Project Plan for Team Falcon's Peregrine

Contents:

1. Overview

As an SOA engineer for A-DEC I need a way to make sure that my jobs and processes are running smoothly. To do this I need to have a consolidated view into the progress of my jobs, and any error messages my process might throw. I would like to have a simple WCF interface to hook my .net processes to, a UI that runs in IIS, and a backend that runs on Microsoft SQL 2008. As an applications maintainer, I would like my logger to run in an N-Tier architecture.

The goals of this project are to provide A-DEC with a tool that they can use to get close to real time monitoring of their manufacturing applications.

2. Organization

- Technical lead
    - Devon Gleeson
- Product Owner
    - Joe Capell
- Documentation lead

    - Nick Benson
- UI
    - Chinh T Cao
    - Anh Nguyen
- Database/Persistence
    - Nick Benson
    - Wilson Lu
- Service API
    - Devon Gleeson
    - Kyle Paulsen
- System Administration
    - Devon Gleeson

3. Resources

The team has procured capstonebb.cs.pdx.edu, we have set up a hudson server that will automatically run our unit tests upon a commit on the branch. This server also has SQL Server 2008 for remote DB testing, and Enterprise IIS that will be used by the Service API team to develop the API. We also have a ReviewBoard instance for code review, and a Jira instance for tracking the progress of the project. The code is hosted on a private github branch.

4. Quality Assurance

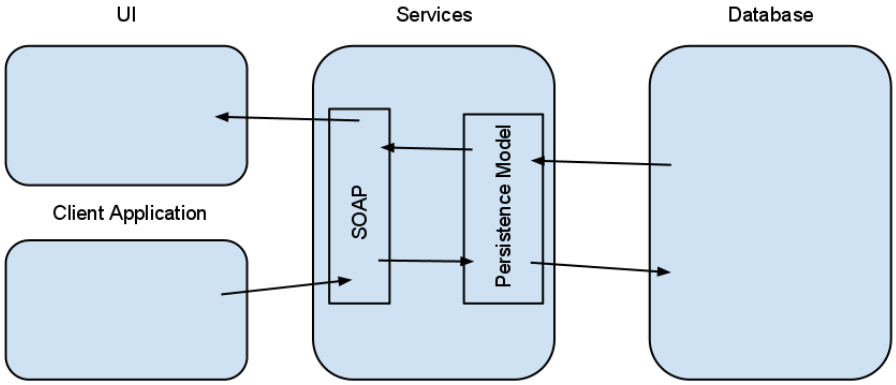
Each sub team has 2 people on it. A development story will be owned by one person on the team, and the other person will provide QA for that story. What this means is that each team should be able to have two development tasks going on at any given time. We are going to strive for 80% code coverage of unit tests. We plan to use NUnit to write unit tests for our project. Running of the unit tests will be automated on hudson.

5. Plans of Attack

5.1 Persistence Plans of Attack

The goal is to build a MS SQL database for logging message or information via web service and fit in the architecture of the Peregrine project. The tasks include but not limit to,

1. Create a database as the center back-end data repository,
2. Do object relational mapping from class object to table, table column and row in the database,
3. Maintain remote access to the database through web service.



MS SQL database in the Peregrine architecture

MS SQL is chosen for implementing the database backend. We are going to use follow C# library for the MSSQL database with MS Visual Studio 2010,
using System.Linq;
using System.Text;
using System.Data.Linq;
using System.Data.Linq.Mapping;
With above library, Linq to SQL classes that are mapped to database tables and views are called entity classes. The entity class maps to a record, whereas the individual properties of an entity class map to the individual columns that make up a record.
The MS SQL database resides in remote IIS server (for instance, in CS Capstone lab) and can be accessed remotely via web service.

There are two alternative ways to build SQL database with C# on Visual studio. With following pros and cons considerations on both two alternatives, we make an architecture decision to implement database in linq to SQL libary,
1. Use linq library
Pros: New technology built on Entity frame work and can develop application in short time.
Cons: Not as flexible as Entity frame work

2. Use Entity frame work
Pros: Flexible and scalable/ Can handle data base other than SQL
Cons: Take more effort to build

3. Details Advantages of linq to SQL

1. Applications can work in terms of a more application-centric conceptual model, including types with inheritance, complex members, and relationships.

2. Applications are freed from hard-coded dependencies on a particular data engine or storage schema.
3. Mappings between the conceptual model and the storage-specific schema can change without changing the application code.
4. Developers can work with a consistent application object model that can be mapped to various storage schemas, possibly implemented in different database management systems.
5. Multiple conceptual models can be mapped to a single storage schema.
6. Language-integrated query (LINQ) support provides compile-time syntax validation for queries against a conceptual model.

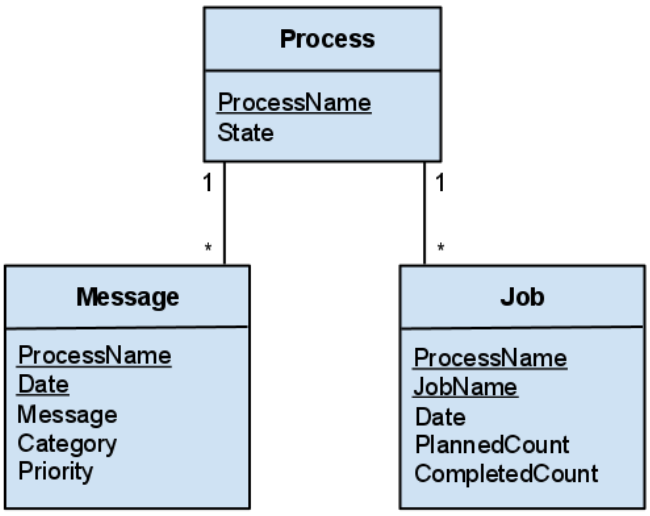


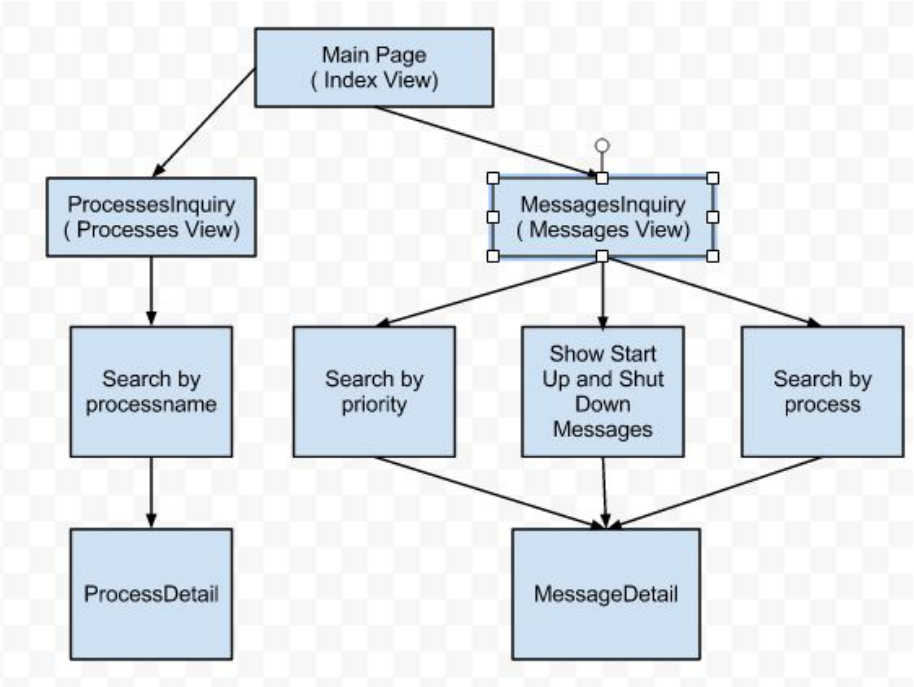
Figure 2. Example schema

Automated test strategy
Install Microsoft SQL server and Hudson (Jenkins) on Capstone lab team project computer. To write a script that installs the tool on a fresh database in Capstone lab and automatically verify all the tables and columns are correct.

5.2 UI Plans of Attack

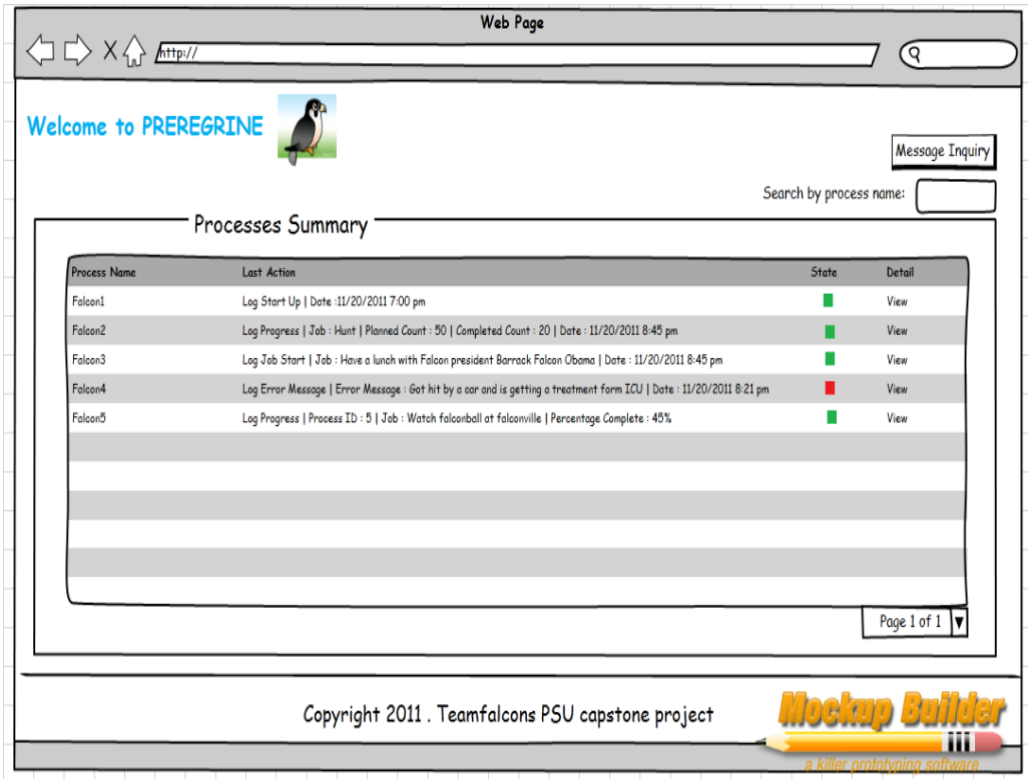
Our goal is to build a web page for the Peregrine project to display the data to users. The application retrieves data from the API and arrange it on the web page. The web page must satisfy desires from user and as must be as user-friendly as possible.
We use ASP.NET MVC 3 Framework, AJAX and HTML5 for the web page.

Website UML



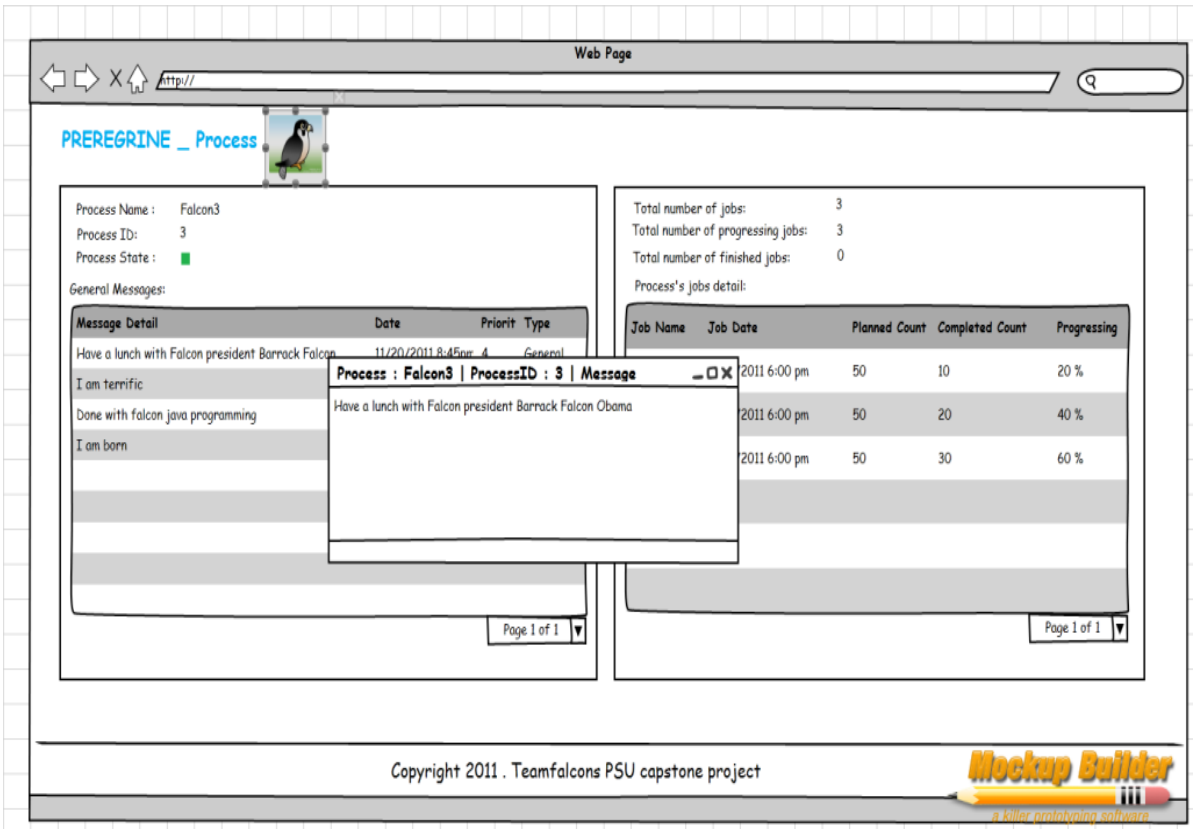
Views Mockups

Main page :

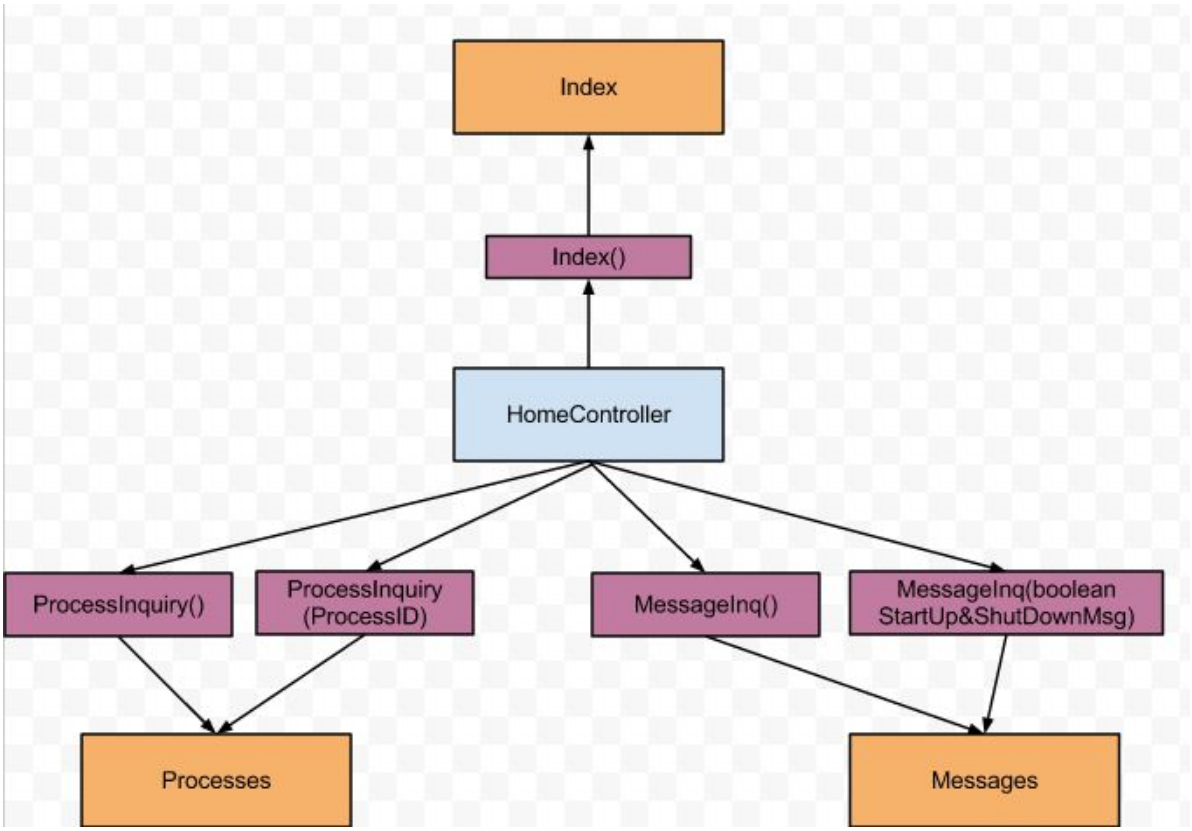


Message inquiry:

Process page:



MVC Pattern Design



Controller functions - corresponding View:
Index() - Index
ProcessInquiry() - Processes
ProcessInquiry(processName) - Processes
MessageInquiry() - Messages
MessageInquiry(boolean ShowStartUpAndShutDownMessage) - Messages

DataTables

DataTables is an amazing JQuery plugins. It is flexible, easy to use and fits well to the needs of our project.
The full capacity about datatable can be shown at http://datatables.net/
I tried to do a quick run with datatable and this is the result. The draft UI of our main pages.

5.3 Service API

Our goal is to create a WCF service for message logging that can be deployed on a Microsoft server running IIS. The service needs to be able to create and manage the logging database as well as provide SOAP methods for accessing that database. It is essential that our service has no downtime and is only performing a task when a request is made to it or it is performing database maintenance. It is also very important that the service can handle multiple connections to it.

Each log message should be associated with a “Process” that it came from. Messages need to contain useful information such as a timestamp and a priority. The service is responsible for inserting, retrieving and deleting these messages.

“Processes” can also contain some number of “Jobs”. The service needs to track these optional jobs in order to provide update information about the same job at a later time.

We will use Microsoft Visual Studio 2010 to develop the service in C#. We will also use review board and git to help up us with our version control. To test deployment, we will use Microsoft Server 2008 and have it run IIS. For automated testing, we will use Hudson. Automated tests might include sending in a collection of requests to perform actions on the database or performing a timed request to make sure the service isn’t hanging too long somewhere.

See the database plan of attack for more detailed information about the “Process”, “Message” and “Job” objects.

6. Sprint Outlay

We have already completed our first sprint as a team, the work was to produce this document and the plans of attack in them. Here is an idea of how we foresee the project unfolding with a story per sub team per sprint. Each story will have at a minimum 4 tasks, those being, “do the dev work,” “quality verification,” “documentation,” “code review/commit.”

6.1 Sprint 2 (November 28th - December 5th)

- API Team
  - Research WSDL definition and binding
  - Begin design on the AP I interface
- DB Team
  - Implement Basis DB Connector
  - Research Basic Startup Process
- UI Team
  - UI Mockups/UI Architecture

6.2 Sprint 3 (December 12th - December 25th)

- API Team
  - Complete API Interface
  - Begin Building Test Tool for DB
- DB Team
  - Implement LINQ for Processes
- UI Team
  - Prototype UI (AJAX), User interview.

6.3 Sprint 4 (December 26th - January 8th)

- API Team - Just a note, Devon is working on this team and will be off work during this sprint so he will have a lot of extra time to spend on this project.
  - Finish Test Tool for DB
  - Research Request Bindings
- DB Team
  - Research Stored Procedures
- UI Team
  - Research HTML 5 (CSS, Masterpage), finish design (flowchart, mockup)

6.4 Sprint 5 (January 9th - January 22th)

- API Team
  - Implement Test Tool for UI
- DB Team
  - Implement LINQ for Jobs
- UI Team
  - Finish UI (Finish AJAX, CSS, Model class after getting Test Tool)

6.5 Sprint 6 (Janurary 23th - February 5th)

- API Team
    - Research Installers and application startup
- DB Team
    - Implement API interface with the help of the API team
- UI Team
    - Connect to API

6.6 Sprint 7 (January 6th - February 19th)

- API Team
    - Implement installers and application startup
- DB Team
    - application startup
- UI Team
    - application startup

6.7 Sprint 8 (February 20th - March 4th)

- Code Complete/Bug Fixes

6.8 Sprint 9 (March 5th - March 18th)

- Code Complete/Bug Fixes

7. Risk Managment

- Less work gets done during the holiday break than anticipated
    - We've padded the back of our sprint outlay with 4 weeks of oops time.
- UI Implementation does not move along as expected.
    - Kyle can move off of the API project and help work on the UI
- Database implementation does not move along as expected.
    - Devon can move off of the API project and help work on the DB
- API team gets behind in the design of the API Interface.
    - We can defer some of the installer work that is not strictly required by the project.