Automated Knowledge Base

A conceptual model is a model made of the composition of concepts, which are used to help people know, understand, or simulate a subject the model represents. Some models are physical objects; for example, a toy model which may be assembled, and may be made to work like the object it represents.

Automated Knowledge Base

Dynamic documentation of The Enterprise eCommerce System

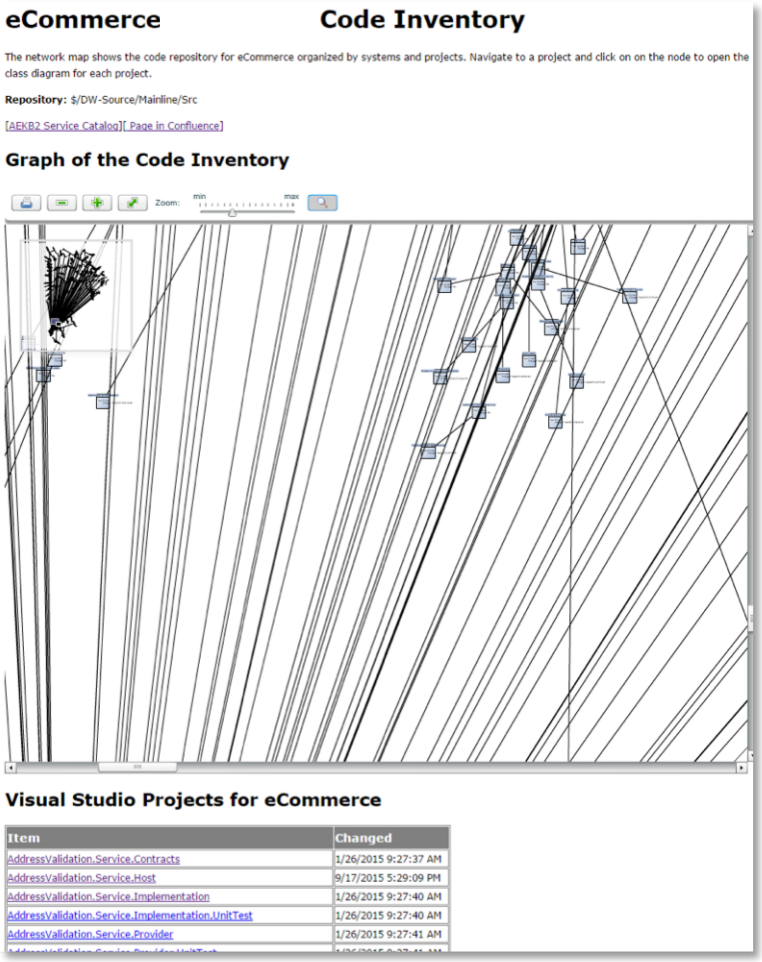
Opportunity

- The Enterprise would like to address the software knowledge base difficulties that they are experiencing.
  - It is difficult to find the correct answer within the three knowledge base tools
  - Teams depend on “tribal knowledge” to figure out answers to questions
  - The processes are not currently documented
  - Current process takes from 3 minutes to several days to find solutions
- This project will accomplish the following:
  - Assess the current process and document the process
  - Develop recommendations for new knowledge base policies
  - Reduce Mean Time to Resolve (MTTR) by 20%

Automation

- A manually authored knowledge base produced the following difficulties:
  - Time consuming to author content (an API takes 3-6 weeks).
  - Tech reviews with technical writers randomizes development teams at a critical point in the SDLC.
  - Update process is dependent on a culture of review and updating inaccurate information.
  - Amount of information is vast (1,200 projects in 136 collections for a single code repository), thousands of hosts.
  - System is constantly being changed as the old system is de-coupled and smaller services are launched, and in turn these smaller services experience faster iteration times.
- This project will accomplish the following:
  - Near instant code-first documentation will reflect against code and map objects in the environment and assemble UML documentation.
  - Truth will be located in the code and environment; documentation will be a map. The map reflects the truth.
  - Updates will require changing build processes and targets rather than "output."
  - Vast amount information will have automatic summaries, glosses, and "zoomed out" views.
  - Identity dependencies for a fully-automated solution.
  - Identify challenges and solutions for an automated system.
  - Create a proof-of-concept to demonstrate the approach.

the Enterprise eCommerce System Property Graph Overview



The system is a map of the behavioral and physical environment of importance to eCommerce. It includes structural models reflecting enterprise elements such as the code repository. The code repository is modeled from, top down: projects, to namespaces, to individual classes. Other mapped structures include data centers, hosts, databases, and applications. In addition, the model reflects behavioral elements such as transactions and user interactions.

As a map it is derived from concrete ties to elements in the O Environment. It uses a code-first paradigm rather than depending on design or planning documentation. The tool models the "as is" environment and retains a history of the "as was" environment.

A conceptual model is a model made of the composition of concepts, which are used to help people know, understand, or simulate a subject the model represents. Some models are physical objects; for example, a plastic toy model which may be assembled, and may be made to work like the object it represents. This model is rendered as a property graph rather than in plastic.

The Enterprise eCommerce System property graph is made accessible to eCommerce support staff and stakeholders through various views of the data rendered as diagrams and tables. The graph can answer common questions such as application dependency, where you can find the source code for a particular application, which aspects of the structural environment have changed from release to release.

In addition, the AEKB2 provides a scalable platform for creating new documents, reports, and sources of insight into the eCommerce platform. Analysts can create ad-hoc queries and add documents to the set of diagrams produced by the tool.

Automated Knowledge Base

SOME STORIES

Story 1

The user opens the site as one possible entry point to the Knowledge Base. There is an issue with an element on accountsummary.aspx. They access the tool, and the tool shows them the current state of the page, and what has changed from the last version since the tool interacts with the DOM (document object model).

They open the tool and it will show them a class diagram of the DOM, information about the page, a list of interactive elements, events, and service calls.

The page is a node in the larger graph model of the enterprise. They can navigate a network diagram of interaction flows. Or they can click a service end point and open an activity diagram representing a data flow diagram.

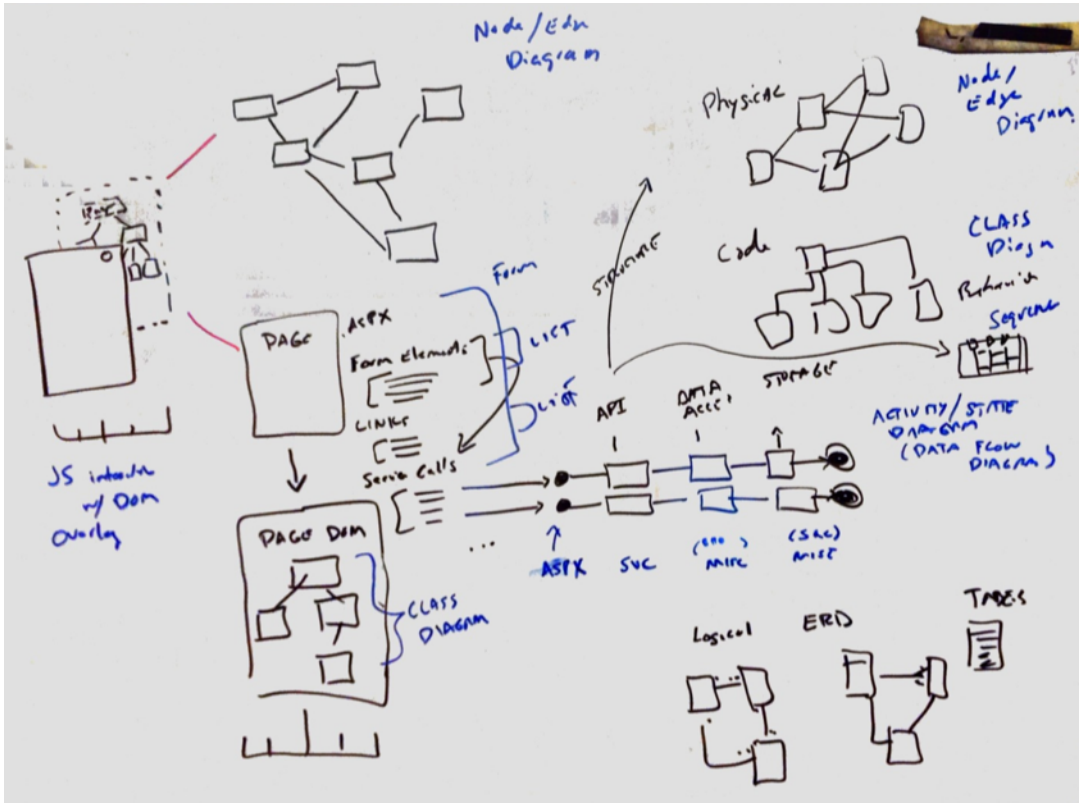
They can then click on the element and it will take them to a page view that shows the page in relation to a dataflow diagram. At each step they can move through the data flow to the API layer, to the Data Abstraction Layer, to the Physical Data Layer and see the specific path they are on, and also the context.

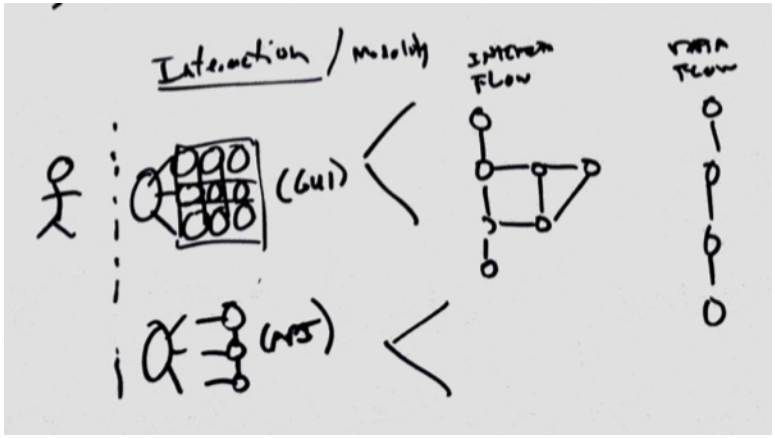
On the API layer, they can switch to a physical (component, host) view, or code view (application, solutions, namespaces, etc.).

Say the API layer is the one that is giving them trouble, t hey can then look at the class diagram of the application, and see the items that have changed.

They can then track down the changed code and isolate the problem to a specific class, and open the code reference page.

They can then retrieve the appropriate code repo location.



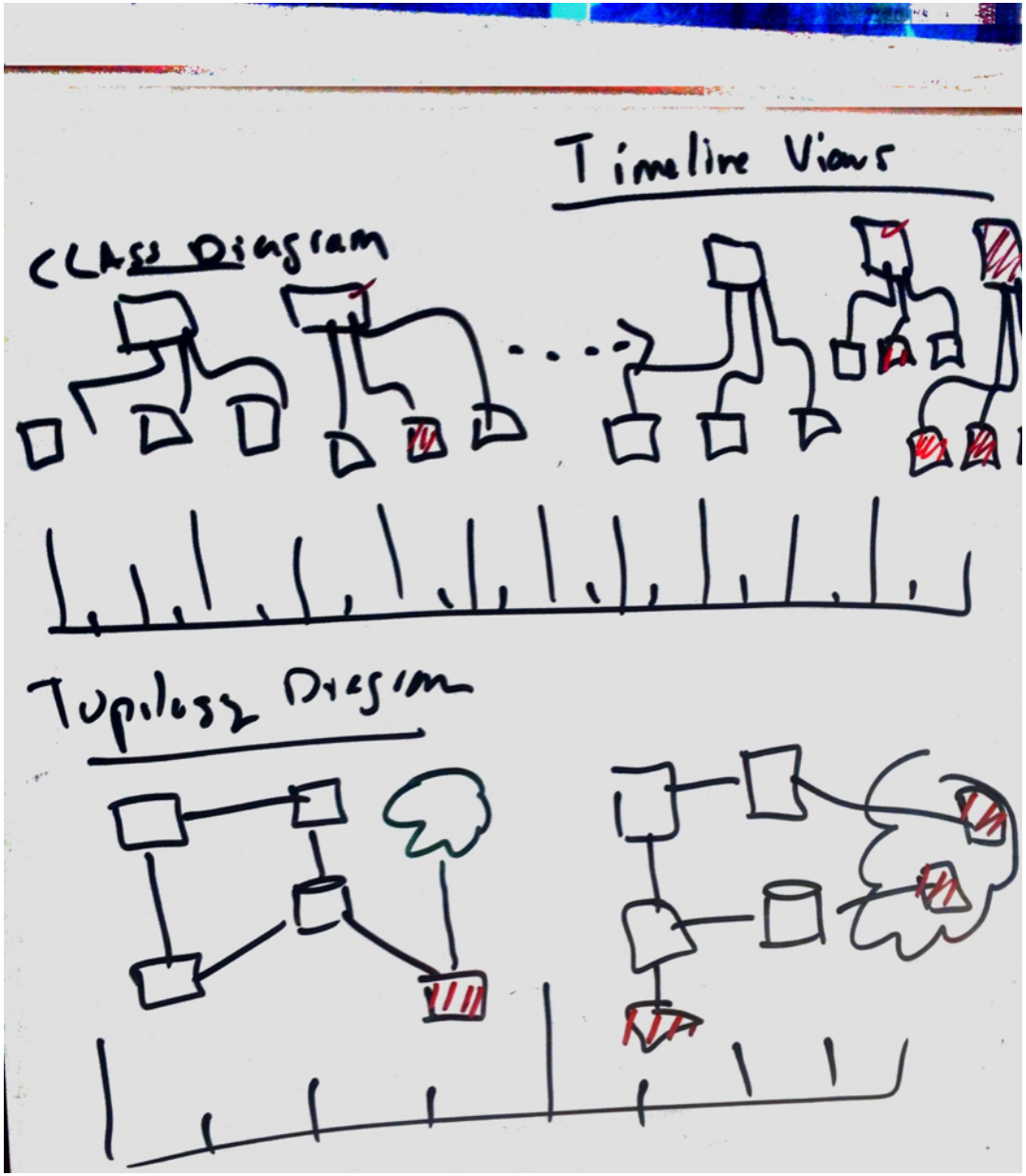


Story 2

You are new developer who may be working on a specific service. You can open the high-level context diagram of the system, and then drill down to the system you will be working on. You can read about each of the services that make up the applications that make up the system, review the systems that border the system by moving back up a level and looking at the various related systems.

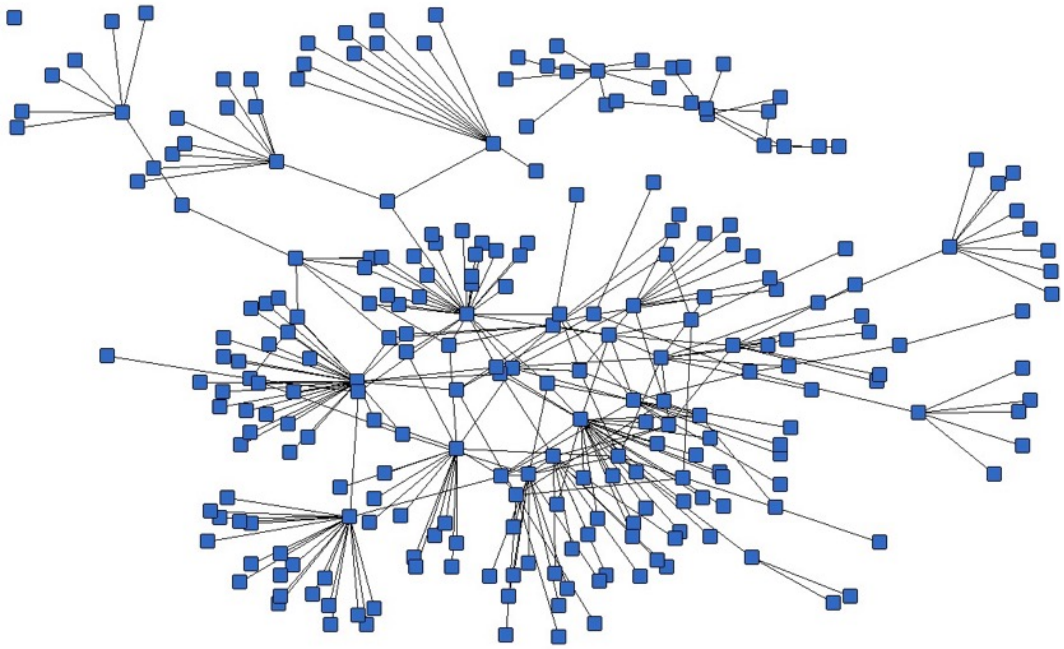
You can access the managed entities associated with the space. You can see how the code has changed over the past year, and you can see some of the planned future state as well including the work that you will be doing.

And then you can drill down to the namespace and set of classes that you will be working on and then review the code.



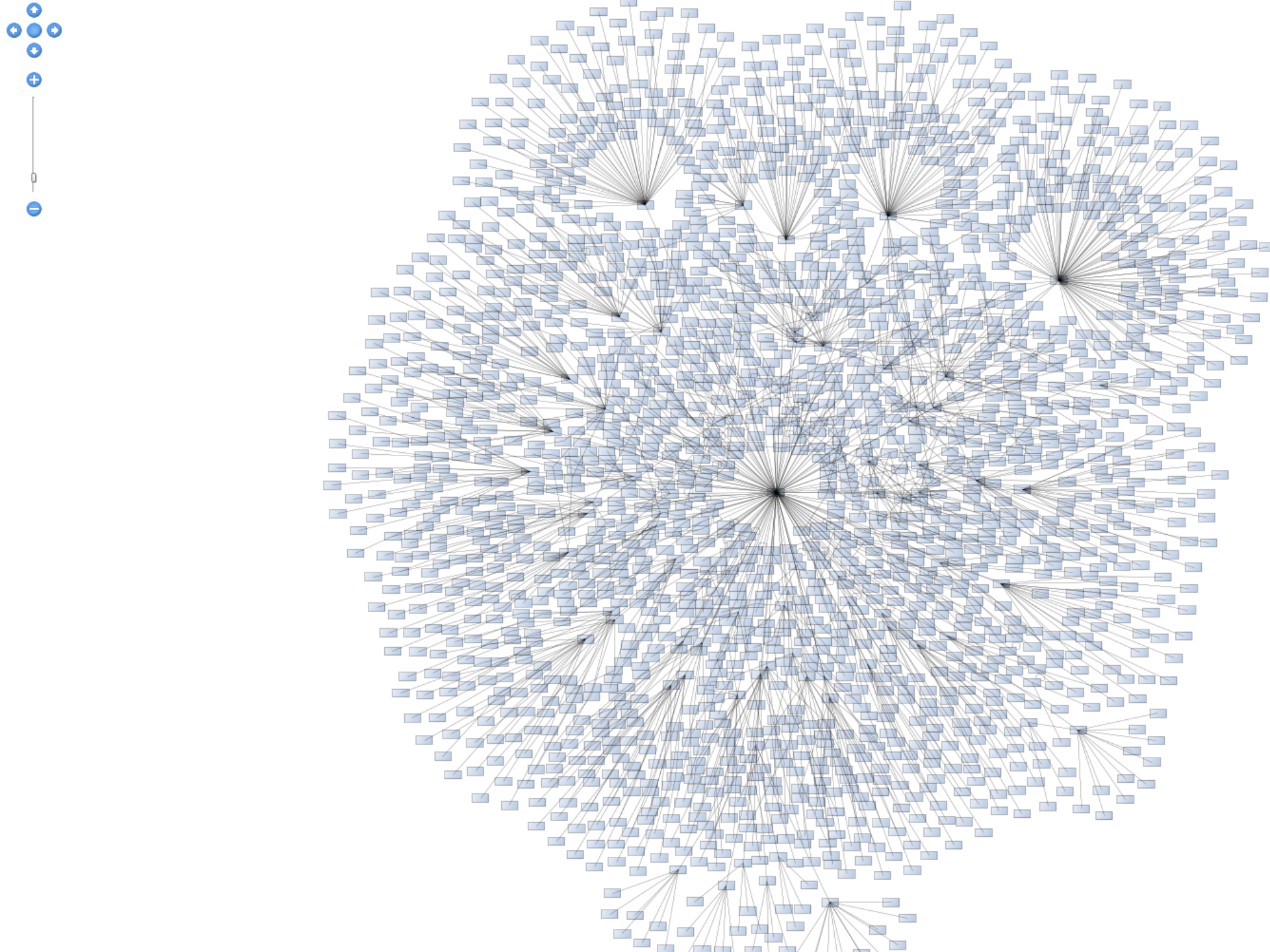
Story 3

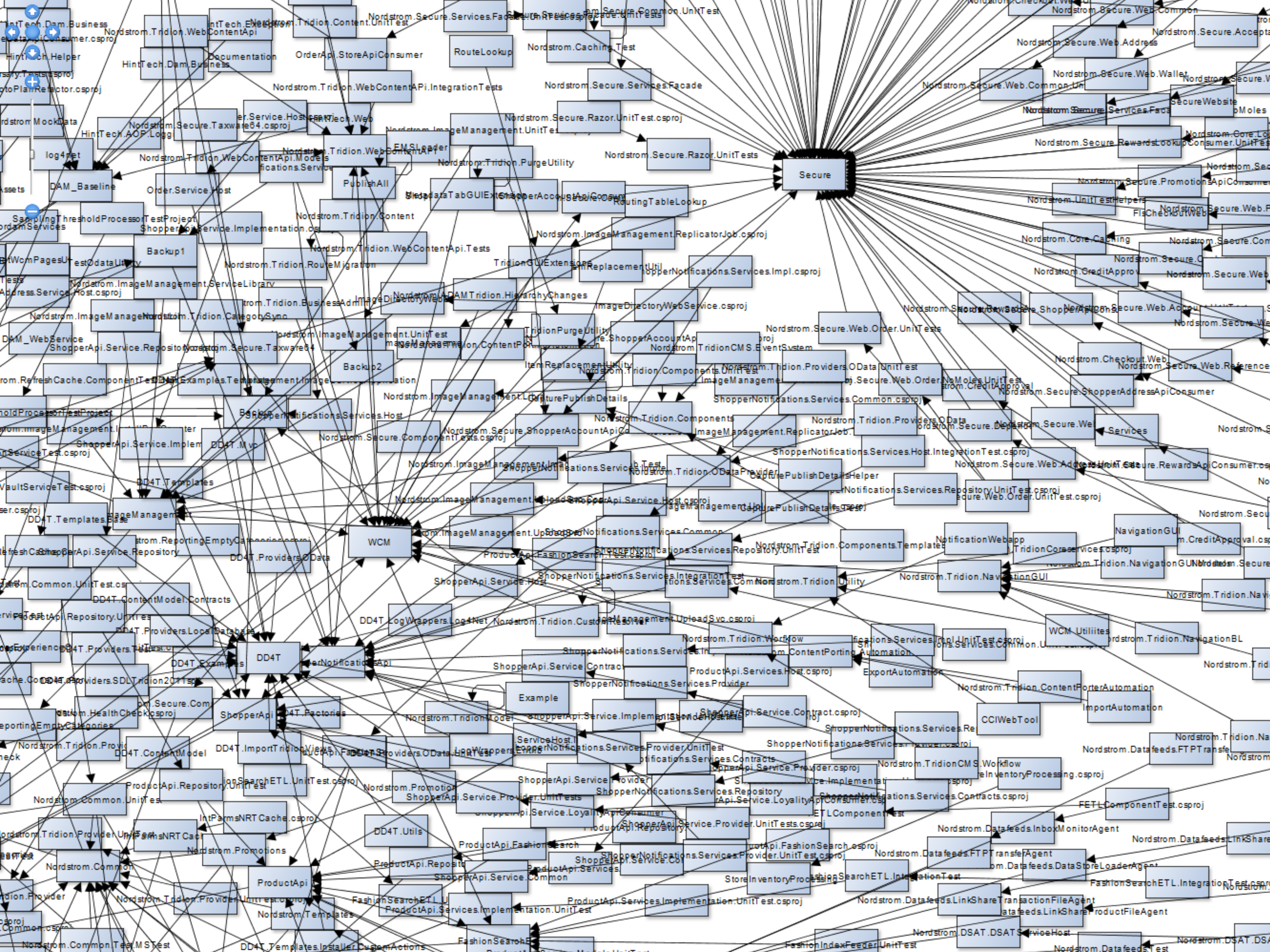
You are a tester looking at a service. You can explore the context of the service, story 3, and then you can review the interaction maps of the pages and determine the critical paths that will need to be tested as you create your scenarios.



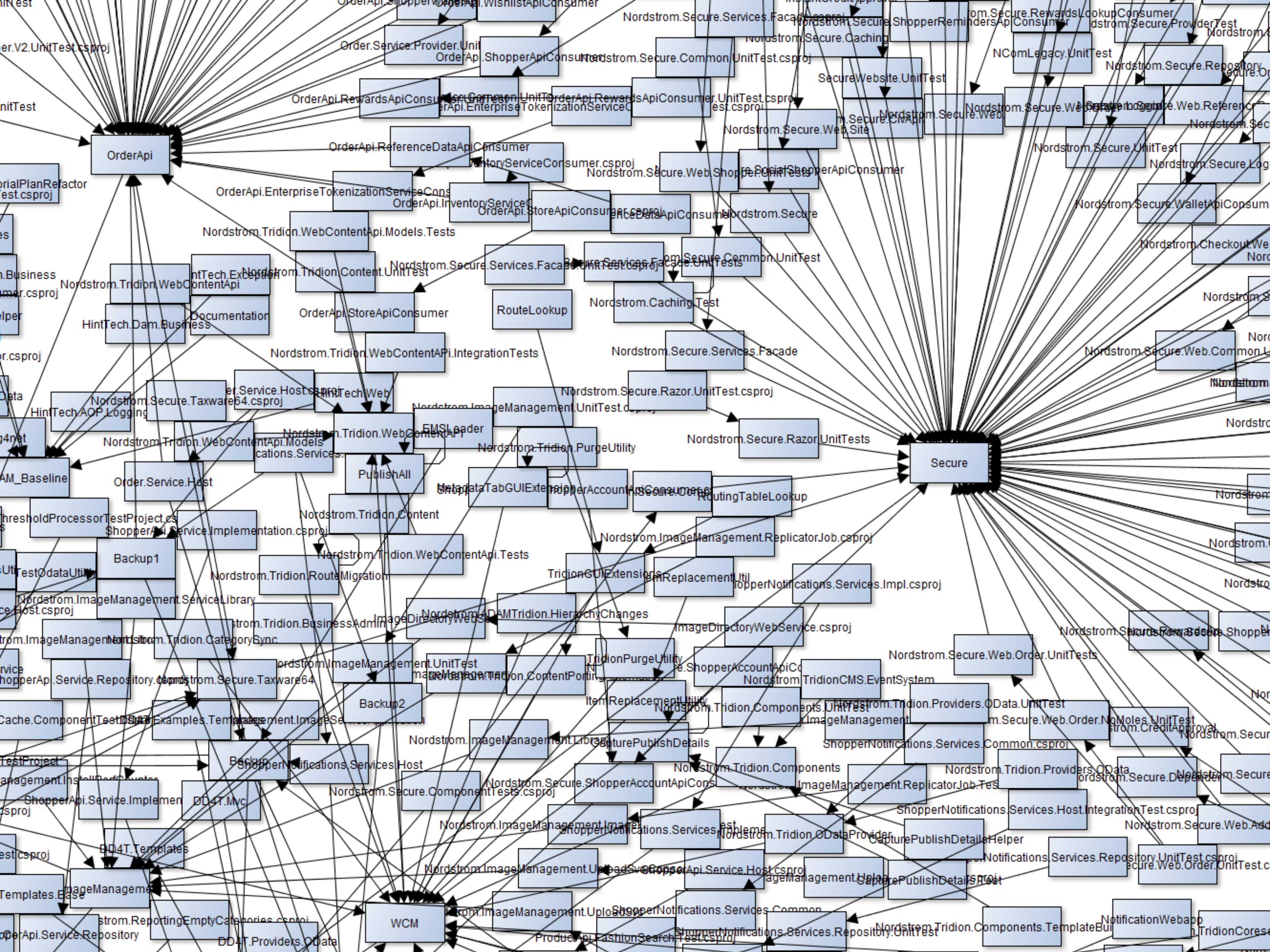
Automated Knowledge Base

THE PROBLEM





Secure
WCM
DD4T
ShopperApi
ProductApi
Nordstrom.Common
RouteLookup
OrderApi.StoreApiConsumer
Nordstrom.Caching.Test
Nordstrom.Secure.Services.Facade
Nordstrom.Secure.Razor.UnitTest.csproj
Nordstrom.Secure.Razor.UnitTests
Nordstrom.Tridion.WebContentAPi.IntegrationTests
Nordstrom.Tridion.PurgeUtility
Nordstrom.Tridion.Content
Nordstrom.Tridion.WebContentApi.Tests
Nordstrom.ImageManagement.ReplicatorJob.csproj
Nordstrom.Tridion.RouteMigration
Nordstrom.ImageManagement.ServiceLibrary
Nordstrom.Tridion.CategorySync
Nordstrom.Secure.Taxware64
ImageDirectoryWebService.csproj
Nordstrom.Secure.Web.Order.UnitTests
Nordstrom.Tridion.Components
ShopperNotifications.Services.Common.csproj
ShopperNotifications.Services.Host.IntegrationTest.csproj
Nordstrom.Tridion.Components.Templates
NotificationWebapp
NavigationGUI
Nordstrom.Tridion.NavigationGUI
Nordstrom.Tridion.Utility
Nordstrom.Tridion.Workflow
Nordstrom.Tridion.NavigationBL
WCM_Utiliites
ExportAutomation
Nordstrom.Tridion.ContentPorterAutomation
ImportAutomation
CCIWebTool
Example
Nordstrom.Datafeeds.InboxMonitorAgent
FETLComponentTest.csproj
Nordstrom.Datafeeds.FTPTransferAgent
Nordstrom.Datafeeds.LinkShareTransactionFileAgent
StoreInventoryProcessing
Nordstrom.DSAT.DSATServiceHost
FashionIndexFeeder.UnitTest
Nordstrom.Checkout.Web
Nordstrom.Secure.ShopperAddressApiConsumer
Nordstrom.Core.Caching
Nordstrom.Secure.Web.Wallet
Nordstrom.Secure.Web.Address
SecureWebsite
Nordstrom.Secure.PromotionsApiConsumer
DD4T.Templates
DD4T.Templates.Base
DD4T.ContentModel.Contracts
DD4T.Providers.LocalDatabase
DD4T.Examples
DD4T.Factories
DD4T.ContentModel
DD4T.Utils
DD4T.Providers.OData
DD4T.LogWrappers.Log4Net
DD4T.Templates.Installer.CustomActions
ShopperApi.Service.Repository
ShopperApi.Service.Host
ShopperApi.Service.Common
ShopperApi.Service.Provider.UnitTests
ProductApi.Services.Implementation.UnitTest
ProductApi.FashionSearch
ProductApi.Repository.UnitTest
Nordstrom.Promotions
Nordstrom.Templates
Nordstrom.Common.UnitTest
Nordstrom.Common.TestMSTest
IntParmsNRTCache.csproj
HintTech.Dam.Business
HintTech.AOP.Logging
Documentation
DAM_Baseline
DAM_WebService
Backup1
Backup2
PublishAll
log4net
Order.Service.Host
Nordstrom.Tridion.CustomResolver
Nordstrom.Tridion.Model


OrderApi
Secure
WCM
RouteLookup
PublishAll
Backup1
Backup2
Backup
DD4T.Mvc
DD4T.Templates
DD4T.Providers.OData
RoutingTableLookup
TridionGUIExtensions
TridionPurgeUtility
EMSLoader
Documentation
HintTech.Dam.Business
HintTech.AOP.Logging
Order.Service.Host
NotificationWebapp
OrderApi.ShopperApiConsumer
OrderApi.RewardsApiConsumer.UnitTest
OrderApi.ReferenceDataApiConsumer
OrderApi.StoreApiConsumer
OrderApi.Service.Repository
Nordstrom.Secure.Common.UnitTest.csproj
SecureWebsite.UnitTest
NComLegacy.UnitTest
Nordstrom.Secure.Caching
Nordstrom.Secure.ProviderTest
Nordstrom.Secure.Web.Site
Nordstrom.Secure.UnitTest
Nordstrom.Secure
Nordstrom.Secure.Services.Facade
Nordstrom.Caching.Test
Nordstrom.Secure.Razor.UnitTest.csproj
Nordstrom.Secure.Razor.UnitTests
Nordstrom.Secure.Web.Order.UnitTests
Nordstrom.Secure.Taxware64
Nordstrom.Secure.ComponentTests.csproj
Nordstrom.Tridion.WebContentApi.Models.Tests
Nordstrom.Tridion.WebContentApi
Nordstrom.Tridion.Content.UnitTest
Nordstrom.Tridion.WebContentAPi.IntegrationTests
Nordstrom.Tridion.WebContentAPI
Nordstrom.Tridion.PurgeUtility
Nordstrom.Tridion.Content
Nordstrom.Tridion.WebContentApi.Tests
Nordstrom.Tridion.RouteMigration
Nordstrom.Tridion.Components
Nordstrom.Tridion.ODataProvider
Nordstrom.Tridion.Providers.OData
Nordstrom.Tridion.Providers.OData.UnitTest
Nordstrom.TridionCMS.EventSystem
Nordstrom.ADAMTridion.HierarchyChanges
Nordstrom.ImageManagement.ReplicatorJob.csproj
Nordstrom.ImageManagement.ServiceLibrary
Nordstrom.ImageManagement.Library
ImageDirectoryWebService.csproj
CapturePublishDetails
CapturePublishDetailsHelper
ItemReplacementUtility
ShopperApi.Service.Implementation.csproj
ShopperNotifications.Services.Host
ShopperNotifications.Services.Common.csproj
ShopperNotifications.Services.Host.IntegrationTest.csproj

Current Problem

The Enterprise eCommerce lacks a discoverable description of the technical infrastructure.

Current Problem

(rephrased)

The Enterprise eCommerce lacks a current, cohesive, and managed collection of described objects and their relationships.

Aspects of the Problem

The Enterprise eCommerce lacks a cohesive and managed collection of objects and relationships.

- **Rate of change**
  The environment is in constant flux.
- **Organic accumulation**
  Objects may or may not have owners and a place. It is more like an attic than a library.
- **Complexity**
  The system is complex; yet the system is maintained by humans who cannot understand complexity.
- **Increasing levels of entropy**
  Entropy is a byproduct of complexity.
- **Heterogeneous**
  Code is moving from a single repo to multiple repos. Code is moving from ASP.NET (C# on SQL) to a polyglot code base (C#, JS, NodeJS, Java, etc) and polyglot datastores (Solr, MongoDB, SQL, etc.)
- **Turn-Over**
  Tribal knowledge is present because it works for members of the tribe, but does not work for the enterprise (which is not a tribe.)

Rate of change

Problem

- The environment is in constant flux.

Solution

- Capture dataflow (objects) rather than focusing on particular data (instances)

Organic accumulation

Problem

- Objects may or may not have owners and a place. It is more like an attic than a library.

Solution

- Flexible, schema-less tagging of all objects. All objects have a GUID and a locator.

Complexity

Problem

- The system is complex, yet the system is maintained by humans who cannot understand complexity.

Solution

- Capture detail at a 1:1 correspondence with the ability to *collapse* or *condense* information. This may be seen as the ability to zoom in and out of detail and the ability to find your way in the detail.

Increasing levels of entropy

Problem

- Entropy is a byproduct of complexity.

- 2nd law of thermodynamics:
  A system will spontaneously proceed towards thermodynamic equilibrium, the configuration with maximum entropy.

Solution

- Embrace granularity in the system such that complexity is product of small, easily understood (and managed) parts.

Heterogeneous

Problem

- Code is moving from a single repo to multiple repos. Code is moving from ASP.NET (C# on SQL) to a polyglot code base (C#, JS, Java, etc) and from SQL-only to polyglot data stores (document stores, Solr, MongoDB, SQL, etc.)

Solution

- Focus on the organic points of control in the system. Code must compile. Web sites must render. Scrums must burn down.

Turn-Over

Problem

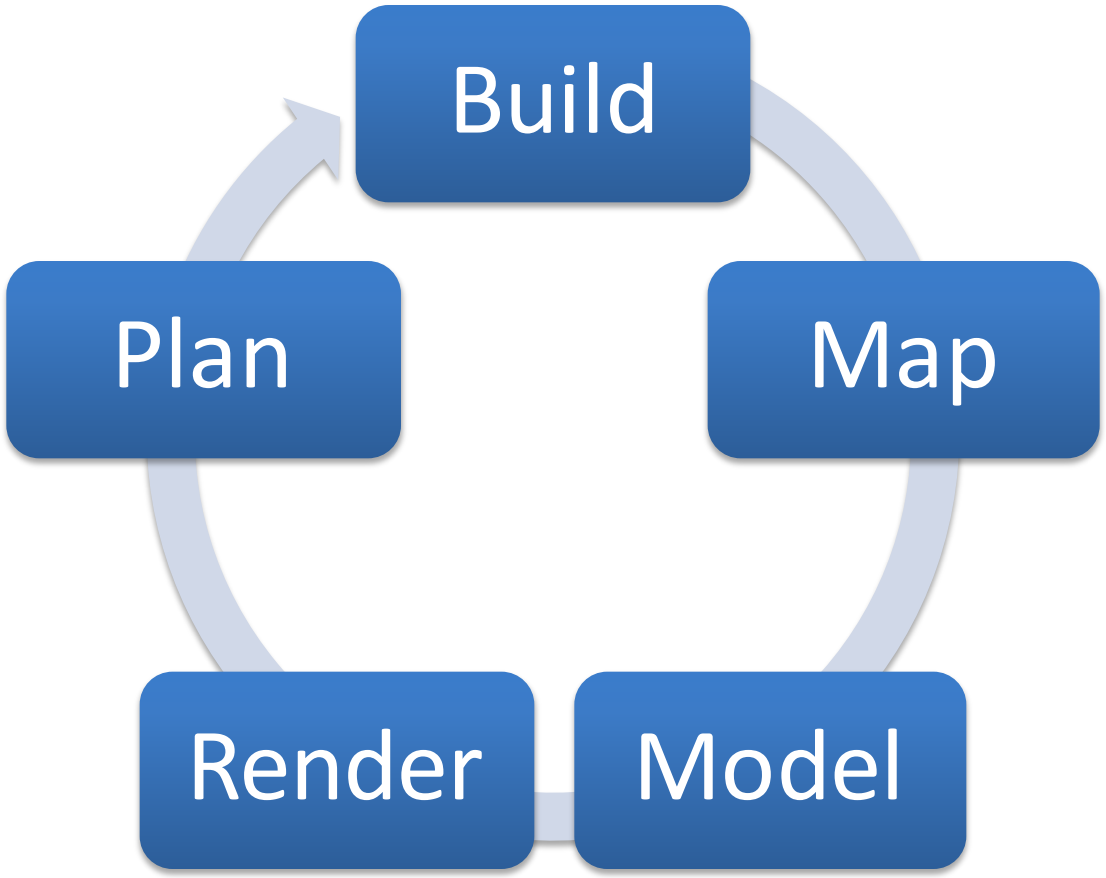
- Tribal knowledge is present because it works; yet it does not work for the enterprise when the tribal members leave the building.

Solution

- Embed knowledge in artifacts associated with the work from the POV of each member of the tribe; make knowledge a product of work as much as functioning code.

Automated Knowledge Base

SOLUTION



The Solution

Mappers and Monitors

Data Model

Views associated with context

Solution Overview

For one flow:

1. Develop the Code
2. Build
3. Refractor (export an XML representation)
4. Place into a model
5. Create a Class Diagram

- **Map and Monitor**
  Map objects, transactions, and events in the environment by tagging them with a GUID and a locator.

- **Inventory**
  Inventory the objects and add definitions to them in plain language.

- **Manage Relationships**
  Map the relationship between objects in the environment. Relationships can be determined by transactions and "contents" (that is a thing is contained in another thing.)

- **Model**
  Ingest this dynamic data and store in a model. The model at this level has three purposes: capture the data, capture change, and enable meaningful views of the environment.

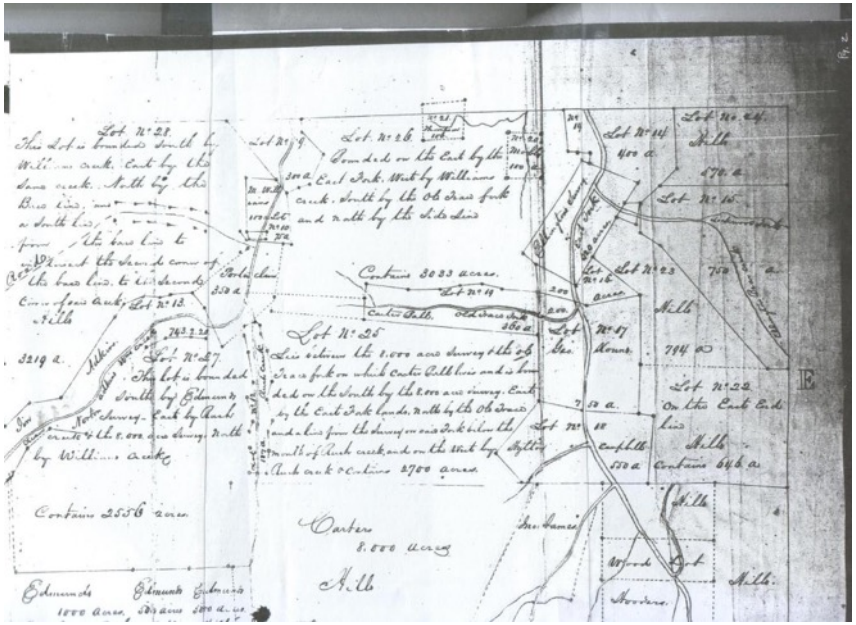
- **Render**
  Create a rendering logic that takes the object, relationships, and definitions and produces human readable documents at comprehensible-levels of abstraction.

Principles of the Solution

- Everything is a data flow (objects, events, transactions, states, and schemas)
- Data flows must be readily exchanged and structured at the lowest possible level. (XML or JSON are ideal through RESTful APIs)
- This level is a granular. A granule is unique and programmatically accessible. (key/value)
- Schemas must be abstracted from the dataflow granules.
- Pay attention to levels of abstraction: that is maintain a 1:1 correspondence with the referent, but allow then for detail to "roll up."
- A 1:1 correspondence allows for granules to be located by users.

The Solution

- **Mappers and Monitors**
- Data Model
- Views associated with context

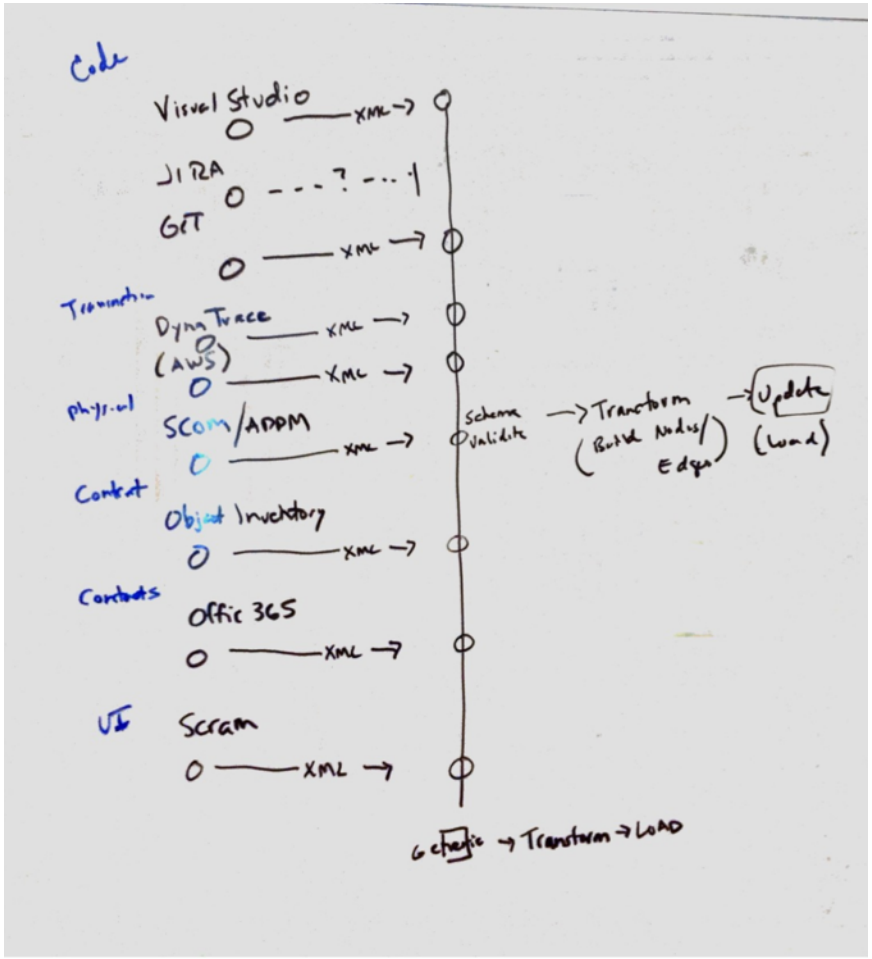


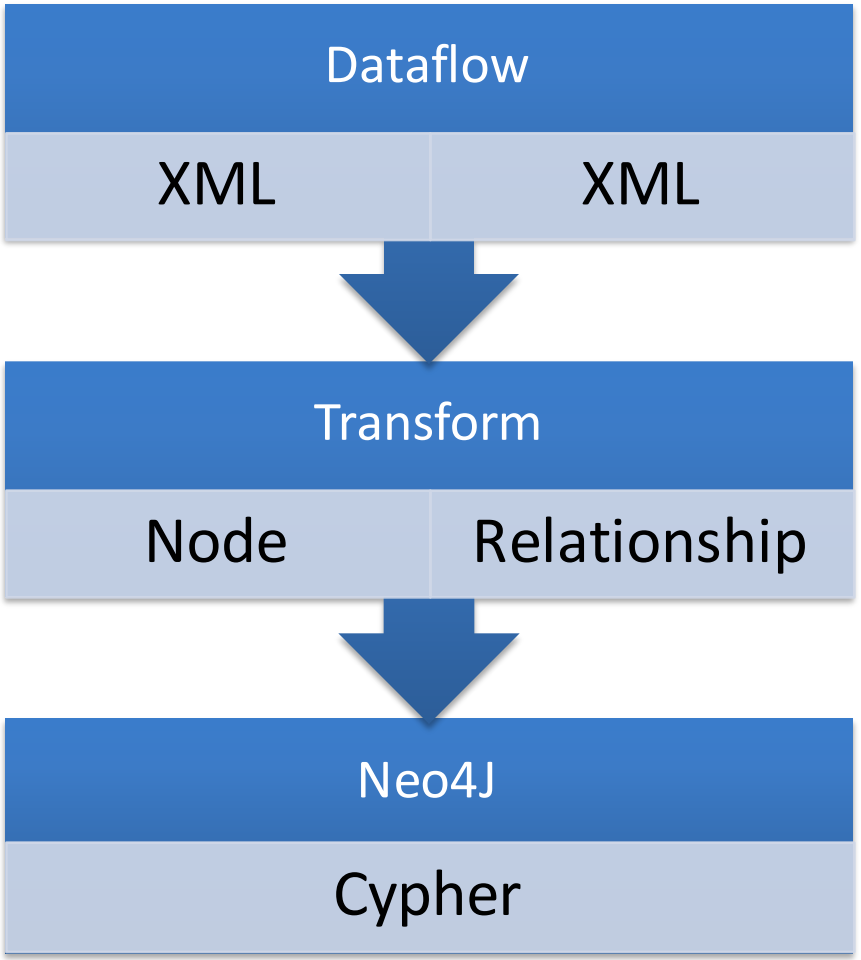
Mappers and Monitors

Element	Product	Output	Endpoint
Code (<namespace)	VSDocman	XML	File
Code (>namespace)	Repo to XML**	XML	File
Transactions	dynaTrace*	XML	API
Hosts	SCOM/ADDM	XML	API
Databases	Enterprise Architect	XML	File
Inventory	Wordpress (MySQL)	XML	RSS Feed
Contacts	Office365	XML	XML Feed
UI	SCRAM**	XML	File

* Amazon Web Services (AWS)
** In-house tools (POCs exist)

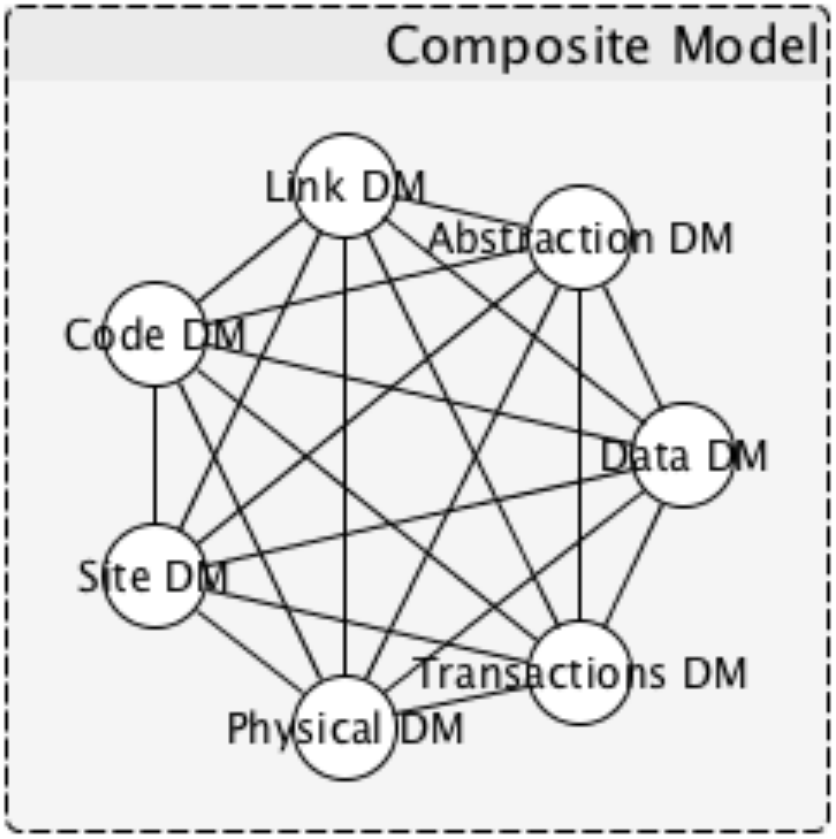
ETL (Ingestion Engine)





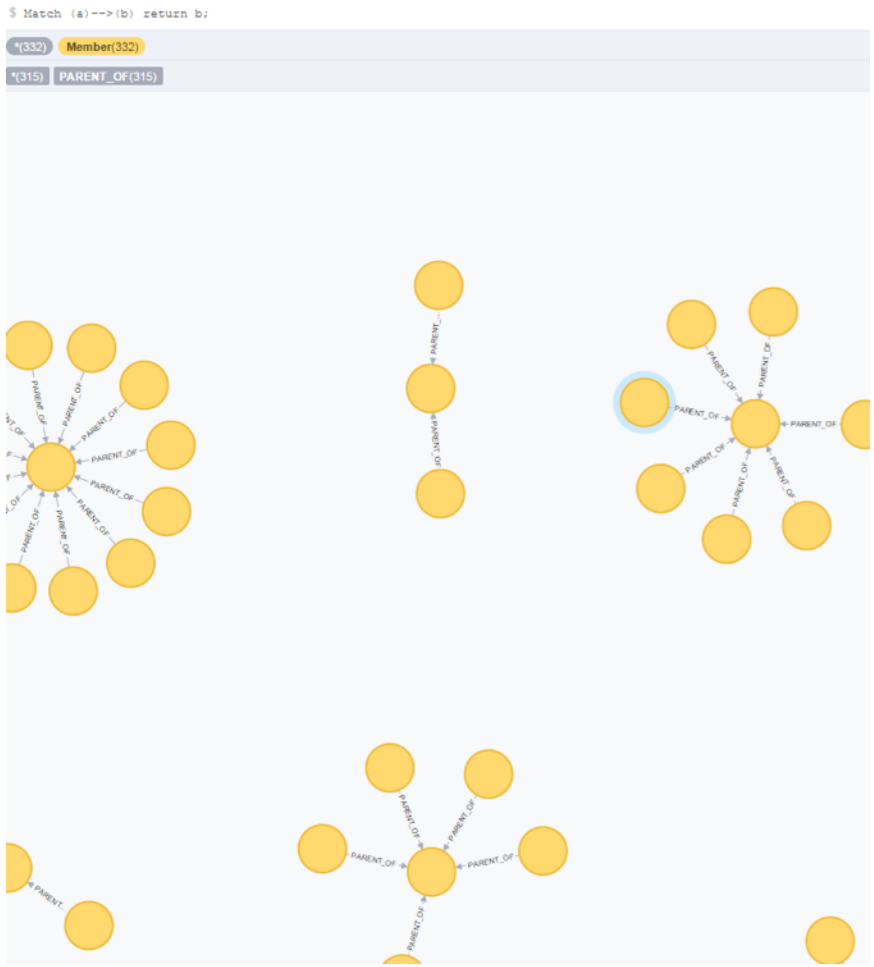
The Solution

- Mappers and Monitors
- **Data Model**
- Views associated with context



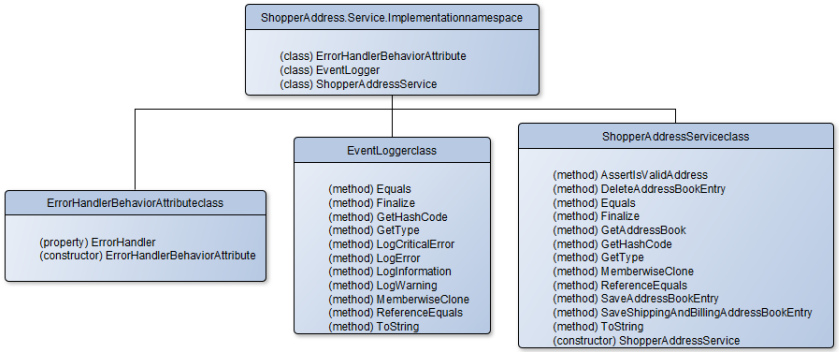
Database (Property Graph)

- Naturally associative: the relationship of one thing to another an element of the data.
- Scale to large data sets since property graphs do not require join operations.
- Graphs are schema-less and can manage ad hoc schemas and changing data.
- Graph databases support graph queries, for example computing the shortest path between two nodes in the graph.
- Can contain multiple-modalities, that is store information in different domains (i.e., events and objects) that can be joined at a particular level of abstraction.



The Solution

- Mappers and Monitors
- Data Model
- **Views associated with context (Rendering)**



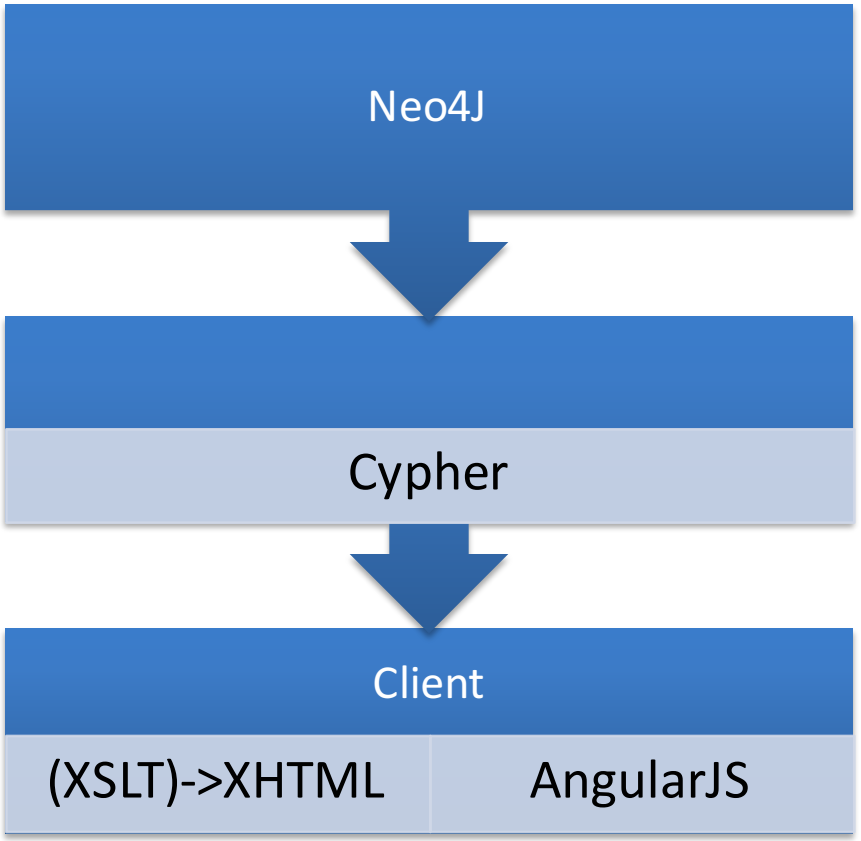
Views associated with context (Rendering)

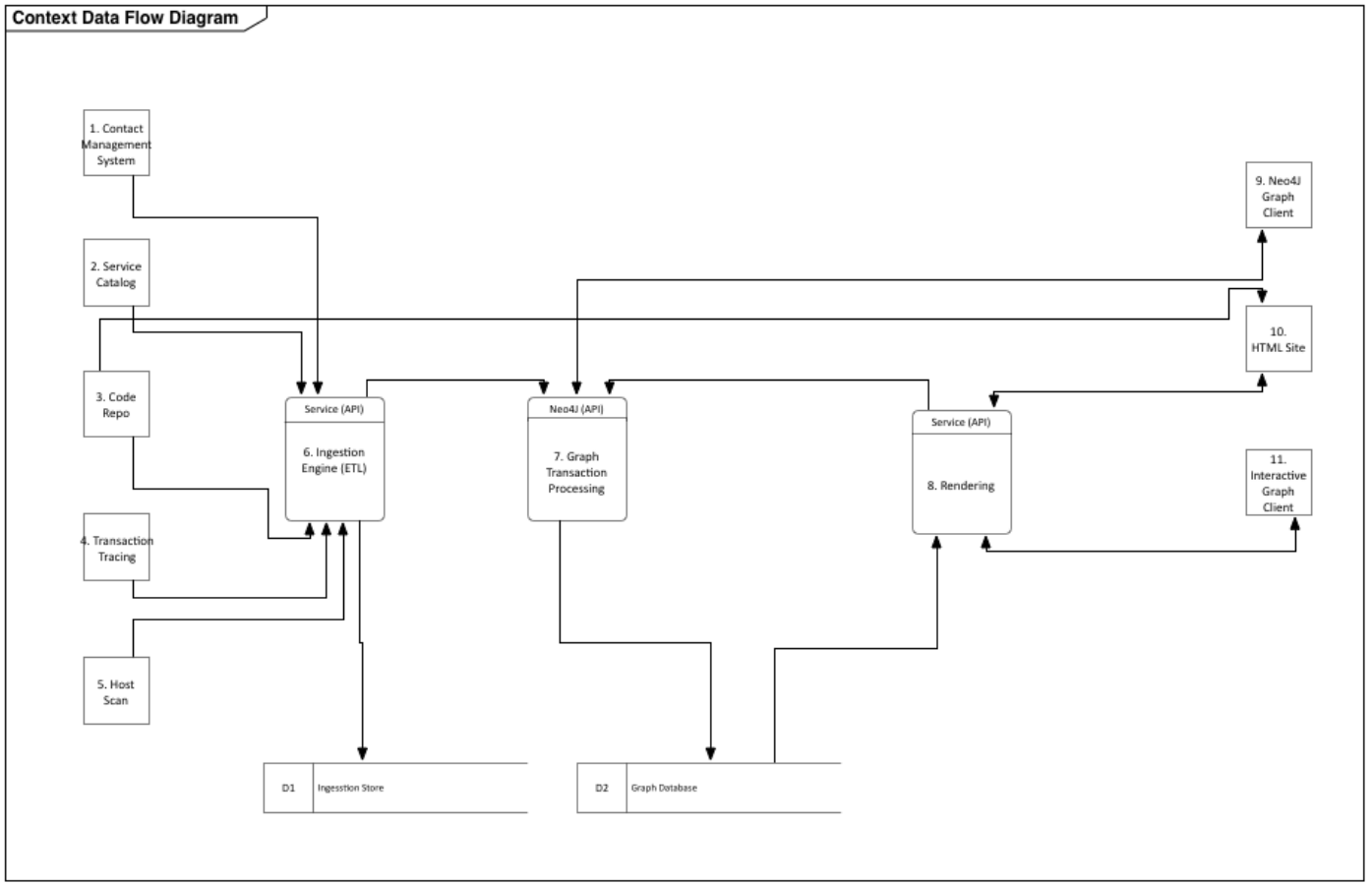
Type of Rending Tasks

- Ad hoc queries
- Static HTML Web Site
- Interactive Client
- JS DOM Hook

Technologies

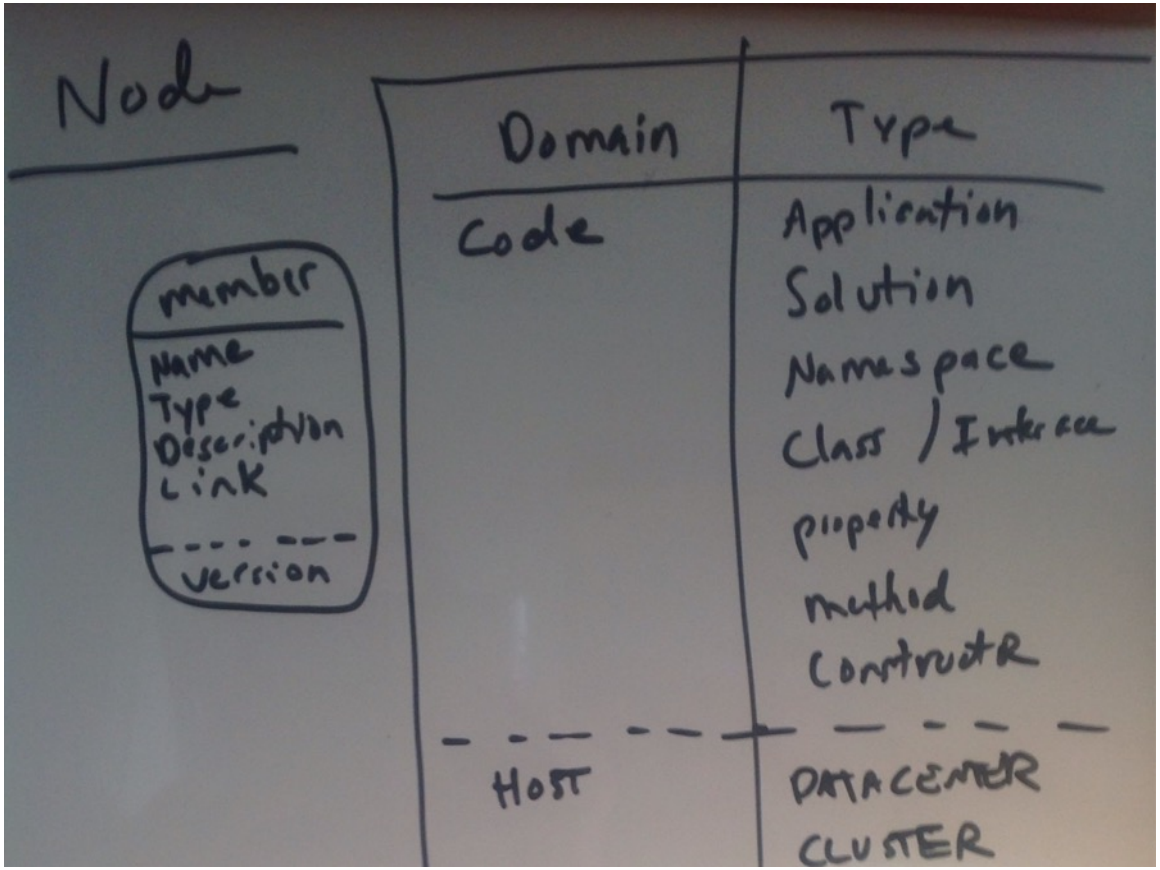
- Cypher Database Queries
- yFiles for HTML
- JSON -> XSLT -> XHTML
- <-Cypher->AngularJS





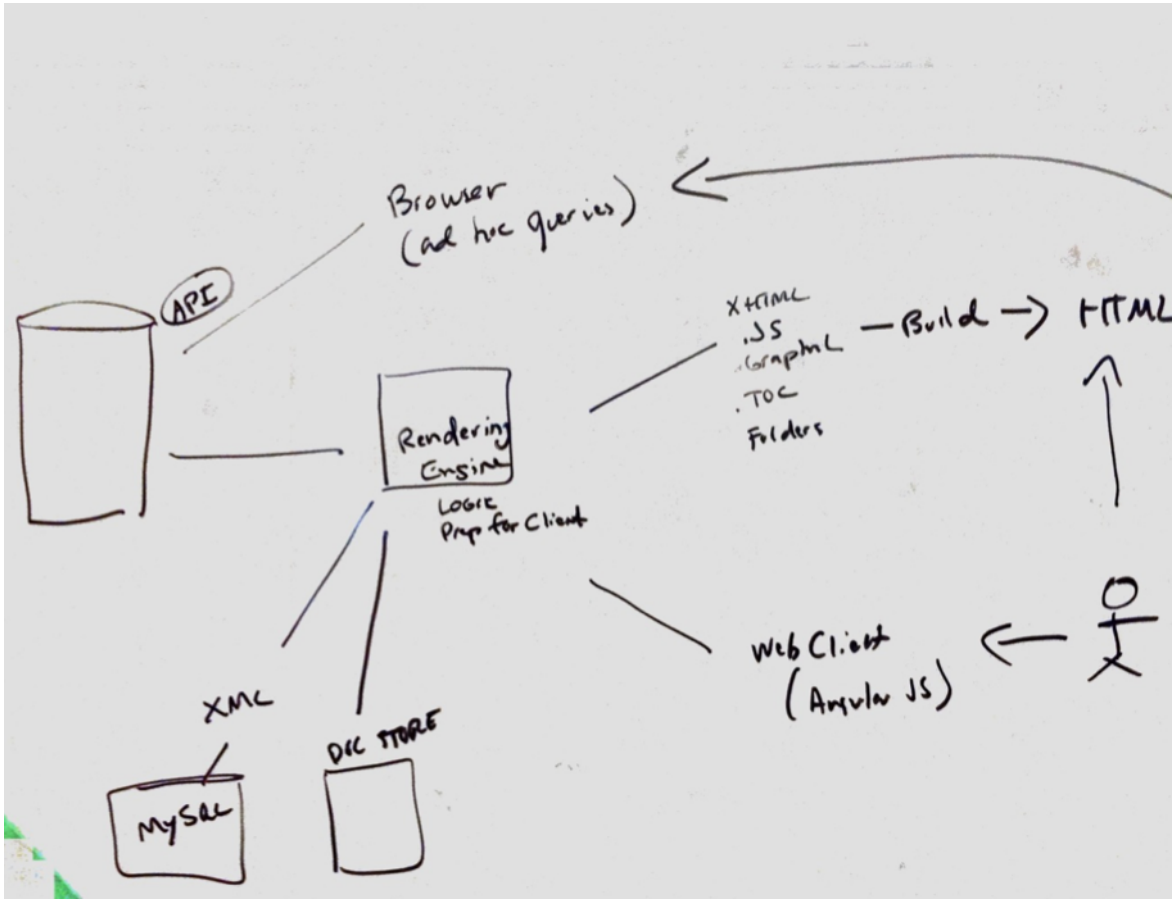
Context Diagram

Monitors and Mappers → ETL → Graph Database ←Rendering←Client



Graph Logical Diagram

Nodes and relationship schemas can be changed after they are created.



Rendering Logic

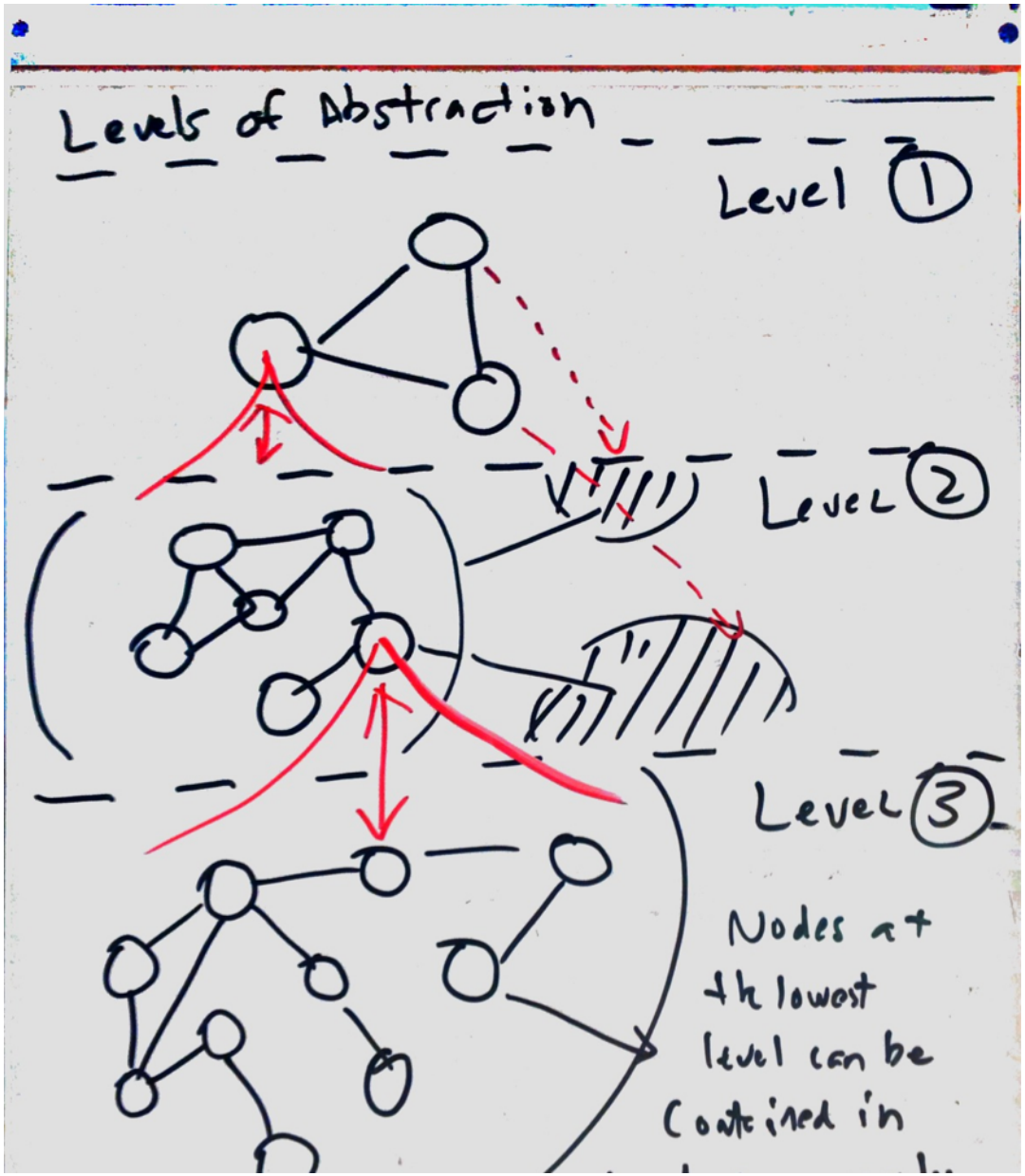
Graphs such as UML diagrams use Node-Edge property graphs. By using a node data base the underlying data is separated from presentation.

Levels of Abstraction

The amount of complexity by which a system is viewed or programmed.

The higher the level, the less detail. The lower the level, the more detail. The highest level of abstraction is the entire system. The next level would be a handful of components, and so on, while the lowest level could be millions of objects. A graph database is ideally suited to create these types of "roll ups."

The end up being essential in trying to encapsulate the complexity of the eCommerce system in a way that can be understood by a person. A person can typically only perceive seven discrete objects at time.



Automated Knowledge Base

QUESTIONS