
# Project Evaluation: MSE Project

November 28, 2010

**Prepared by** Doug Smith
**Version** 0.1

Table of Contents



# Revision History

| Version | Date | Changes |
|---|---|---|
| 0.1 | 11/28/2010 | First draft. |

# Introduction

This document provides an overall evaluation of my MSE project, including a summary of the problems encountered, an analysis of the time spent on the project, lessons learned, and remaining and future work.

# Problems and Challenges

## JTA Transactions in Standalone Applications

When building the first architectural prototype, a distributed transaction manager was needed to provide transactions that could coordinate the commit of changes to the database with the cache replication.

There are many standalone transactions managers that are available for Java programming, for example the JBoss transaction manager and the Atomikos transaction manager. The bigger challenge, however, was coordinating the use of the transaction manager among the Hibernate, JBoss Cache, and Spring APIs.

This required implementing a transaction manger lookup for JBoss Cache, and required explicitly wired transactions in Spring configuration, as opposed to using declarative transactions via annotations.

## Hibernate Caching Integration

At the start of this project, Hibernate has made some significant changes to their second level cache interface and APIs, and there had been some changes in the JBoss cache APIs as well. Because of the uncoordinated changes between the two open source projects, I had to use older versions of both libraries based on a known working integration between the two versions.

## Replication via Gossip Router

After deciding to target Amazon EC2 as the computing infrastructure for the project, I needed to adjust the replication method for the cache. Prior to adopting EC2, I had been using group multicast for communication between cluster members. However, multicast is not supported in the EC2 environment.

With the first architectural prototype, the underlying library providing reliable group multicast for JBoss cache is the JGroups library. This library provides a gossip server, where a process running the gossip server is started, with subsequence group members needing only to know the address of the gossip server. The gossip server then is able to send information about group members as nodes join the group, set up communication between group members, and so on.

Updating the system configuration to use the gossip server for JBoss cache node communication, I discovered a bug in the JGroup/JBoss integration that prevented correct functioning of the cache.

Learning the bug had been fixed in a later version of the JBoss cache software, I was forced to upgrade this library, which also forced an upgrade to Hibernate, configuration changes to transaction management in Spring, and so on.

At this point in time the caching API and JBoss/Hibernate integration issues had been solved, so I was able to get a version of the software working again.

### Object Relational Mapping

On the surface of things, Hibernate appears to provide a powerful mechanism for retrieving the persistent state of objects from the database. However, with power comes responsibility – while Hibernate abstracts away the nuts and bolts of retrieving data from a relational database and populating objects with data, several other concerns that are normally explicitly under the control of the programmer get pushed into XML configuration.

In my initial testing of the system, after enabling tracing and observing the database access, I observed that simple fetches of data could in effect traverse the entire relational graph of that object. In the naïve mapping case, this could mean retrieve for example a process definition could also fetch the process instances referencing that data, all their activities and the associated activity definitions, their property values and associated instances, and so on.

Obtaining the right set of data for each transaction required modifying relationships (going to eager fetching to lazy loading for example), scoping the relationships in the ORM mapping config to the view of the data desired when retrieving an aggregate root (for example a process definition referencing only definitional objects), and in some cases coding the persistence of related data when auto persistence based on configuration causes undesired retrieval patterns.

### Non-Performing Architectural Prototype

The biggest challenge that was overcome in this project was the failure of the initial architectural prototype to exhibit the desired scalability characteristics. This was detailed in the software architecture description document in phase 2 (for details see http://people.cis.ksu.edu/~dougs/Site/Phase_2_Artifacts_files/sad.pdf).

## Time breakdown and Estimation

Overall, I spent 376 hours working on this MSE project. The time was distributed as shown in the chart below across the 3 phases:

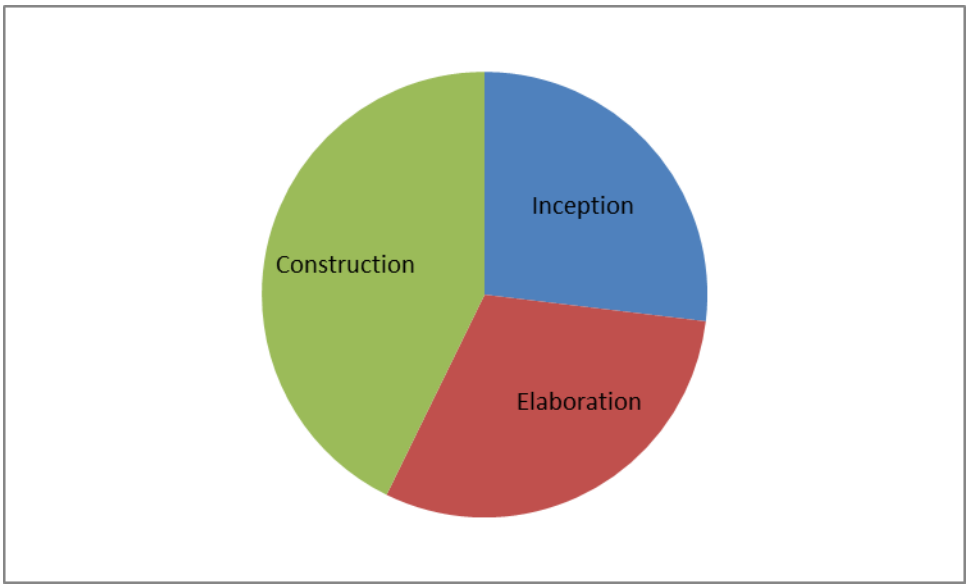


**Figure 1 Time Distribution Across Project Phases**

There were no real surprises with the time breakdown; I would expect to see a distribution similar to what was observed based on project norms and the lack of a transition to operations phase as part of this project.

Figure 2 shows the time breakdown for the inception phase. This has what I would argue are appropriately weighted activities for the inception phase: the bulk of the time was spent reading, investigating, prototyping, and documenting.

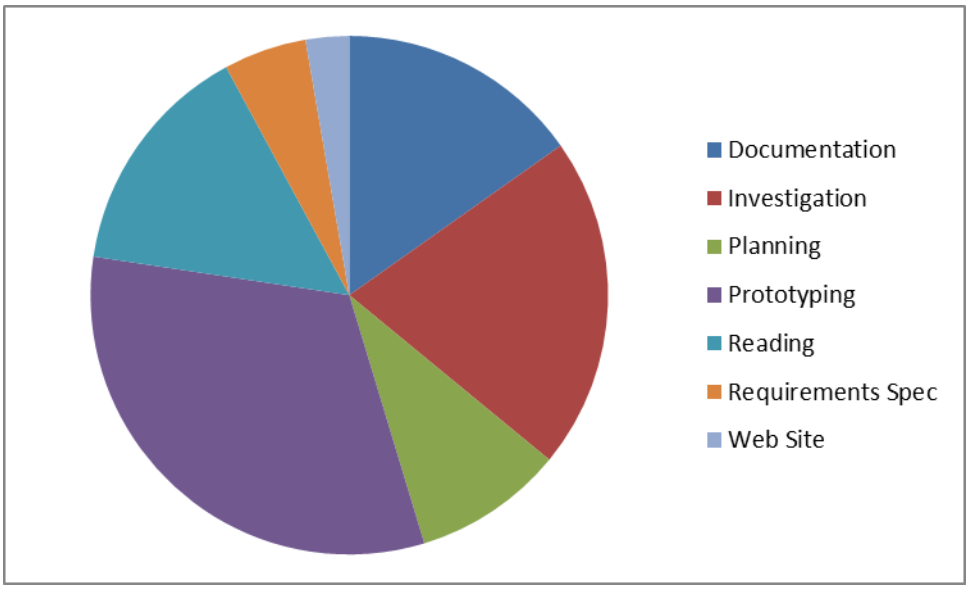


**Figure 2 Inception Phase Time Breakdown**

Figure 3 shows the time breakdown for the elaboration phase. As expected, the formal specification, system design, and the coding of the architectural prototype dominated the effort in this phase.

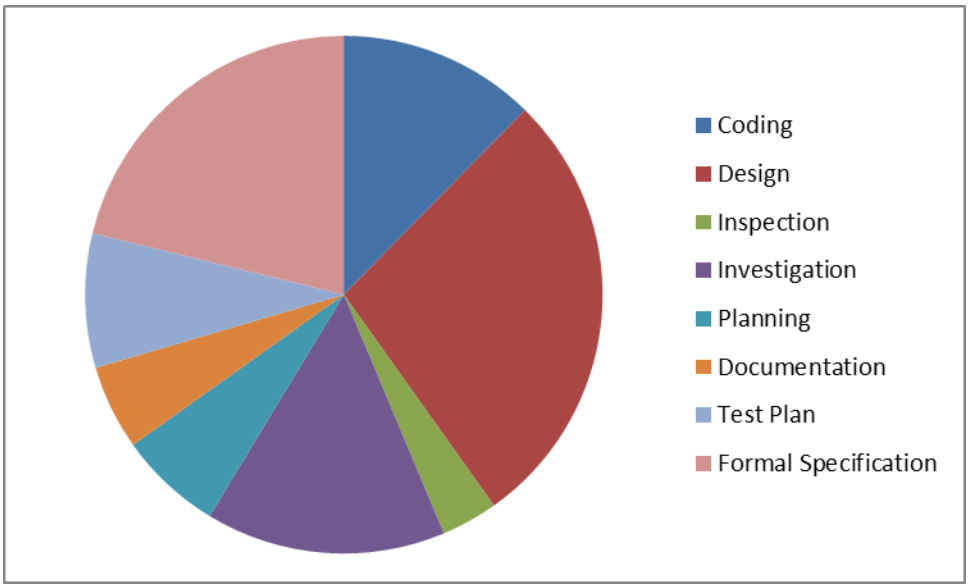


**Figure 3 Elaboration Phase Time Breakdown**

Figure 4 shows the time breakdown for the construction phase of the project. In this phase, testing took the most time for this phase. This was not due to a rigorous or thorough testing of the system or a system that was hard the test, but rather was the result of my performing many different types of scale testing against the original architectural prototype when the results did not meet my intuition. Quicker acceptance of what the testing revealed, namely the initial architecture did not produce the desired scalability characteristics, would have meant less time spent testing.

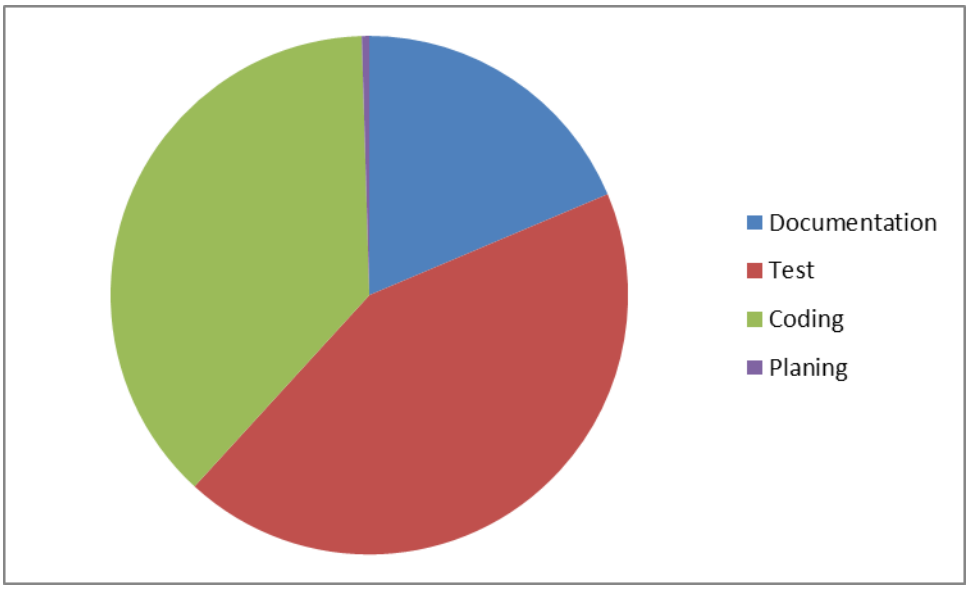


**Figure 4 Construction Phase Time Breakdown**

In terms of estimates, my initial estimate based on my intuition about the scope of the project and a work breakdown structure based on the artifacts produced as part of the MSE project, I estimated 555 hours, considerably more effort than the actual 376 hours expended. I could argue this was prior to the scope reduction taken after the inception phase, but I don't see the scope reduction being more than 80 hour reduction in scope.

Contrast this with the COCOMO II estimation model. While the COCOMO II estimate given at the end of the inception phase was way off, it was based on a SLOC count that was off by a factor of 5. The estimate produced at the end of the elaboration phase, based on a more accurate SLOC count, came in at 2.6 person months, which based on 152 hours per person month, puts the effort at 395 hours, which is pretty close to the actual effort (much closer than my intuition).

Before getting too excited about the COCOMO II result, it should be pointed out that for a small scale project such as mine, it will be very sensitive to the sizing metric, and when the sizing metric is SLOC, it can predict fairly well how long it takes to produce the lines of code based on the lines of code you've produced.

# Lessons Learned

## There's Always Room for Improvement

The MSE Software Portfolio project cuts across the entire software development lifecycle. In industry it is typical to see individuals focus their careers in a single discipline. This project gave me the opportunity to do it all, and with that came an appreciation of the breadth of disciplines in software engineering. This project challenged me not only in areas I have not focused on before, it showed me there is always opportunities to increase expertise and effectiveness.

## Real Progress Requires Focused Effort

I would describe my progress on this project to have come in fits and starts: when motivation was high and there was time available, I was able to make good progress. In other times, when work and family demands escalated, it was hard to make progress, especially in those cases when only 30 – 60 minutes could be spent at a time. The biggest progress came during days off from work when I could isolate myself and focus for several hours at a time.

Professionally, I have adjust how I manage my time to try to compress meeting and collaboration time into a single part of the day, retaining a solid block of time during the rest of the day to allow deep focus.

## You Can't Be Too Diligent in Record Keeping

Time tracking during the project turned out to me a challenge, especially when trying to squeeze in 30 minute here or 45 minutes there.

## Formal Specification is Very Valuable

The formal specification part of the project was more valuable than I thought it would be. Producing a UML model with OCL constraints produced the high level structure of the service, as well as the underlying data architecture. The artifact produced for the formal specification, and the effort that went into producing it, provided a solid conceptual foundation for the elaboration and construction of the system.

For a single person producing a system, the formal specification was much more valuable than the component designs. The component designs would be better suited to scaling up the construction of the system, but the component designs themselves would be higher quality when based on a formal specification.

### Trust Data Over Intuition

While I was able to acquire the data that showed my initial architecture did not scale in the elaboration phase, instead of rethinking the architecture at that point I doubted the tests, and redid them several times using different approaches. In other words, I thought the tests were flawed, not my architecture.

After distancing myself from the problem and trying the think deeper about what was going on, I realized that in fact there were some fundamental problems with the initial architecture, to the point where optimizing the performance of one part of the system only increased the pressure on a bottleneck elsewhere in the system. The acceptance of the facts presented by objective data allowed me to come up with a different architecture that produced the desired results.

### Infrastructure As A Service Is A Huge Development Aid

Using Amazon's Elastic Compute Cloud (EC2) infrastructure as a service (IAAAS) platform provide to be a very effective way to dynamically allocate computing resources, including computing resources and load balancers. I was able to scale up my system on demand at a very effective price point.

### Invest in Automation

While there are good consoles and other tools for EC2, I would have made even better used of EC2 had I taken full advantage of Amazon's APIs and toolkits to automate launching and shutdown of images, and to automate load balancing configuration. Tooling to script application configuration and deployment would have also increased efficiency, as well as producing a Linux system image that automatically booted Tomcat.

## Remaining and Future Work

### Metadata Services

Currently process definitions codified as metadata are loaded into the database using SQL scripts. The initial system vision included service enabling process definition, such that definitions could be created and modified using web services.

### Automated Deployment

Currently, configuring and deploying a system requires several manual steps. The APIs, tools, and services needed to automate application deployment exist, this is a natural follow on for this system.

## Apply the Patterns of Domain Driven Design and CQRS

The design for this system was produced prior to my introduction to domain driven design. While the initial architecture of this system was conventional, it had any of the same problems inherent in typical multitier applications, such as an anemic domain model that is mostly data oriented, with behavior external to the domain objects. It also has a data model that must scale both for write transactions as well as for read transactions, while being optimized for neither.

Given the nature of process instances and tasks, there are several opportunities to leverage DDD patterns, such as aggregate roots and bounded contexts. A redesign of the system using DDD would be interesting.

With the introduction of the in-memory data grid, we were able to mitigate some of the problems related to a single model supporting reads and writes by virtue of temporal decoupling of the database write from the write transaction. In retrospect, however, segregating reads from writes via the Command-Query Responsibility Segregation pattern would have allowed me to independently scale the reads from the writes.

## Consider Alternate Data Stores

Given the content and properties of process instances and their tasks are not known at application construction time, the fit of the relational model as an appropriate data representation mechanism should be questioned. As process instances and tasks are access in their entirety based on a unique identifier, a document oriented store such as MongoDB or Couch might be a more appropriate storage mechanism, as opposed to using a relational data store and an Object-Relational mapper. Using CQRS to produce query views independently of the store needed on the write side also makes this technology more applicable.