Managing a Student Project with Enterprise Architect – Part 2

Doug Rosenberg -- ICONIX and USC

www.iconixsw.com

Introduction

In Part 1 of this article series we presented some introductory info about a student project that I’m managing at the University of Southern California Center for Systems and Software Engineering (USC CSSE). We’re attempting a “crowdsourced bad driver reporting system” this semester, and because we need to be really productive, we’re using Enterprise Architect to model the project, field-test the Resilient Agile process, and to coordinate all of the student homework. Students communicate with each other and with me using a shared EA model.

This semester I’m working with a group of 15 Masters students and an aggregate effective time budget of 80 student hours per week. We’ve got about 12 usable weeks of student time, so it works out to a time budget of roughly 1000 student hours (that’s about half-a-person-year at 40 hours a week) over a 3 month schedule.

Resilient Agile is a flexible process in that it can be employed with traditional Scrum/Kanban sprints and backlogs, or alternatively we can leverage parallelism, and each student can be assigned a use case and develop their use case independently.

I’ve been a big fan of leveraging parallelism in software development since I was a programmer at NASA/JPL way back in the 80s when I rescued a late project using a “divide-and-conquer” coding strategy, so we’re trying to see how far we can push the limits on massively parallel development with student projects at USC. Communication and well-defined interfaces are key when team members are working in parallel, so the shared EA model is critically important.

Parallel modeling and development has also been a theme of our ICONIX JumpStart classes for the last 20 years, where we go into industry and work a client’s real project by splitting the class up into “lab teams”. Typically in ICONIX JumpStart classes we put 3 or 4 students on a package of use cases, whereas on this project each student got a single use case.

If you’re going to leverage parallelism in development you have to do things a little bit differently. Here’s an overview of the process we’re following:

1. Plan for Parallelism (identify dependencies and architect for parallelism)
2. Build the Right System (discover requirements, prototype areas of technical risk, and agree on conceptual designs)
3. Build the System Right (carefully review detailed designs)
4. Integrate as often as necessary

Enterprise Architect is a key enabler of the above process. I would never attempt this approach without a good solid modeling tool at the heart of it. This article will show how we’ve used EA to accomplish the 4 steps above.

On this project we’re also field-testing a PhD-student developed code generator for NoSQL databases and REST APIs, which he has built into EA, and we’ll show how that works as well.
Crowdsourced Bad Driver Reporting System – what and why?

In case you missed Part 1 of this article series, I got this idea for the bad driver reporting system while I was working on a consulting assignment that required me to commute from Santa Monica to the Los Angeles Airport (LAX) area, putting me on some of the busiest freeways you'll find anywhere.

One day after a particularly harrowing commute, I came up with the idea of building a crowdsourced video database for insurance companies using a voice-activated “dashboard cam” app to capture and upload video. So here’s the mission statement I gave this semester’s students:

*Mission is to eliminate bad drivers from the road system by providing a risk database to insurance companies so they can charge more to bad drivers. Business model is “software as a service”.*

The basic operational concept of our system is as follows:

*Dashboard camera continuously records looping video. Voice command to mobile triggers the autosaving and uploading of 15 second video clip to a cloud-database. License plate numbers are extracted from imagery (optional), and video/report metadata is filed by license plate number. Insurance companies can query against license plate numbers to see if there are any bad driver reports logged against a vehicle while issuing policies.*

Plan for Parallelism

Drawing a use case diagram helps us identify dependencies between use cases:

![Figure 1 - Bad Driver Reporting System Use Cases](image)

When we analyze dependencies we’re trying to find scenarios that can be developed independently, while communicating through our REST API with the Mongo cloud database.
The first step in planning for parallelism is to assign a student to each of these use cases.

So this semester we've got 5 students on the Android team, 2 students on the iOS team, and about half-a-dozen students on the Web App team. A couple of highly motivated students volunteered to be on two teams. We also have two students on the test team.

If we were teaching an ICONIX class in industry, we’d simply set the EA model up on a shared server and collaborate in real-time, but with a student project we decided to have each student deliver their homework by exporting their package to XMI, and emailing it to me. This way I get to review progress and make suggestions as I merge the homework into the master model. I republish the entire model and a “diagrams report” to the whole class a couple of times a week, generally before class meets on Wednesday and Friday.

**Architect for parallelism**

The architecture of the system is shown here: mobile apps for iOS and Android, connected to a MongoDB repository via a Node JS RESTful API, combined with Angular JS pages to handle web-forms for posting and reviewing reports.
This "microservice architecture" is in widespread use across industry, so students are gaining valuable job skills while working on the project. This architecture is also amenable to our “experimental secret weapon” code generator, which takes an EA class diagram (domain model with attributes) and autogenerates a fully functional Mongo database along with a Node JS API for all of the database CRUD (Create Read Update Delete) functions.

This code generator enables a key concept of Resilient Agile called "executable domain models" and on this project gives the entire class an “instant database” that they can write to and read from. As the database schema evolves while we work through the use cases and develop technical prototypes, we can just re-generate the schema and REST API.

Don’t code by hand what you can instantly code generate

The Resilient Agile process supports evolutionary development. The architecture of our system supports parallel development because students can integrate their work to a NoSQL database in the cloud. The ability to instantly re-generate the database as the schema evolves while prototyping and exploring requirements and conceptual designs enables evolutionary development at a different level than traditional agile/scrum/TDD processes do.

![Figure 3 - Executable Domain Models enable evolutionary development](image)

We developed a domain model with attributes immediately when the class started meeting and students start writing first draft use cases and doing conceptual designs. Elaborating the use cases using “conceptual model-view-controller” diagrams pointed us towards the right set of Mongo collections to develop and an initial set of attributes for each collection.
It's never too early to start testing

We generated our first draft database about 2 weeks into the semester, and immediately put our test team to work exercising the database. Here we've imported a screenshot of a test run from a tool called Postman into the EA model so that all students can see the results when they look in the EA model.

![Test Run Screenshot]

**Figure 4 - Good News! Our generated database code works!**

Once the database came on line, students who were prototyping could immediately access it. On this project, that means the mobile app team can start uploading video and the web app team can start using the uploaded video.
Building the Right System

During the first month of the semester the students focused on "building the right system". This exercise involved the following elements:

- **Storyboarding screens**
- **Writing first-draft use case narratives**
- **Identifying requirements for each use case**
- **Disambiguating use cases using MVC decompositions**
- **Risk mitigation by prototyping**

Each student delivers homework as appropriate for the use case they were working. As a general statement our web app team did more modeling and the mobile app teams did more prototyping as we were investigating things like how to make the app respond to voice commands and continuously capturing video to a circular buffer. We'll discuss each of these in turn for the remainder of this article.

**Storyboarding**

Students storyboarded their screens (in this case using EA's built-in storyboarding tools) and we iterated on the use case narrative, requirements, and storyboards as they delivered homework to me each week. Figure 5 shows the storyboard for Ankita's screen that allows insurance agents to review bad driver reports for a given license plate number.

![Storyboarding helps develop use cases and identify requirements](image)
Disambiguation via MVC Decomposition

Most people write first draft use cases that are vague and ambiguous, and our USC student developers were not exceptions to this rule. Fortunately for this group of grad students, my co-instructor Hunter and I have decades of experience at teaching developers how to disambiguate use cases by drawing a conceptual Model-View-Controller diagram (aka robustness diagram).

Here in Ankita’s homework you can see the use case narrative in the large note and functional requirements linked to the controllers and views.

![Diagram](image)

**Figure 6 – Decomposing use cases into Models, Views, and Controllers forces disambiguation**

Reviewing these MVC diagrams, along with the storyboards, as each student delivered homework by email, allowed me to avoid scope creep and coordinate work that students were doing independently. Also our test team (Parth and Seema) used the ICONIX/Design Driven Testing add-in to generate test cases for each controller.

We’ll show the sequence diagram and Angular JS screen for Ankita's use case in Part 3 of our article series, when we discuss “Building the System Right”.
Risk mitigation via technical prototyping

One of the key requirements of our system is hands-free operation. Since fiddling with cell phones is one of the leading causes of traffic accidents, we thought it would be a bad idea for a bad driver reporting app to cause more accidents by causing drivers to fiddle with their phones. So our mobile app team searched around for a voice activation SDK and Longjie, Rajat and Preksha started prototyping voice activation on both iOS and Android.

![Figure 7 - Our bad driver reporting mobile app is voice-activated so people don’t fiddle with their phones](image)

Once our prototypers had hacked a path through the jungle, we captured how the voice toolkit works in the EA model. Other students (Zhongpeng and Qinfeng) undertook technical prototyping for continuous video capture to a circular buffer, and Ankur prototyped a dual display that allows the app to toggle back and forth between a map view and the video camera view. Meanwhile Shobha developed a REST API for video upload and download and our remote student Tapashi worked on a license plate number reader.

![Figure 8 - We decided that a map display would be useful in our dashboard cam app](image)

Once again, we imported the screenshots into the EA model so that all students could see what their classmates were doing.
Wrapping up Part 2

Part 1 of our article series introduced you to the crowdsourced bad driver reporting system project, and in this article we've covered Planning for Parallelism and Building the Right System. Our next article will discuss Building the System Right and then we'll show you the final results of the project at the end of the semester.

To recap what we covered in this article:

Planning for Parallelism involves doing a good use case decomposition and understanding dependencies between use cases. We've also enabled parallel development by choosing an architecture that features a NoSQL cloud database (Mongo) accessed via a RESTful API (Node JS). Parallel development is virtually impossible without some form of shared communication among the participants. Enterprise Architect plays a vital role in enabling communication across the project.

Building the Right System involves writing use case narratives, identifying requirements for each use case, and disambiguating the use cases using conceptual MVC decompositions. I reviewed student homework while merging in XMI exports from each student's package into the master EA model, and then re-publishing the model to the class a couple of times a week. We complemented this modeling activity with an aggressive prototyping strategy to mitigate technical risk, and then share the results of the prototyping in the EA model.

If we've got your attention now, stay tuned for Part 3 when we explore the design of the system further. As I'm writing this, our student team is busy bringing the system online.