any kind of process then the organization has to determine how to apply the application to the business process. Often, sophisticated general purpose applications require extensive training. With a custom developed application, representatives from the organization participate in the requirements definition and system design. Custom developed applications embody the organizational process so less training is required because the user community understands the concepts and terminology.

Owning the source code positions the application developers to respond quickly to new requirements. Commercial applications often have a number of customers who wish for a wide variety of functions and features. An organization might have to wait years before a commercial product implements a desired feature. An organization could pay the vendor to implement a desired function but how is that different than developing a custom application? The vendor saves research and development funding and releases the new functions in the next version of the proprietary product.

Leery project managers express concerns about custom applications because:

- A small development team might have a single point of failure,
- Programmers don't have time to provide instant help-desk support,
- Custom application tend to have insufficient documentation,
- Quality of custom software does not match commercial applications,
- Continued dependence on the development team that wrote the code

Free Open Source Software (FOSS) alleviates these concerns through standards, community, and automation. Applications created with FOSS benefit from standardized tools, practices, templates, and reusable code. These standards mitigate the risk of one programmer becoming a single point of failure and enable future maintenance by programmers other than the original developers. Well established FOSS has large communities that thoroughly test and document the systems. Online forums hosted by these communities provide technical support to developers. By building upon FOSS, the development team gains tremendous leverage in documentation. Websites dedicated to the FOSS provide required documentation about the internal structure, application programming interface, and tutorials. This foundation allows the development team to focus on the user's guide and documenting the business logic portion. Standards and automation ensures quality through templates and code stubs. Standardized file structures and templates ensure that everything has a place and the code is consistent. Automated generation code stubs enable development of function and regression tests. Programmers populate the code stubs with expected inputs, calls to functions, and error checking code to capture potential bugs. Applying standards, receiving assistance from the community, and gaining leverage from automation enables a development team to offer the technical support, documentation, and quality that users would expect from any commercial application.

According to the Agile Software Development Manifesto[1], This design philosophy values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Initially, the CAIT development team analyzed more than hundred written requirements. After developing a prototype the CAIT team demonstrated it to key personnel within the ATA. Comments about the prototype were positive and indicated that the CAIT team understood the needs of the customer. Clara Welch, who had conducted a thorough analysis of the requirements, pointed out the prototype implemented only a small fraction of the documented requirements. Though the document identified everything the customer could possibly want, the prototype captured what the customer needed to get the job done in the near term. After the demonstration, the CAIT development team and the ATA office agreed to move from a traditional water-fall approach to an agile software development method. Working with Josh Arceneaux, the development team iterated the refinement of the prototype each week until it because a deployable application. Rapid prototyping enabled the team apply the values of agile software development and to deploy a working system within three months.

Design philosophies applied in the CAIT development process involved building upon FOSS, applying the values of agile software development, and demonstrating early and often through rapid prototyping. Building with FOSS enabled rapid prototyping because the development team gained leverage from the standards and automation. Agile software development values ensured that the ATA organization accepted the deployed system. Rapid prototyping engaged the customer and enabled the development team to deploy the system on a tight schedule. To apply these philosophies, the customer had to accept that testing and documentation would come after system deployment. Developers accepted long hard hours during the rapid prototyping phase and the project manager had to manage customer and stake-holder expectations about delivery schedules.

DEVELOPMENT PROCESS

Developing the CAIT database involved requirements analysis, system design through entity-relationship diagrams, rapid prototyping, configuration management, functional and regression testing, system deployment, and meeting with the user community to discuss future improvement. Figure 2 depicts the iterative nature of the process.

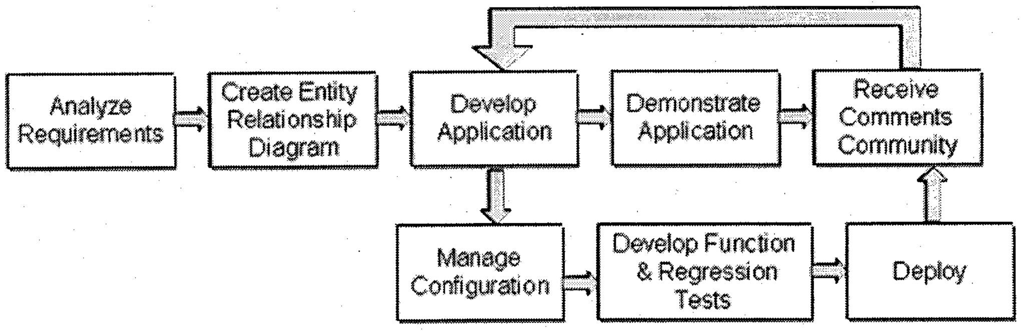

Analyze Requirements
Create Entity Relationship Diagram
Develop Application
Demonstrate Application
Receive Comments Community
Manage Configuration
Develop Function & Regression Tests
Deploy

Figure 2 System Development Process

DB Designer 4

LaMonte Dent, who works at the University of Alabama in Huntsville, made significant contributions to the project by developing an Entity-Relationship (ER) diagram and a prototype of the user interface. Working with LaMonte's ER diagram and tool of choice, Dennis Bulgatz and Josh Arceneaux used DB Designer 4 to identify the data objects and interfaces within the CAIT database. Integrating design, modeling, creation, and maintenance into a seamless environment, Fabulous Force Database Tools offers DB Designer for free. Figure 3 presents a screen-shot of an ER diagram that derived from the requirements document.

After several iterations of the diagram, the development team produced a prototype of the system. With the deployment of the system, the emphasis shifted from the design to system improvements so the development team ceased work on the ER diagram. Plans for future development will return to the ER diagram to document the "As-Deployed" version of the system.

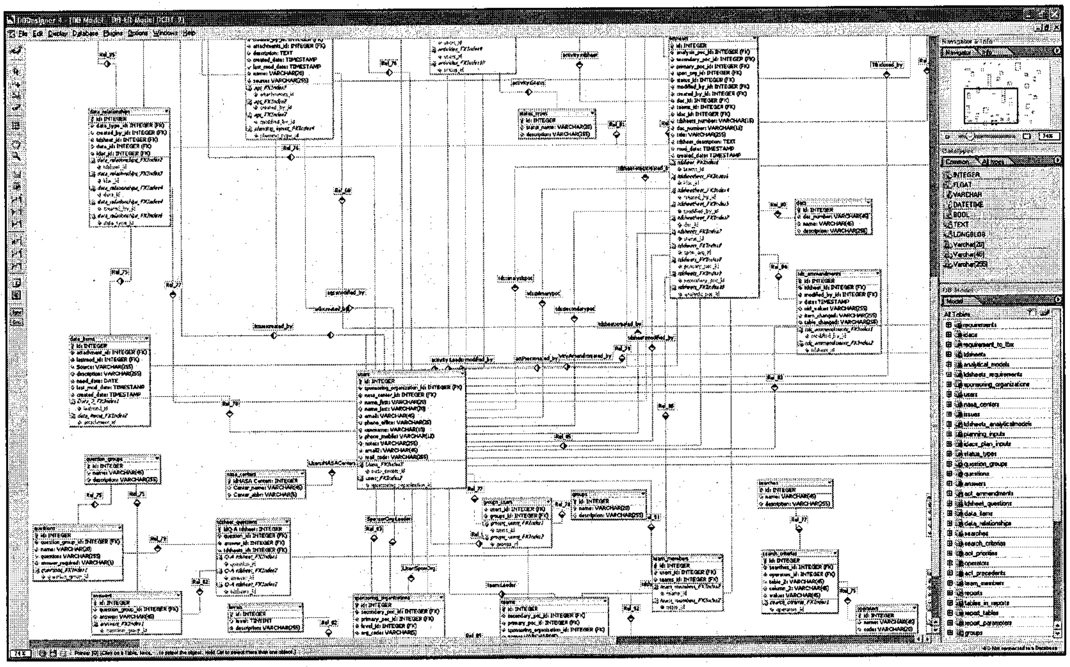


Figure 3: Screenshot of the CAIT ER Diagram in DB Designer 4

Solution Stack

Ruby is an interpreted, object oriented, scripting language created by Yukihiro Matsumoto.[2] According to the Pragmatic Programmer's Guide, Ruby follows the principle of least surprise, meaning that things work the way you expect with few exceptions. Proponents claim that Ruby programmers can write more code in one sitting, with seldom syntax errors, no type violations, and fewer bugs when compared to other languages. These claims are based on language features such as no end of line semi-colons, no type declarations to keep in sync, no compiler specific words, and no framework code.[3]

Rails, a development framework, enables production of database-backed web applications based on the Model-View-Controller (MVC) pattern.[4] The MVC pattern decouples the user interface, or view, from the data, or model. Separating the model and the view allows programmers to make changes in the front-end or back-end of the application without affecting the other end. Also, MVC separates business logic from data presentation. Controllers respond to events, typically user actions and might involve changes to the model.[5]

Mongrel is a fast HyperText Transfer Protocol (HTTP) library and server for Ruby that is intended for hosting Ruby web applications.[6] The Apache' web-server can be configured to forward all or some traffic to Mongrel. With the possibility of two or more users hitting the application server simultaneously, it is possible to create multiple Mongrel instances.[7] For a Ruby-on-Rails application, Mongrel serves the web-pages with embedded Ruby scripts.

Debian is a free operating system that uses a Linux kernel and basic tools from the GNU project. Started in 1993, the name Debian derives from the developer's name Ian Murdock and his wife Debra.[8] Debian has a world-wide development community ready and willing to assist developers. For example, Stephen Young discovered a bug in Debian that prevented a RubyGems library from working. Stephen determined the source of the bug a programmer in Japan who was willing to update the code base. Within two days, Stephen installed the updated operating system that enabled the RubyGems library to work.

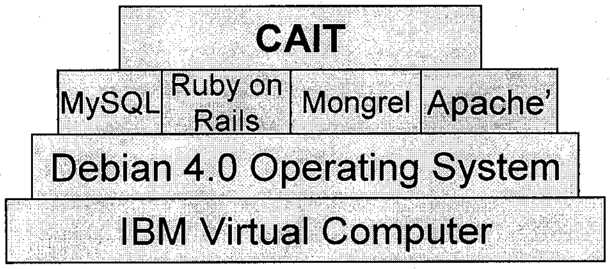


Figure 4: Solution Stack

Deployed on a virtual computer within an IBM mainframe, the CAIT database is built upon the previously describe systems. Figure 4 depicts this solution stack of the virtual computer, operating system, database, framework, application server, and web server.

A three stage system involved a development server, staging server, and production server. By making the development server publicly accessible, the development team provided a capability for the customer to interact with prototypes and provide comments. With the staging server identical to the production server, the system administrator applied operating system patches and determined how they affected the application without running the risk of interrupting the production system. After approval from the customer on code changes and successful tests of operating system updates, the system administrator deployed new versions of the application on the production server at the end of the workday.

Configuration Management

The development team chose Subversion (SVN) as the code repository server and Tortoise SVN as the client. Version control systems solve the problem of enabling users to share information and preventing them from overwriting one another's changes in the repository.[9] As a project, CAIT used a development repository and a production repository. Developers check new code into the development repository, where it is integrated with other new code. At the end of a development cycle, the code is promoted to testing- where integration testing is done, and then when ready, the code is promoted to production.

Experiences of the development team found that merging code between multiple branches to be a tedious, error prone, and time consuming process. To make better use of code branches, focus the code on a single task because merging code with unrelated changes complicates the process.

Code and Data Migration
With Ruby on Rails, developers can generate migration scripts for changing code and content. Benefits of migration scripts include updating code on multiple development systems, rolling out new baselines on a production system, and providing a roll-back capability for debugging.

Functional and Regression Testing
Ruby on Rails provides a capability to auto-generate code stubs for functional and regression tests. As the development team discovered bugs in the code, Clara Welch incorporated error checking code to look for the specific bug. As the team defined work-flows, Clara created regression tests that executed the code based on use-cases.

Elevating the Application to the Enterprise
Initial operations of the CAIT database began in the first week of December 2006. By February 2007, there existed a sizable user community and a substantial amount of content. Within the ESMD-DIO, Nantel Suzuki advocated the application of the database to manage task descriptions for studies sponsored by the enterprise. With an operational system and on-going implementation of Constellation requirements, rapid prototyping proved difficult. To define and incorporate new requirements identified by the DIO, Daniel O'Neil and Nantel Suzuki developed "As-Is" and "Go-To" screen-shots. PowerPoint presentations contained current screen-shots of the user interface and revised images that depicted new fields and recommended revisions to labels. After implementing changes depicted on the go-to screen-shots, Mitch Hunt and Dennis Bulgatz demonstrated the system to Nantel on the development server. Upon receiving approval of the revisions from Nantel, the development team deployed the new system.

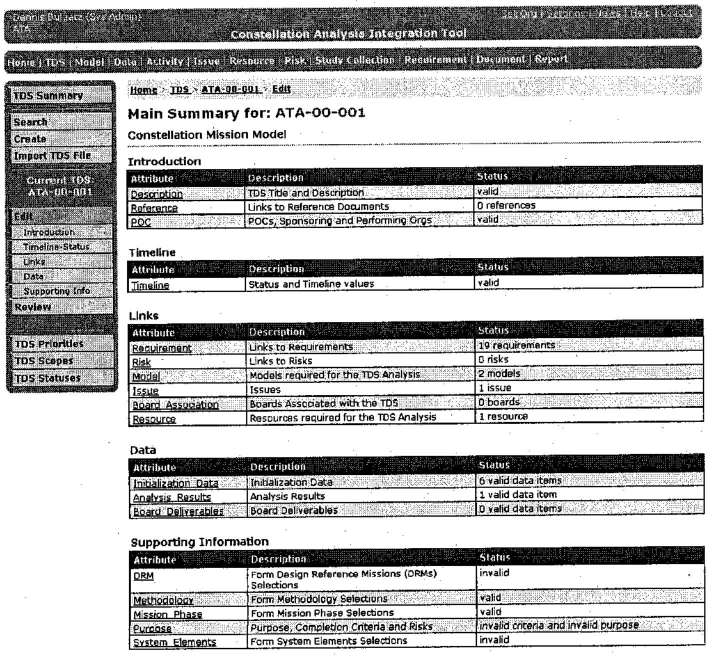
LESSONS LEARNED

Experiences associated with deployment of a web-based task management database have taught the team valuable lessons about requirements definition, system design, software development, configuration management, testing, deployment, and continuous improvement. Generically, these lessons sound familiar to systems engineers and project managers with experiences in software systems development. The project management and programming gems among these lessons pertain to the tools and techniques applied on this project or their recommended alternatives.

Requirements Definition
Originally, the development team began with a traditional requirements document with over 100 requirements. Contents of the document included a high level process description and detailed data tables. While this type of document provided a starting point for creating ER diagrams, it did not convey the business model, detailed work-flows or specifications for a graphical user interface. The development team found DB Designer 4 to be a fine ER diagramming tool but it did not capture the system behavior and interactions. Recommendations for requirements definition range from using a graphical notation like the Unified Modeling Language 2 (UML) to writing a user's manual complete with mock screen-shots. Developed by the Object Management Group (OMG) consortium, the UML offers 13 diagrams that depict the structure, behavior, and interactions of a system.[10] A user's manual and mock screen-shots focus on the step-by-step workflow of a user interacting with the system. Also, it serves system testing and verification because users can follow the manual and determine whether the deployed application matches the document.

System Design

Other than developing ER Diagrams in DB Designer 4, the design of the CAIT database evolved through rapid prototyping. The development team produced a version of the database and presented it to the ATA office. Based on consolidated comments provided by the voice of the customer, the team improved the system and conducted another demonstration. From October 2006 through February 2007, the team deployed one or two versions per week. By February, the system had several hundred registered users so the team was careful about making frequent major changes to the graphical user interface. Based on conversations with the end-users, the development team implemented user interface to improve clarity, navigation, searches, and announcements. Figure 5 presents an image of the summary screen for a TDS. Notice the "bread-crumb" trail above the title of the TDS, which indicates the location within the system. Highlighting in yellow indicates the active menu. On the left side, a vertical area presents buttons for the active menu. As the team implemented requested revisions, the customer viewed the application on the development server instead of the production server. This approach provided some rapid prototyping capability without stopping service on the production system.


Constellation Analysis Integration Tool
Home | TDS | Model | Data | Activity | Issue | Resource | Risk | Study Collection | Requirement | Document | Report
TDS Summary
Search
Create
Import TDS File
Current TDS: ATA-00-001
Edit
Introduction
Timeline-Status
Links
Data
Supporting Info
Review
TDS Priorities
TDS Scopes
TDS Statuses
Home > TDS > ATA-00-001 > Edit

Main Summary for: ATA-00-001

Constellation Mission Model

Introduction

Attribute	Description	Status
Description	TDS Title and Description	valid
Reference	Links to Reference Documents	0 references
POC	POCs, Sponsoring and Performing Orgs	valid

Timeline

Attribute	Description	Status
Timeline	Status and Timeline values	valid

Links

Attribute	Description	Status
Requirement	Links to Requirements	19 requirements
Risk	Links to Risks	0 risks
Model	Models required for the TDS Analysis	2 models
Issue	Issues	1 issue
Board Association	Boards Associated with the TDS	0 boards
Resource	Resources required for the TDS Analysis	1 resource

Data

Attribute	Description	Status
Initialization Data	Initialization Data	6 valid data items
Analysis Results	Analysis Results	1 valid data item
Board Deliverables	Board Deliverables	0 valid data items

Supporting Information

Attribute	Description	Status
DRM	Form Design Reference Missions (DRMs) Selections	invalid
Methodology	Form Methodology Selections	valid
Mission Phase	Form Mission Phase Selections	valid
Purpose	Purpose, Completion Criteria and Risks	invalid criteria and invalid purpose
System Elements	Form System Elements Selections	invalid

Figure 5: Screenshot of a TDS summary page

Rapid prototyping involves fast and furious programming. This approach can lead to a Mad Hatter's Tea Party where "there is never time to do the dishes." Automated document generation enables a team back-out a design from the code so documentation is not sacrificed for rapid prototyping. For MySQL and other databases, DB Designer 4 provides a reverse engineering capability.[11] For Ruby on Rails, there exists RDoc and RailRoad. With RDoc, programmers can generate HyperText Markup Language (HTML) documentation from Ruby source code.[12] With RailRoad, programmers can generate class diagrams from Ruby on Rails source code.[13]

Software Development

Between October 15th and December 7th, 2006, the CAIT development team learned RoR and developed several versions of the database and application. After deployment, the team continued improving the product and the associated development process. To update the database and roll the baseline code, the team learned a powerful technique known as migration scripts. As mentioned in an earlier section, migration scripts provide a capability to change schema and content of a database.

Code written for the CAIT database was developed without an Integrated Design Environment (IDE). Benefits of an IDE include text color coding, code completion, debugging, and statistics. The Rails IDE Minus Eclipse (RIDE-ME) is an open source, windows based environment designed specifically for RoR development.[14] A CodeSnippet manager in RIDE-ME support software reusability.

When this project started, roles and responsibilities were not assigned, which served as a source of frustration and overlapping effort. Eventually, everyone member discovered areas where they could contribute. Early in a project, it is a good idea to identify tools, techniques, and talents. For example, to use RDoc, developers must to embed formatted comments so the tool can generate HTML documents.

Configuration Management

A release manager merges of source code from multiple programmers. Responsibilities of this person involve establishing a process for receiving, accepting, integrating, and releasing code. A team should use one centralized configuration management system such as Subversion to check-in their code. Everyone on the team should understand and adhere to acceptance criteria such as formatted comments for document generation, results of functional tests, and comments about the changes. Integration or regression tests enable the release manager to determine whether the new baseline works and if any old bugs have crept back into the system. A release manager can set a cadence by developing a schedule for accepting code and delivering the baseline system to the deployment team.

Testing

A code generator, provided by RoR, enables development of functional tests. With functional tests, developers can determine how code will react over the range of inputs and check for previously discovered errors. Integration tests focus on the performance of the system to ensure robustness against any combination of user inputs. Applying the system requirement's use-case scenarios, integration tests can step through the user workflows. Writing test code requires the developers to think through inputs that are out of range or wrong type as well as the combinations of input data.

Deployment

Working with FOSS provided flexibility in deployment of the production sever and solution stack. At the beginning of the project, the team did not know that the production system would be a virtual machine within an IBM Mainframe. The Debian operating system worked well on both the desktop development server and the mainframe. As mentioned earlier, a bug discovered in the Debian operating system was corrected within two days. A significant change in the solution stack was switching from Apache's FastCGI to a Mongrel cluster. In both cases, the FOSS community provided technical support and needed documentation.

CONCLUSIONS

A summary of the lessons learned from deploying a web-based task management system include:

1) Define use cases and illustrate requirements with structural, behavior, and interaction diagrams
2) Apply open source systems such as Debian, Ruby on Rails, MySQL, Mongrel and Apache'
3) Ask for help from the open source community if problems arise in the solution stack
4) Assign roles and responsibilities for development, release management, and deployment
5) Document procedures for comments, migrations, testing, and configuration management
6) Write with an Integrated Development Environment and build a library of reusable code
7) Evolve the system design through rapid prototyping on a development server
8) Engage customers in the system design by asking for screen layouts and prototype critiques
9) Capture the system design with automated document and diagram generators
10) Develop functional tests along with the code and integration tests based on the use-cases

References

1. Manifesto for Agile Software Development, http://www.agilemanifesto.org/
2. Ruby Programming, http://en.wikibooks.org/wiki/Ruby_programming_language
3. Programming Ruby, http://www.ruby-doc.org/docs/ProgrammingRuby/
4. Ruby on Rails, http://www.rubyonrails.com/
5. Model-Viewer-Controller, http://en.wikipedia.org/wiki/MVC_Design_Pattern
6. Mongrel, http://mongrel.rubyforge.org/
7. Apache Best Practice Deployment, Charles Brian Quinn, http://mongrel.rubyforge.org/docs/apache.html
8. About Debian, http://www.debian.org/intro/about
9. Version Control with Subversion, Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato, http://svnbook.red-bean.com/nightly/en/svn-book.html#svn.basic
10. Introduction to OMG's Unified Modeling Language, http://www.omg.org/gettingstarted/what_is_uml.htm#12DiagramTypes
11. DBDesigner 4 Features, http://www.fabforce.net/dbdesigner4/features.php
12. RDoc Documentation from Ruby Source Files, http://rdoc.sourceforge.net/
13. RailRoad Ruby on Rails diagrams generator, http://railroad.rubyforge.org/
14. Riding Rails on Windows!, http://www.projectrideme.com/

MSFC-456-Abstract

"Lessons Learned from Deploying an Analytical Task Management Database."

Mr. Daniel A. O'Neil
NASA
Mr. Joshua Arceneaux
Booz Allen and Hamilton

Presentation Abstract

Large space exploration development programs involve geographically distributed teams conducting a wide variety of studies to mitigate risks and eliminate "To Be Determined" (TBD) from requirements. A web-accessible database to manage the task plans saves time and effort because everyone can contribute information about data products, analysis expertise, reviews, and task descriptions. Developing and deploying such a database involves integration of requirements from multiple offices at different layers of the organization, design of a schema and user interface that accommodates those requirements, and applying web-technologies. This presentation explains lessons learned from deploying a web-accessible task management database.

MSFC-456-Presentation

Lessons Learned from Deploying an Analytical Task Management Database

A Presentation to the
58th International Astronautical Congress

Mr. Daniel A. O'Neil
NASA

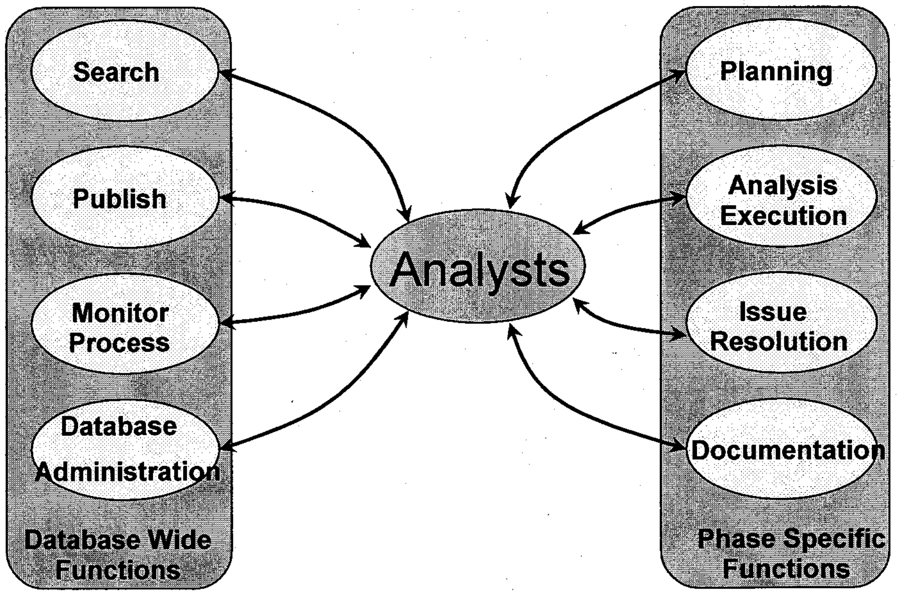
Mr. Joshua Arceneaux
Booz Allen and Hamilton

September 26th, 2007

Overview of the Constellation Analysis Integration Tool (CAIT) Database

- **Objectives**
  - Report the status of studies sponsored by NASA's Exploration Systems Mission Directorate (ESMD) via a password protected web-accessible database.
  - Integrate Design and Analysis Cycles (DAC) conducted by projects sponsored by the Constellation program.
  - Generate schedules from the data and relationships defined by the contents of the database.

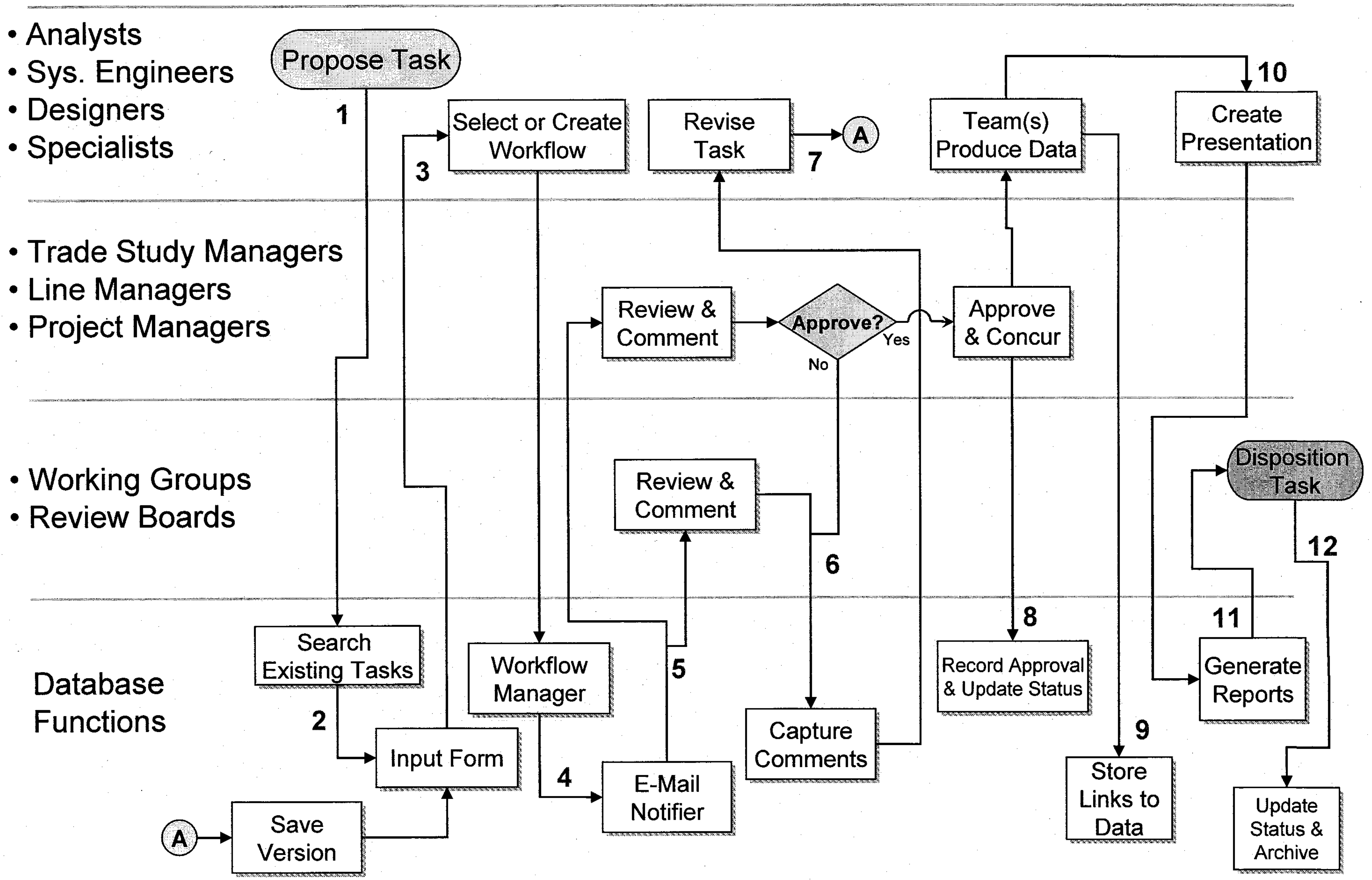
- **Functional Requirements**



Elements of a Task Description Sheet (TDS)

Introduction	• Description, Title, Points of Contact
Timeline	• Current Status, Projected Completion Date, Priority
Links to External Items	• Requirements (system and mission) • Risks (potential problems and impact) • Models (analytical tools used) • Issues (current problems) • Review Boards • Resources (analysts, facilities, etc.)
Analysis Data	• Inputs • Outputs • Board Deliverables
Supporting Information	• Design Reference Missions • Methodologies • Mission Phases • Purpose • System Elements

Overview of a Task Management Database Supported Process



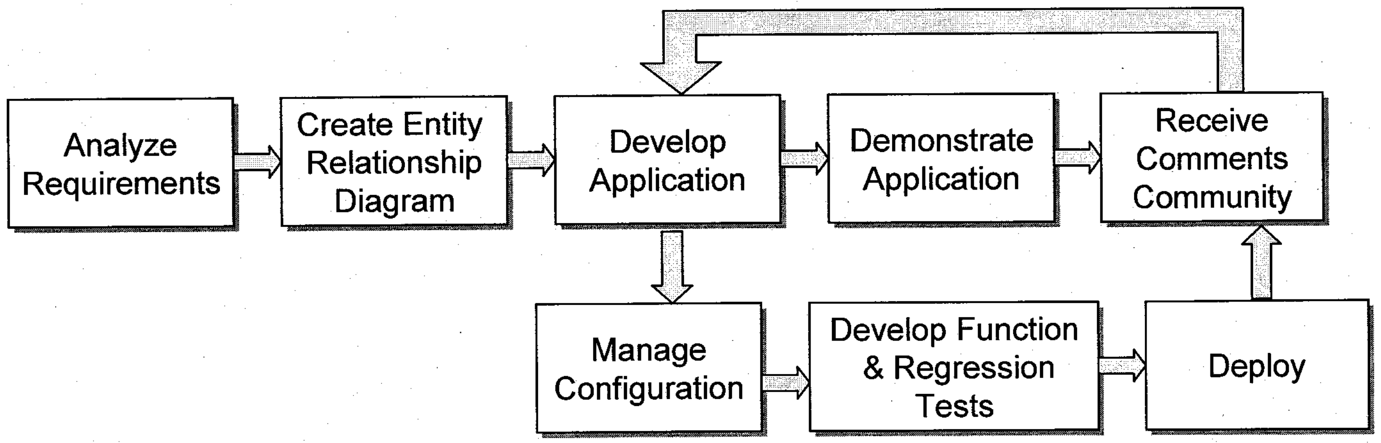
Development Philosophy and Process

Agile Software Development Manifesto

Individuals and interactions over processes and tools	Working software over comprehensive documentation
Customer collaboration over contract negotiation	Responding to change over following a plan

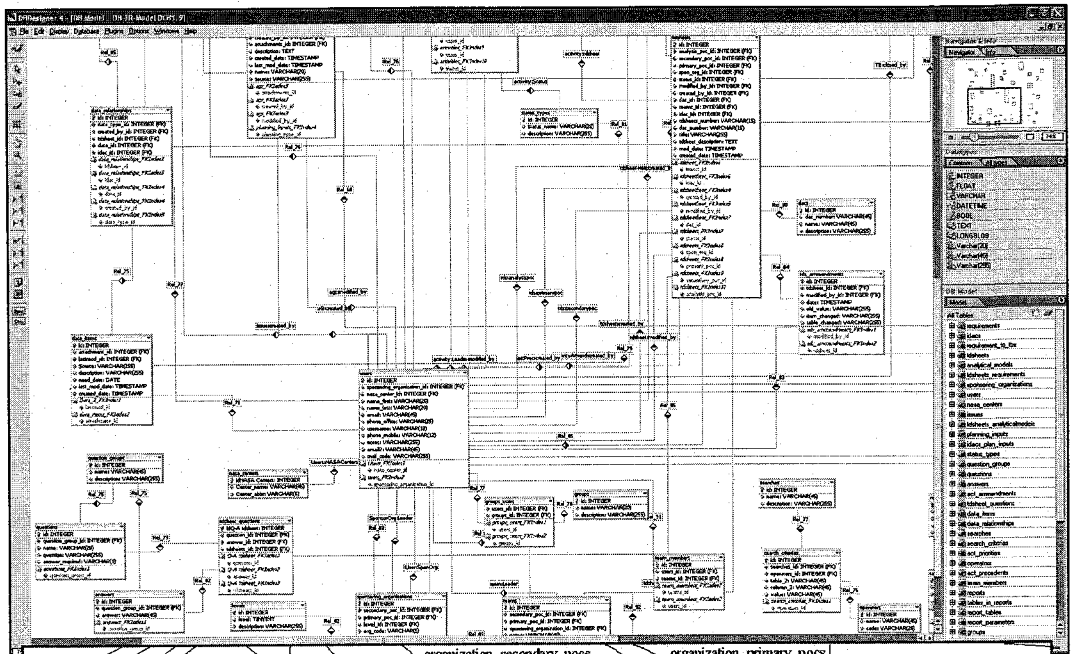
Rapid Prototyping

- Discussed requirements with "voice of the customer" and stake-holder representatives
- Quickly developed and demonstrated code via the Web and teleconferences
- Iterated the design of the graphical user interface and database reports based on comments
- Created "As-Is" and "Go-To" screen shots as designs to implement later requirements

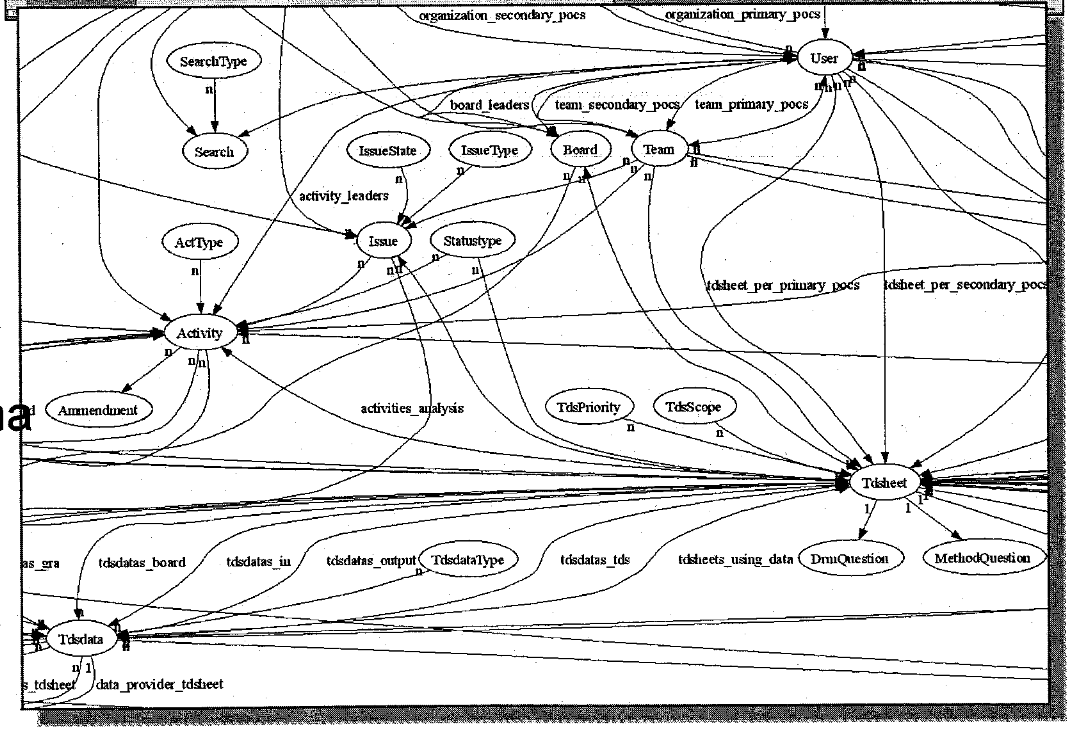

Analyze Requirements
Create Entity Relationship Diagram
Develop Application
Demonstrate Application
Receive Comments Community
Manage Configuration
Develop Function & Regression Tests
Deploy

Software Development Tools

- Notepad ++ 4.1.1 code editor
    - Ruby on Rails language and development framework
    - MySQL 5 for the database
    - Debian 4 for the operating system
    - Apache' 2 for the web server
    - Railroad 0.4 to generate Entity Relationship (ER) Diagrams
    - Graphviz 2.1.2 to generate a printable image of the ER diagram
    - DB Designer 4 to generate

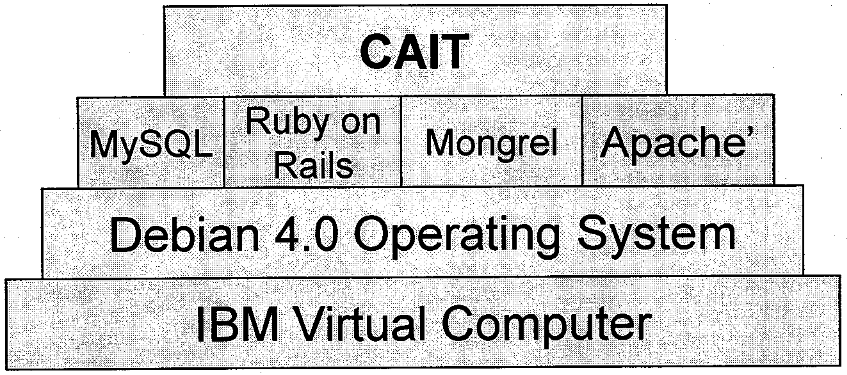


A screenshot of DB Designer 4 depicts the database schema.



Railroad and Graphviz were used to generate an Entity Relationship diagram.

Application Architecture

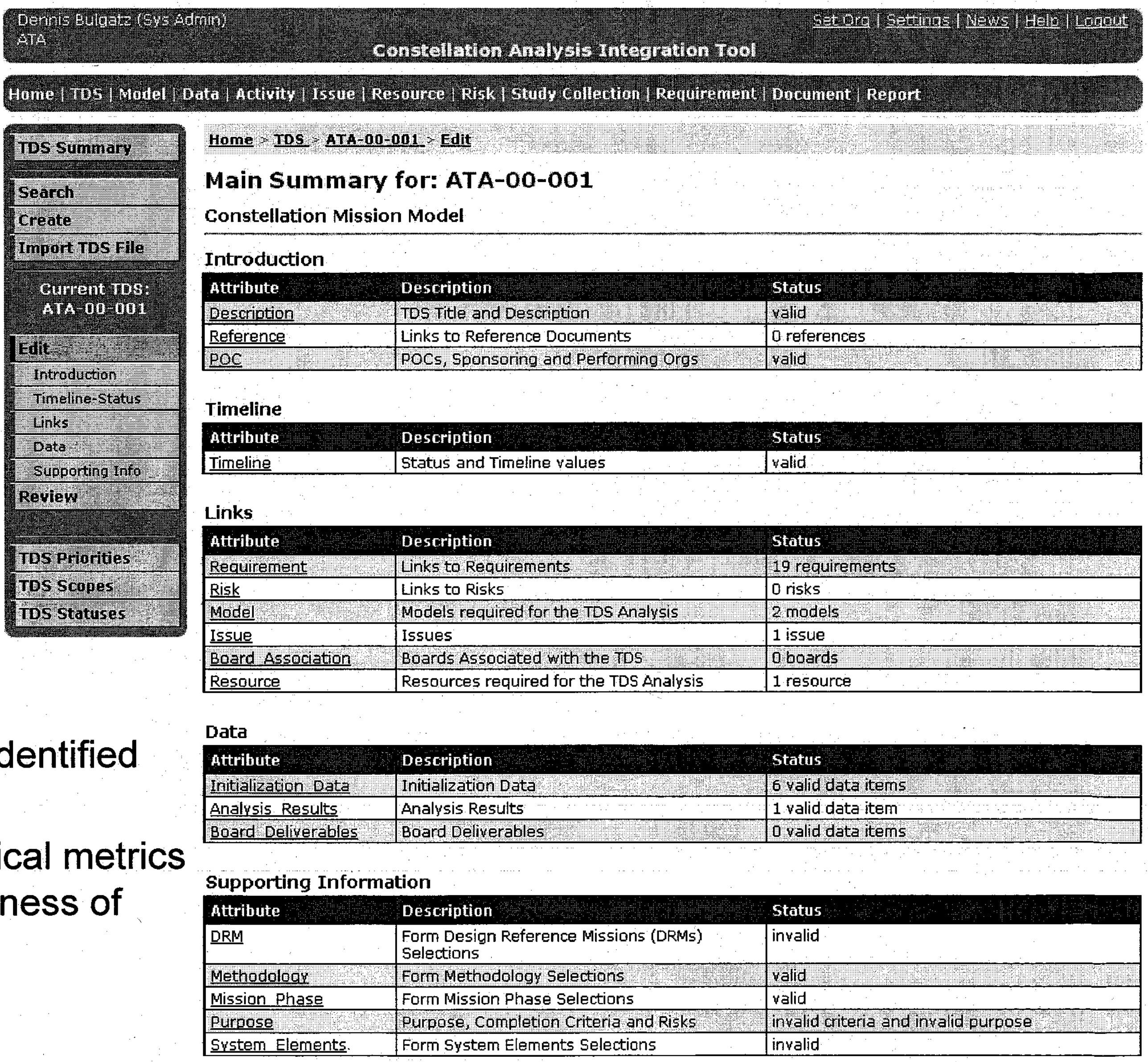


- The NASA Data Center (NDC) hosts an IBM Z Series Mainframe computer
- Two virtual computers within the mainframe serve as a test environment and a production environment
- Current operating system is Debian 4.0
- Planned migration to Suse Linux so we can access external data storage.
- MySQL 5 is the open source database
- Ruby on Rails is a free computing language and library for developing web applications
- Mongrel provides the capability to serve dynamically generated web pages.
- Apache' is the open source web-server that manages the application web site.
- The Constellation Analysis Integration Tool (CAIT) application is the Ruby-on-Rails, Javascript, and HyperText Markup Language (HTML) code.

Screenshot of TDS Status Page

Design Features

- Navigation bar with high-lighted text to indicate the current menu.
- A menu on the left side
- A "bread-crumb" trail to indicate the path to this screen
- A heading that identifies the type of information and the record number
- Links to "drill-down" to lower levels of detail
- Descriptions of attributes identified on the page.
- Status column with numerical metrics that indicate the completeness of the record



Dennis Bulgatz (Sys Admin)
ATA
Set Org | Settings | News | Help | Logout
Constellation Analysis Integration Tool
Home | TDS | Model | Data | Activity | Issue | Resource | Risk | Study Collection | Requirement | Document | Report
TDS Summary
Search
Create
Import TDS File
Current TDS: ATA-00-001
Edit
Introduction
Timeline-Status
Links
Data
Supporting Info
Review
TDS Priorities
TDS Scopes
TDS Statuses
Home > TDS > ATA-00-001 > Edit

Main Summary for: ATA-00-001

Constellation Mission Model

Introduction

Attribute	Description	Status
Description	TDS Title and Description	valid
Reference	Links to Reference Documents	0 references
POC	POCs, Sponsoring and Performing Orgs	valid

Timeline

Attribute	Description	Status
Timeline	Status and Timeline values	valid

Links

Attribute	Description	Status
Requirement	Links to Requirements	19 requirements
Risk	Links to Risks	0 risks
Model	Models required for the TDS Analysis	2 models
Issue	Issues	1 issue
Board_Association	Boards Associated with the TDS	0 boards
Resource	Resources required for the TDS Analysis	1 resource

Data

Attribute	Description	Status
Initialization_Data	Initialization Data	6 valid data items
Analysis_Results	Analysis Results	1 valid data item
Board_Deliverables	Board Deliverables	0 valid data items

Supporting Information

Attribute	Description	Status
DRM	Form Design Reference Missions (DRMs) Selections	invalid
Methodology	Form Methodology Selections	valid
Mission_Phase	Form Mission Phase Selections	valid
Purpose	Purpose, Completion Criteria and Risks	invalid criteria and invalid purpose
System_Elements	Form System Elements Selections	invalid

Lessons Learned

A summary of the lessons learned from deploying a web-based task management system include:

- Define use cases and illustrate requirements with structural, behavior, and interaction diagrams
- Apply open source systems such as Debian, Ruby on Rails, MySQL, Mongrel and Apache’
- Ask for help from the open source community if problems arise in the solution stack
- Assign roles and responsibilities for development, release management, and deployment
- Document procedures for comments, migrations, testing, and configuration management
- Write with an Integrated Development Environment and build a library of reusable code
- Evolve the system design through rapid prototyping on a development server
- Engage customers in the system design by asking for screen layouts and prototype critiques
- Capture the system design with automated document and diagram generators
- Develop functional tests along with the code and integration tests based on the use-cases