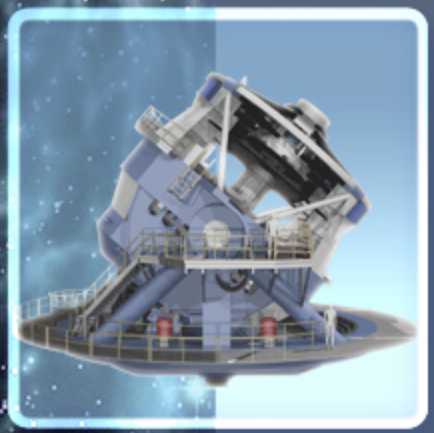
Designing the Large Synoptic Survey Telescope's Image Processing Pipelines

# 20 Terabytes a Night

by Doug Rosenberg with Matt Stephens



# 20 Terabytes a Night

*Designing the Large Synoptic Survey Telescope's Image Processing Pipelines*

## Contents


*Foreword*

## Geoff Sparks, Sparx Systems CEO

Since 2002, Sparx Systems has benefitted by having ICONIX as a member of its Global Partner Program. ICONIX CEO Doug Rosenberg is a luminary in the field of Software Engineering and noted author. Over the years he has contributed valuable expertise, helping to ensure Enterprise Architect provides world-leading support for the ICONIX process.

At Sparx, we have enjoyed the 'hands-on' experience Doug has related to us from years of successfully applying the ICONIX process to projects in industry. We'd like to share some of these insights with the broader Enterprise Architect community.

In this e-Book, we asked Doug to distill his experiences and lessons-learned from the Large Synoptic Survey Telescope (LSST) project. The sheer size and complex nature of LSST, bring a unique set of challenges and a massive software modeling endeavor. However, the unchanging principles behind use of model abstractions, UML and the ICONIX process remain beneficial in such an undertaking, as highlighted throughout this account.

We hope you also enjoy and benefit from Doug's shared experience on the amazing LSST project!

*Prologue*

# Long Ago (and Some Galaxies Far Away)

Before we get started, here's a short summary of how I came to be involved with the Large Synoptic Survey Telescope (LSST), and an introduction to a couple of the key players on the LSST team (and good friends of mine), Tim Axelrod and Jeff Kantor.

## *NASA JPL: The Birthplace of Image Processing*

I graduated from the University of Southern California in 1980 with a degree in Electrical Engineering and the ability to program computers in 12 or 13 different languages—and having taken only one Astronomy course (which I enjoyed quite a lot). I bounced around between a couple of aerospace companies in Southern California and a VLSI CAD company in Silicon Valley for a few years, and discovered that

- *5% of the people in high-tech knew everything, and did 95% of the work*
- *I had absolutely no stomach for company politics*
- *Bad technical decisions made for political reasons on my projects kept me up at night*
- *Consultants/contract programmers made double the salary of regular employees*
- *I had the ability to create software of significant value*

Given these discoveries, it seemed to make sense to become a contract programmer, double my salary, and invest the extra money in starting my own business to create software (and only hire "top 5%" overachievers). It took me a couple of years to figure out what kind of software I wanted to create, and I finally settled on developing better tools for programmers.

So 25 years ago (1984), I found myself as a contract programmer at the **NASA Jet Propulsion Laboratory (JPL)** working in a lab called the Multi-Mission Image Processing Laboratory.[1] This happened to be the lab that was processing the photos received by Voyager as it passed Jupiter, Saturn, Uranus, and Neptune. I wasn't personally doing image processing, I was working on a command-and-control system that did something called *Tactical Data Fusion*, where we would take all sorts of different information, fuse it together, and display it on a map. But I was surrounded by folks who were doing real image processing and I always found it to be interesting stuff. Plus the giant photo of Jupiter's Red Spot[2] on the wall of the office where I worked was pretty cool. It's possible that somebody, somewhere was doing image processing before JPL, but they started doing it in 1966, so MIPL was certainly one of the places where the image processing techniques now being used on LSST were developed.

I worked four 10-hour days a week at JPL, and spent the rest of my time starting ICONIX. I had bought a Lisa 2/10 computer (the predecessor to the Macintosh, which came out in 1984) that had a 32 bit processor, 2 Megabytes of RAM, and a 10 Megabyte hard disk, which was a lot of computer

[1] http://www-mipl.jpl.nasa.gov/

[2] http://photojournal.jpl.nasa.gov/catalog/PIA02259

for $10,000 back then. Our department VAX 11/780 minicomputer supported 16 concurrent users on something like a single megabyte of RAM. By contrast, the topic of this book is an image processing system that will process ***20 Terabytes of data every night*** for a decade.

## *NASA Johnson—Space Station SSE*

ICONIX changed from being a pipe dream to a real business in 1986-87 after I met Jeff Kantor at a conference in Seattle called the Structured Development Forum (OO methodologies hadn't been invented yet). Jeff was working near NASA Johnson in Houston, defining the common Software Support Environment (SSE) for the Space Station.[3]

Jeff wanted an option for developers to use Macintosh computers, and ICONIX was just about the only game in town. We opened an office after Jeff bought 88 licenses of our Mac CASE tools (called *ICONIX PowerTools*), and ICONIX became a real company. Jeff is now the LSST Data Management Project Manager, and a key player in this story.

## *NASA Goddard—Hubble Repair Project*

A quick check of the NASA website shows that the first servicing mission to the Hubble Space Telescope was flown in December 1993 (another servicing mission is about to be flown as I write this[4]), which means that it was sometime in 1992 when I found myself in Greenbelt, Maryland at the NASA Goddard Space Flight Center, teaching a class on Structured Analysis and Design to the team that was re-hosting the coprocessor software.

Many people are aware that when the Hubble was first built, there was a problem with the curvature of the main mirror (it was off by something like the 1/50$^{th}$ the width of a human hair) that required "corrective lenses" to be installed. A lesser known fact is that the onboard coprocessors of the Hubble, originally some sort of proprietary chip, were failing at an alarming rate due to radiation damage, and part of the repair mission was to replace them with radiation-hard chips (I believe they were Intel 386 processors). The coprocessor software[5] did things like point the solar panels at the sun. So all of the software needed to be re-hosted. The Hubble Repair project was my first experience with large telescopes, and I got a cool poster to put up in my office, next to the Space Station poster.

## *ICONIX: Putting the "U" in UML*

ICONIX spent about 10 years in the CASE tool business, and along the way developed one of the first Object-Oriented Analysis and Design (OOAD) tools, which we called *ObjectModeler*. Jeff Kantor had left the Space Station program and worked with me at ICONIX for a while. One of the things he did was analyze the emerging plethora of OO methodology books, looking for commonality and figuring out which of these methodologies we wanted to support in ObjectModeler. We came up with Booch, Rumbaugh, Jacobson and Coad/Yourdon, which of course includes the 3 methodologies that went into UML. We did this several years before Booch, Rumbaugh, and Jacobson got together to create UML, which happened a couple of years after I published a CD-ROM called *A Unified Object Modeling Approach*. So I like to think that Jeff and I put the "U" in UML.

After UML came out, it became clear to me that ICONIX as a tool vendor wasn't likely to remain competitive for very long. But I had developed an interesting training course that taught people how to use Booch, Rumbaugh, and Jacobson methods together, and with the advent of UML, that class became marketable. So ICONIX became a training company, focusing on our "JumpStart"

---

[3] http://www.nasa.gov/mission_pages/station/main/index.html

[4] http://www.nasa.gov/mission_pages/shuttle/shuttlemissions/hst_sm4/index.html

[5] http://hubble.nasa.gov/a_pdf/news/facts/CoProcessor.pdf

approach to starting client projects using our lightweight "unified" UML process. I also started writing books, initially *Use Case Driven Object Modeling—A Practical Approach*, with Kendall Scott, which became pretty popular.

## *Steward Observatory—The Large Binocular Telescope*

Fast-forwarding 8 or 10 years and another book written, I received a phone call one day from Tim Axelrod at the University of Arizona Steward Observatory. Tim, in his quiet, soft-spoken way, said that he had read my book and was hoping that I might be able to pop out to Tucson and run one of my JumpStart workshops because "somebody has built a very big telescope (the Large Binocular Telescope[6]) and spent 10 years working on the hardware and completely forgot about the software, and I've just been put in charge of getting the software built." Tim is now the Project Scientist for LSST Data Management.

As it happened, the LBT class was the first occasion I had to use Enterprise Architect. I figured out how to use it on the (very short) flight from Los Angeles to Tucson. We'll tell the whole story in Chapter 1, but to make a long story short, the Sparx Systems software worked like a champ and solved the shared model issues that the (then) "industry standard" tools ignored.

As a result of the positive experience we had with LBT, ICONIX joined the Sparx Systems partner program immediately after I returned from Tucson. Our initial project was to produce a multimedia tutorial titled *Mastering UML with Enterprise Architect and ICONIX Process*, followed by *Enterprise Architect for Power Users* and we rapidly switched our focus towards training Enterprise Architect users. During this time I was also writing *Extreme Programming Refactored* [7], the first of several books that I've written with Matt Stephens[8].

It was during the 5-day class for LBT, where we modeled the Observatory Control System (OCS), that my whole perspective about telescopes changed. At the time, my son's 8th grade science class was grinding an 8-inch mirror by hand, and building a telescope—so that was my frame of reference when I headed for Tucson the first time. LBT has twin primary mirrors (hence "Binocular") and they are each 8.4 meters in diameter. By comparison the Hubble has a 2.4 meter primary mirror, and the big Hale telescope at the Palomar Observatory[9], which was the world's largest for 45 years, has a 5.1 meter (200 inch) mirror.

Depending on who you talk to, LBT is either the largest optical telescope on the planet, or one of the top 2 or 3… in any event, it's BIG[10]. The Keck Observatory[11] on Mauna Kea has a 10 meter mirror, but it's made in sections. On the other hand LBT being a binocular telescope means its twin primary mirrors are working together, so I'll leave that debate to the astrophysicists.

During the week, Tim arranged for me to have a lunchtime tour of the Mirror Lab at Steward. Seeing "8.4 meters" on a page doesn't really convey the scale of these mirrors. Each mirror weighs 20 tons. The Mirror Lab[12] (which is under the football stadium at the University of Arizona) has an oven that melts 20 tons of glass in an 8.4 meter mold, and spins it until the molten glass forms a parabolic shape, then they cool it down. This saves a lot of grinding time and it's a pretty unique facility. One of the LBT primary mirrors was being polished when I was there and I got to crawl

---

[6] http://lbto.org

[7] http://www.softwarereality.com/ExtremeProgrammingRefactored.jsp

[8] http://www.softwarereality.com/MattStephens.jsp

[9] http://www.astro.caltech.edu/palomar/hale.html

[10] See http://medusa.as.arizona.edu/lbto/observatory_images.htm to get a sense of LBT.

[11] http://keckobservatory.org/index.php/about/telescopes/

[12] http://mirrorlab.as.arizona.edu/index.php

around underneath it and look at it up close. When I returned to the class, those use cases that said *Raise the Mirror*, *Lower the Mirror*, *Track an Object* etc suddenly seemed a lot more significant. It dawned on me that getting a chance to contribute (even in a small way) to the LBT software was an opportunity that not many people get, and to have had a fingertip in both the Hubble and LBT software was really quite amazing.

***Figure 1—Tim Axelrod points out a feature of one of LBT's twin primary mirrors to Jeff Kantor on one of my trips to Mount Graham. The scale is a bit deceptive, we were several floors up. You can better appreciate the size of these mirrors by noticing that there are two people down by the white railing next to the mirror.***

Thanks to Tim, I was fortunate enough to make two trips to Mount Graham, the first when the first mirror had been installed and the second time after both mirrors were up on the mountain and they were preparing to commission the telescope. The second time, they had the Observatory Control System up and running, and LBT is now observing galaxies over 100 light years away[13].

## *The First Thing I Need to do is Hire a Really Good Project Manager*

A few years after the class we did for the LBT OCS, Tim and I spoke again and he told me he had left LBT and was now Project Scientist for another telescope, called LSST (Large Synoptic Survey Telescope). LSST has a single primary mirror, also 8.4 meters in diameter[14], and a 3.2 gigapixel CCD camera[15]. However, unlike LBT, LSST is a survey telescope and will continuously sweep the entire sky rather than focusing on one spot at a time. Continuously sweeping the sky for a decade with a camera that captures 3 billion pixels in each image is what will drive the unprecedented volumes of image data that LSST will produce. So you might say that image processing was born at JPL and

[13] http://medusa.as.arizona.edu/lbto/astronomical_images.htm

[14] http://www.lsst.org/lsst/gallery/mirror-casting/Group_photo

[15] http://www.lsst.org/lsst/gallery/camera

they're perfecting it on LSST.

When I spoke with Tim, he said: "The first thing I need to do is hire a really good project manager." I knew that Jeff Kantor was working at a company in Georgia where they were grinding him half to death with overtime, and that he needed to get out of that situation. So I told Tim that he should meet my friend Jeff. They met, and that brings us to our starting point for this story.

## *Lucas, Meet Spielberg*

Before we start, I'd like to share one more story. Some years ago, Jeff and I were at a baseball game in Chicago, at Wrigley Field, and some drunk Cubs fans pointed out to us that Jeff bears a strong resemblance to Steven Spielberg (they did this by shouting "Hey, Spielberg!" at him for most of the game). A few years later, my son Rob observed to me that Tim has a resemblance to George Lucas. So it's almost as if I introduced Lucas to Spielberg, and we all know the results of that collaboration.

In all seriousness, I can't imagine two more qualified people to spearhead an effort to figure out how to analyze 20 Terabytes of image data per night, and it continues to be my privilege to work with both of them.

## Chapter 1

# The Large Binocular Telescope

"I've been reading your book," said the voice on the phone, "and I was hoping you might be able to come out to Tucson and give us one of your training workshops. I've just been put in charge of the software for a large telescope project, they've been working on the hardware for about 10 years and completely forgot about the software, and now I have to get it done in a hurry."

## JumpStarting the LBT Software

That, as close as I can remember it, was my introduction to Tim Axelrod. Tim is a soft-spoken PhD astrophysicist from Caltech, and he's responsible for my involvement in both LBT and LSST. He's one of the smartest guys that I know, and I think we share a common distaste for dogmatic approaches to software development (and for dogma in general).

This was some time during 2002, and I was in the middle of writing my third book (which was my first one with Matt Stephens), *Extreme Programming Refactored: The Case Against XP*. Matt also shares my distaste for dogma; XPR is very much a "my karma ran over your dogma" sort of book.

At the time, Extreme Programming (as dogmatic as it gets) had become quite the trendy thing to do in the industry, and the CASE tool market was dominated by expensive tools that had some significant issues. The thing that disturbed me the most about modeling tools back then was the lack of a concurrent, multi-user, server-based repository. I always felt that this, combined with a high price point, was a significant impediment to the adoption of UML modeling in the field, and in a way, added a big supporting argument to XP proponents, many of whom used XP to justify not designing their software up front or documenting anything (and then skipped the hard parts of XP).

I had heard of Enterprise Architect (EA) previously, because one of their early adopters was a fan of my first book, and suggested to Geoff Sparks that he support robustness diagrams in his software, and Geoff, who is one of the most prolific builders of high quality software that I've ever met, went ahead and did so. In effect, Sparx Systems changed the whole price/performance equation in the industry with Enterprise Architect, flipping the situation from high-price/low-performance to high-performance/low-price.

### *High Performance and Low Price Makes a Good Combination*

But back in 2002, I had never used Enterprise Architect when I got Tim's call, and as part of the preparation for the JumpStart workshop, he arranged for me to get a software license and I recall figuring out how to use it on the short flight from Los Angeles to Tucson. It seemed pretty intuitive, and my plans to spend the evening getting acquainted with the software proved largely unnecessary.

#### Modeling Tip: Good tools make a big difference

Don't settle for anything less than a modeling tool that's affordable, easy to use, and supports concurrent, multi-user modeling. Good tools like Enterprise Architect make a *big* impact on your project.

I was interested in trying Enterprise Architect because it seemed to address my two biggest issues with modeling tools at the time; price point (at that time, an Enterprise Architect license was $99 and it's still amazingly affordable) and an out-of-the-box multi-user server based repository. But

when Tim told me of his plans to run it on Linux machines using a Windows emulator, and to keep the repository for the lab on a networked machine in another building (we were in the Steward Observatory on the University of Arizona campus), I was less than enthused, because I thought we were running a significant risk of JumpStarting the project into a brick wall.

## *A Push in the Right Direction*

JumpStart is the name of our week-long training class in ICONIX Process where we work a client's real project as the lab example. Clients like Tim hire us when they need to get a software project moving in a hurry. This is a trickier process than it might first appear, because if we get the project moving in the wrong direction, it creates big problems for the client, and for us. At ICONIX, we're invested in our client's success.

Our JumpStart classes are about 20% lecture (we try to keep it simple) and 80% lab, and the lab is not a textbook example, but the real project—and most of the time a critically important project to the client. So anything that puts the lab session at risk of not going well is something that I try to avoid like the plague. I explained my concerns about the network configuration to Tim, he understood, and proposed that we try it, and if it proved problematic we'd quickly switch to plan B.

### Modeling Tip: Not everything is in the UML spec

There are several really useful extensions to UML that make a big difference to a successful project. For example, **Requirements**, and **Screens** are not part of the UML, but are essential to a successful project. Easy-to-use **document generation** is also important. **Reliability** of a modeling tool is also very important.

As the week progressed, I became increasingly impressed with the capability and reliability of the Sparx Systems Enterprise Architect software. It was easy to use, never crashed once during the entire week, and had lots of useful features like built-in document generation and extended UML with important things like Requirement and Screen elements.

Having spent a decade building CASE tools, I knew a quality modeling tool when I used one, and ICONIX joined the Sparx partner program immediately after my return from Tucson. Thus began a long and fruitful association with the folks at Sparx, who continue to implement my ideas about improving process within Enterprise Architect.

### ICONIX and Sparx Systems continue to collaborate on extensions to UML and Enterprise Architect

ICONIX has developed a "process roadmap" that details all the steps of ICONIX Process on a set of activity diagrams, and a method for driving test cases (and JUnit/NUnit test code) from UML models, called "Design Driven Testing" (DDT). Sparx Systems provides excellent support for these ICONIX extensions in Enterprise Architect. DDT is supported in the "Agile/ICONIX add-in" from Sparx Systems. Synergy between process and tools is a wonderful thing.

## *There's Big, and Then There's BIG*

The highlight of my week in Tucson was a Thursday afternoon tour that Tim arranged at the Steward Observatory Mirror Lab. When we arrived, they were in the process of polishing the first of LBT's two 20-ton mirrors.

***Figure 1. LBT's twin primary mirrors each weigh 20 tons.***

The Mirror Lab at Steward is actually under the bleachers of the football stadium on the University of Arizona campus. This has always reminded me of Enrico Fermi's nuclear reactor under the football stadium at the University of Chicago. If you ever find yourself in Tucson, I highly recommend taking a tour of the Mirror Lab.[16] They have a rotating oven that spins 20 tons of molten glass into a parabolic shape in an 8.4 meter mold. You've probably never seen one before.

## *Size Matters*

While taking this tour, my jaw dropped open when I realized the true scale of LBT. It takes a big telescope to image galaxies more than 100 light-years away (especially through the atmosphere, which requires the use of adaptive optics to correct for distortion), and with that size comes complexity.

For example, because glass is not perfectly rigid, when you tilt a 20-ton mirror vertically, gravity affects the curvature of the mirror. For this reason the mirror's "skin" has devices called actuators attached to the back of it, which can either push out or pull in under software control to maintain the proper curvature of the telescope mirrors. Back in the JumpStart lab that I was running, some of my students were writing the use cases to control these actuators. Others were doing mirror pointing, aiming, and tracking. Another team (which as I recall included Tim's wife Robyn, who is now the QA Manager on LSST) was doing Telemetry use cases.

I remember thinking, as I walked back to the classroom from the football stadium, that I was pretty lucky to be involved with this project, and that between LBT and the week I had once spent working with the Hubble software folks, I was doubly lucky. I resolved to stay in touch with Tim and hoped that I might get to visit LBT one day. So far I've made it up the mountain twice, and I hope someday around 2015 I might get to visit Chile to have a look at LSST.

[16] http://mirrorlab.as.arizona.edu/MISC.php?navi=tours

# Stretching ICONIX Process: LBT's Observatory Control System

The Observatory Control System (OCS) for LBT was by no means a "classical" use case system, but rather a system with many realtime embedded aspects, that was controlled by an operator through a relatively simple GUI.

## *There's a GUI, but That's Not the Hard Part*

During the LBT JumpStart workshop in Tucson, the strategy that Tim and I adopted was to use the operator use cases as an organizing framework for the model, even though we knew that the user interface was not where the real complexity of the LBT software lived.

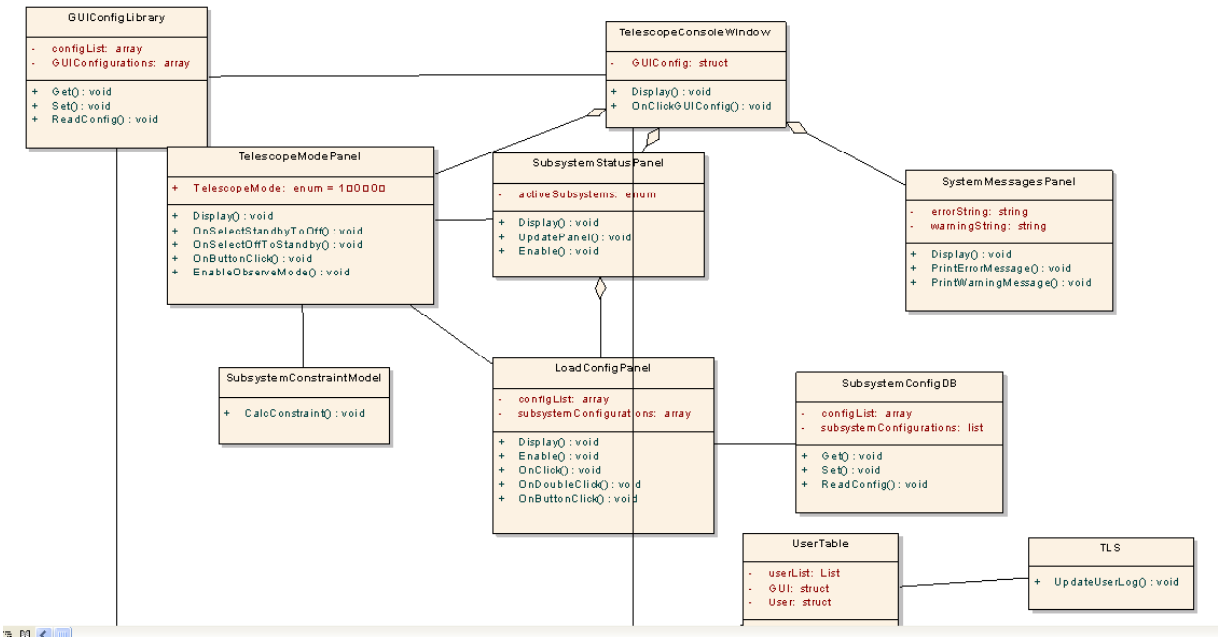



***Figure 2a. LBT's Observatory Control System GUI is not where the complexity of the OCS software resides.***

***Figure 2b. LBT's Observatory Control System does have some legitimate use cases.***

Starting from the GUI, we were able to use this approach to identify “embedded operational scenarios” that were invoked from the operator-centric use cases, and recognized that within these embedded scenarios (which involve software controlling hardware), there would be some deviation from “classical” use-case-driven ICONIX Process.

## *Modeling Embedded Realtime Systems With Scenarios*

LBT, although a behemoth among telescopes, behaves in a similar manner to most traditional telescopes in that the telescope is aimed at a celestial object, and has to track that object while the earth rotates and photos are taken. So there are various commands issued by the telescope software that drive the servo-motors and other hardware that move the mirrors, the telescope (and actually the entire building that encloses it, which swivels on giant rollers) around.

While moving the telescope isn’t a use case intensive process, it’s fairly straightforward to identify scenarios of operation such as: “Point the telescope at a preset location on the sky” and we can model these scenarios with use cases, with an understanding that the rules for modeling use cases can be bent a little to accommodate the realtime nature of the software.

### Modeling Tip: Scenario techniques work well for realtime embedded systems

Use case driven development is a scenario-based approach. It stretches reasonably well to realtime embedded systems when you can easily identify “control scenarios”. Some of the “use case rules” can be bent when there is no user interface.


***Figure 3a—The OCS has plenty of scenarios, even though those scenarios aren’t classical use cases.***

So the OCS isn't really too big of a stretch for ICONIX Process because it's straightforward to identify scenarios. In general, our experience over the years has been that ICONIX Process handles realtime embedded systems pretty well. By contrast, as you'll see later, the LSST Data Management software is almost purely algorithmic in nature, making for a much bigger stretch.

Even though the scenarios are fairly simple, like moving the telescope to a pre-set position on the sky, the software within them needs to be designed pretty carefully, because software failures can get pretty costly with hardware on the scale of LBT.

***Figure 3b. Aiming the telescope involves some significant effort (Jeff Kantor explains it to my son Rob)***

## *Modeling Realtime Systems: It's OK to Bend the Rules a Little*

I had very little interaction with Tim's group after the initial JumpStart workshop until we started working together on LSST. This isn't particularly unusual with our engagements at ICONIX, as our five-day workshops are generally pretty successful at getting the UML model organized, and giving the modeling effort a big push in the right direction. But I wasn't around to guide the modeling effort, and credit for the project's success goes entirely to Tim.

Much of the benefit realized from a modeling effort is simply having a shared medium with which people on the project can communicate ideas about the design. So it's much better to do a model, and bend the rules a little, than not to do a model at all.

A good UML modeling tool like Enterprise Architect provides a shared workspace that allows different team members simultaneous access to the model, and, while providing assistance in checking semantics, is not overly intrusive about adherence to those semantics.

### Modeling Tip: A shared workspace facilitates communication

UML modeling is a collaborative activity, whose purpose is to create a shared understanding of the problem and the solution. A good modeling tool that provides a shared workspace makes this communication more effective.

Tim, like many other experienced “algorithm developers” is quite familiar with using data flow diagrams to decompose complex processes, and, without me being around to harass him about staying “pure” object-oriented in his thinking, he occasionally expressed some modeling ideas in DFD-like form.

Of course, I’m no stranger to DFDs and functional decomposition myself. Back in the early days of ICONIX we built and sold a DFD editor CASE tool, and I taught many classes in Structured Analysis and Design.[17] So I recognize them when I see them, and I understand the difference between a functional decomposition with DFDs and a scenario decomposition with use cases.

I’m a firm believer that a scenario decomposition is better, in most cases, especially if the implementation is going to be object oriented, because we know how to reliably drive good OO designs from scenarios.

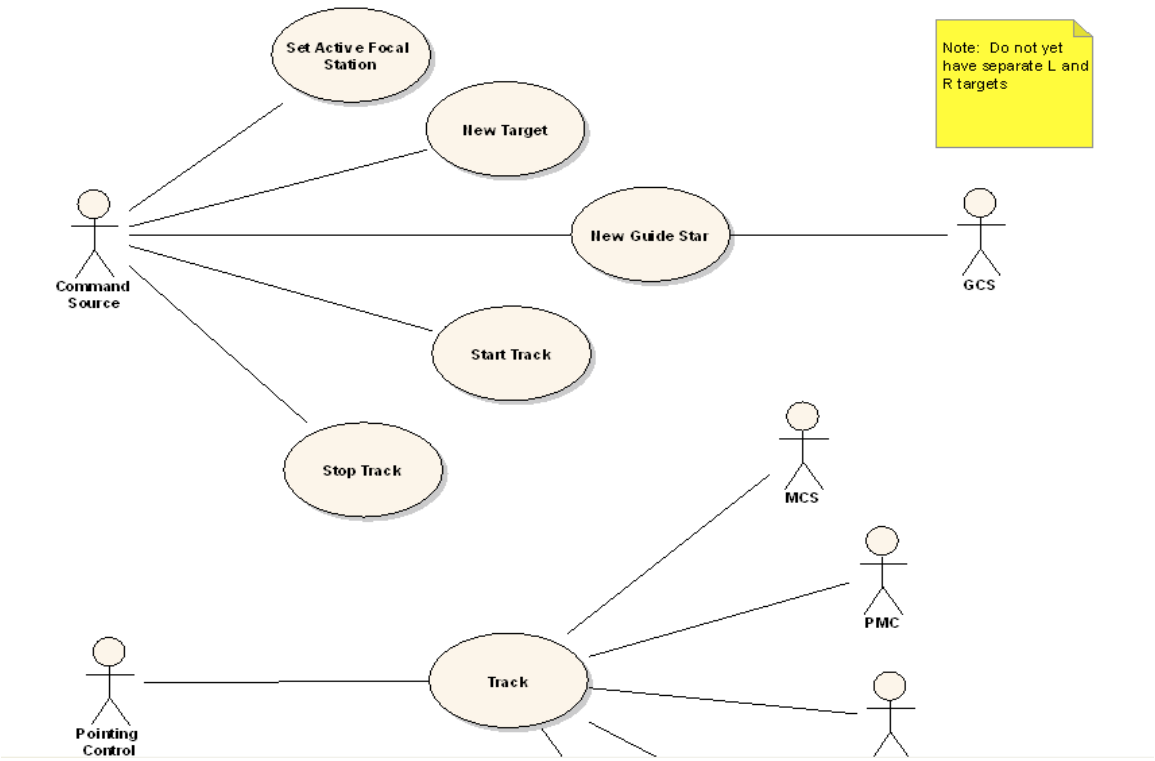



***Figure 4. A few of the LBT use case diagrams have a distinct functional decomposition flavor to them***

## *Semantic Consistency is a Good Thing, but It’s Not the Only Thing*

Successful developers and project managers like Tim (who does both) recognize that models exist to get the job done, and getting the job done comes before semantic purity in modeling.

Since it was obvious during the JumpStart workshop that the most complex part of the LBT software didn’t involve very much user interaction, I encouraged Tim to bend the “ICONIX use case

[17] This included the class I taught at NASA Goddard to the folks who were re-hosting the Hubble co-processor software.

modeling rules" as he needed to, and use the UML model as a communication vehicle for him to communicate the design he wanted to his development team.

While the robustness diagram below (which consists almost entirely of "controller" objects—those are the circles with the arrow) probably wouldn't win a prize for semantic consistency with the textbook-specified notation,[18] it does succeed in showing the conceptual design of a group of "controller objects" interacting together to point the telescope at a desired spot on the sky (Celestial Position).

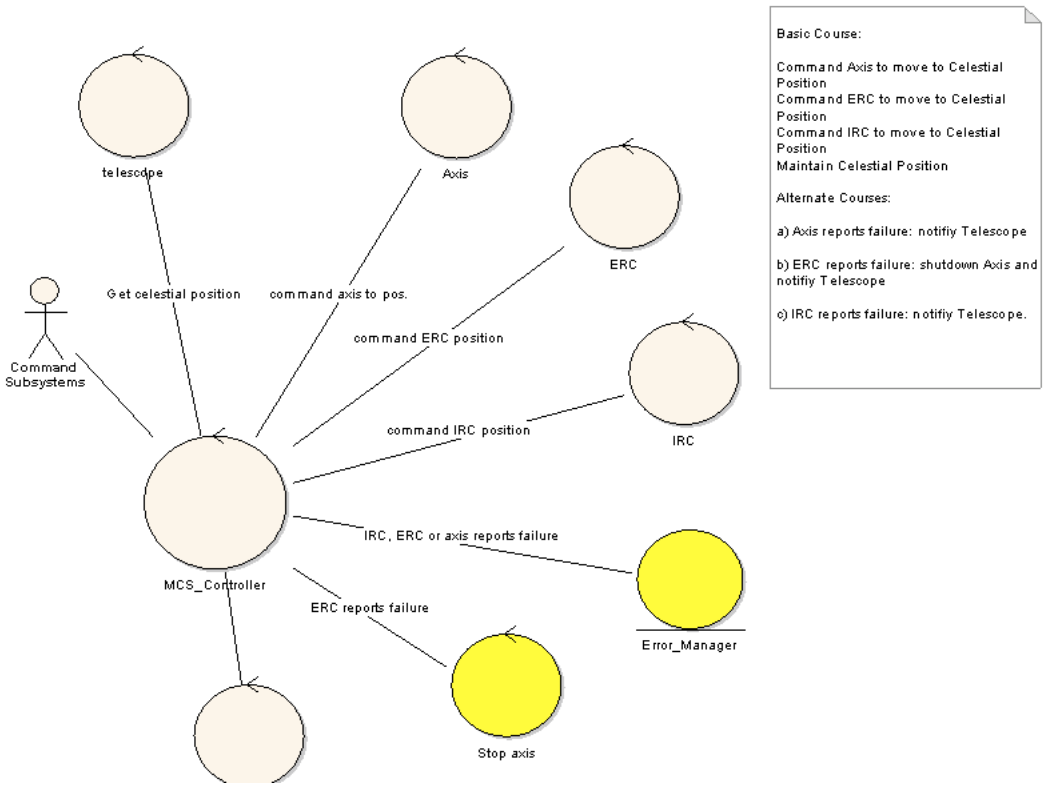



***Figure 5a - LBT's Observatory Control System is just that—a control system that uses software to control hardware. Normal "use case modeling rules" don't always apply.***

It's a simple scenario, but you really wouldn't want to screw it up… refactoring the LBT hardware could get expensive. So "Test-Driven-Developing" the stopMirrors method by first making a unit test fail and running the mirrors into the wall might not be the ideal approach here.

[18] The controllers represent verbs, or actions, in the emerging software design.

***Figure 5b. Tim's controller objects are hauling 40 tons of mirrors (and a lot of steel) around.Unambiguous Communication is the Only Option***

ICONIX Process makes heavy use of sequence diagrams to describe a software design at an object/message level. While the process systematically drives sequence diagrams from use cases, UML sequence diagrams are still highly useful even when there are no use cases, as the "Status Request" diagram in Figure 6 shows.

By specifying the design at this level, Tim was able to unambiguously communicate to his developers exactly what the LBT software needed to do. To the best of my knowledge, this design is still the basis of the software that's running the LBT Observatory today.

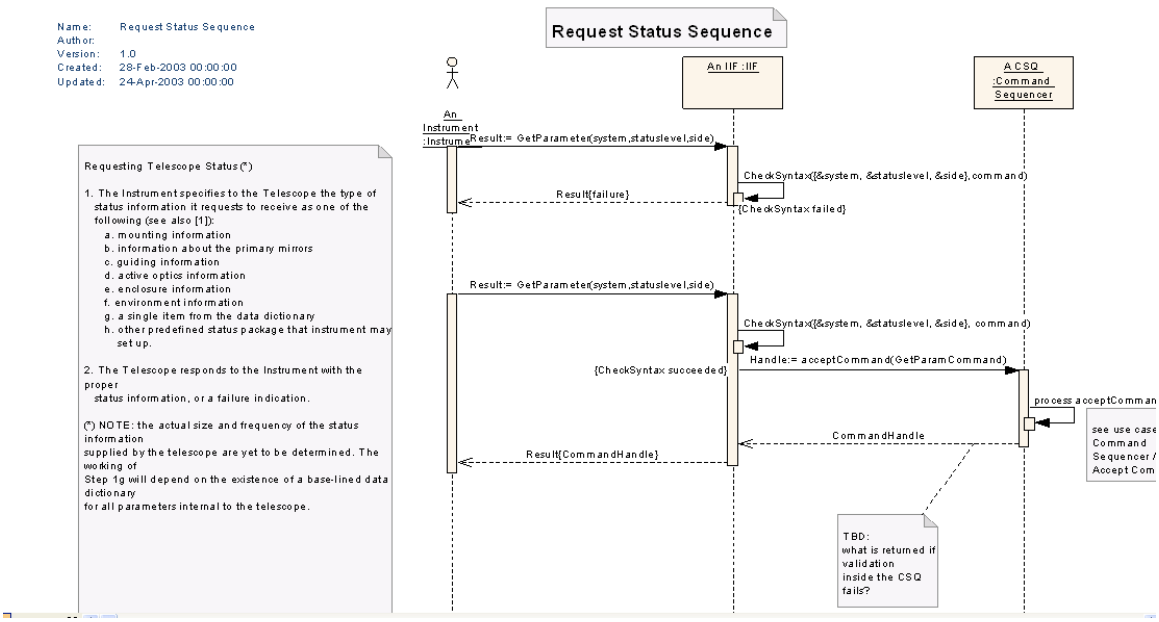



***Figure 6. Unambiguous communication between management and developers is essential for project success.***

## Chapter 2

# What's a Large Synoptic Survey Telescope… (and why do we need one)?

Many thanks to the folks at LSST Corporation[19] for allowing us to use some content (virtually all of this chapter) from the LSST.org website!

***In its first month of operation, the LSST will see more of the Universe than all previous telescopes combined.*** Its rapid-fire, 3 billion pixel digital camera will open a movie-like window on objects that change or move; its superb images will enable mapping the cosmos in 3D as never before. Surveying the entire sky every few days, LSST will provide data in real time to both astronomers and the public. For the first time, everyone can directly participate in our journey of cosmic discovery.

***Figure 1. Suzanne Jacoby with the LSST focal plane array scale model. The array's diameter is 64 cm. This mosaic will provide over 3 Gigapixels per image. The image of the moon (30 arcminutes) is placed there for scale of the Field of View. (Image credit: LSST Corporation)***

LSST's main science areas include Dark Matter/Dark Energy, Near Earth Objects, The Outer Solar System, Mapping the Milky Way, and the Transient High Energy Universe. LSST will produce and distribute a set of data products, including Images, Catalogs, and Alerts, that astronomers will use to explore these science areas.

[19] LSST is a public-private partnership. Design and development activity is supported in part by the National Science Foundation. Additional funding comes from private foundation gifts, grants to universities, and in-kind support of Department of Energy laboratories and other LSST Member Institutions. The project is overseen by the LSST Corporation, a non-profit 501(c)3 corporation formed in 2003, with headquarters in Tucson, AZ.

## *Synoptic = All Seeing*

The 8.4-meter LSST will survey the entire visible sky deeply in multiple colors every week with its three-billion pixel digital camera, probing the mysteries of Dark Matter and Dark Energy, and opening a movie-like window on objects that change or move rapidly: exploding supernovae, potentially hazardous near-Earth asteroids, and distant Kuiper Belt Objects.

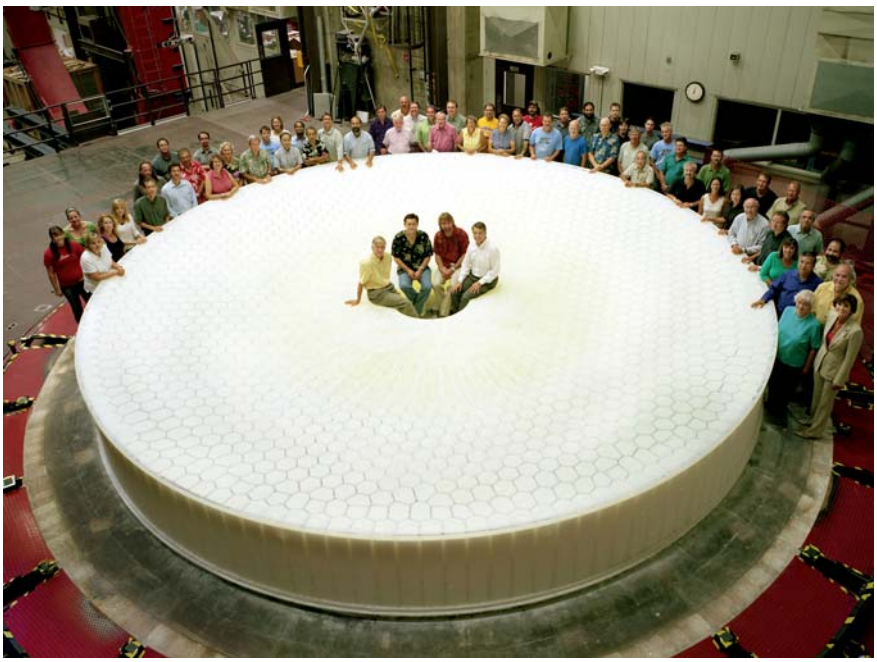


***Figure 2. LSST Group Picture. Members of the team building the LSST, a large survey telescope being built in Northern Chile, gather to celebrate the successful casting of the telescope's 27.5ft-diameter mirror blank, August 2008. (Image credit: Howard Lester / LSST Corporation)***

Plans for sharing the data from LSST with the public are as ambitious as the telescope itself. Anyone with a computer will be able to fly through the Universe, zooming past objects a hundred million times fainter than can be observed with the unaided eye. The LSST project will provide analysis tools to enable both students and the public to participate in the process of scientific discovery.

We've summarized some science information from the LSST website below.[20]

## *Dark Matter*

About 90% of the Universe is dark—we can't see it except through its gravitational pull. Although this was suspected more than 60 years ago, we are just now in a position to explore the dark matter in large areas of the Universe through a technique called weak gravitational lensing.

## *Dark Energy*

Dark energy is a mysterious force that is accelerating the expansion of the universe. The expansion has slowed the clustering of dark matter, one of the universe's main building blocks.

---

[20] You can find out much more about the science enabled by LSST at http://www.lsst.org/lsst/science

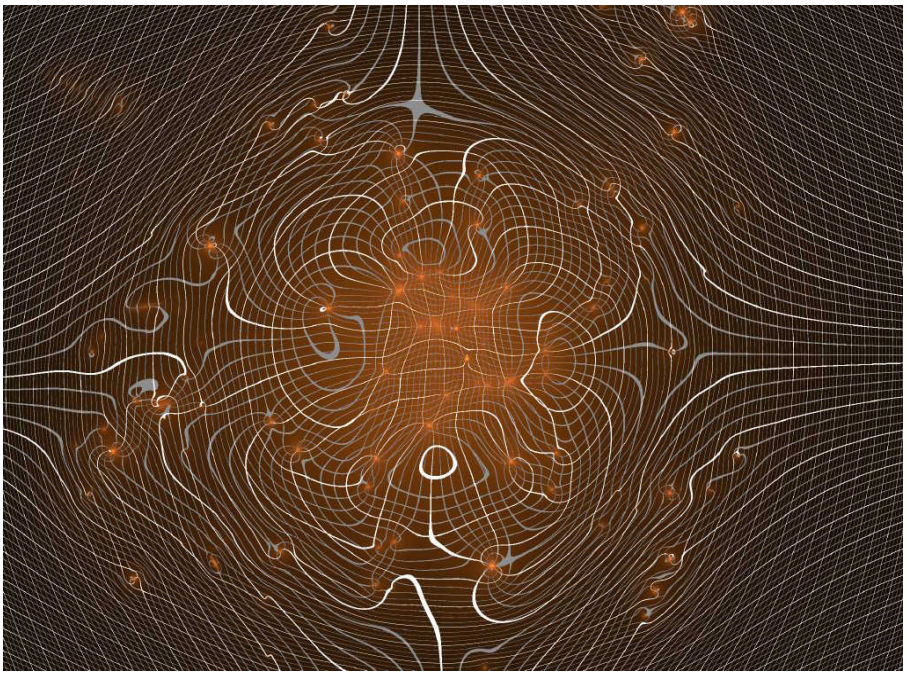


***Figure 3. Space-time warp: the detailed mass distribution in the cluster CL0024 is shown, with gravitationally distorted graph paper overlaid. This detailed dark matter distribution can be used to constrain theories of dark matter.***

## *Near-Earth Objects*

The questions of how the solar system came into being and how life on Earth might end are two of the "nearest" astronomical issues to mankind. Answering the former requires as an initial step a census of the solar system. But identifying objects in the outer solar system has proven to be a difficult challenge despite some recent success. Answering the latter, at least in the case of a possible asteroid impact, is not strictly a scientific question but might be the most important contribution astronomy makes to life on Earth. The LSST would make uniquely powerful contributions to the study of near-Earth objects (NEOs).

## *The Outer Solar System*

The LSST has been identified as a national scientific priority in reports by diverse national panels, including several National Academy of Sciences and federal agency advisory committees. Investigating the extent and content of the solar system beyond Neptune requires a detailed understanding of the Kuiper Belt, which in turn will lead to an improved understanding of the link between our Solar System and those being discovered around other stars. The study of the outer solar system will not only clarify the formation history of our solar system, but will point the way to how other solar systems may form and how star formation in general proceeds.

## *New Light On The Transient High-Energy Universe*

The LSST will explore a new universe of transient sources of luminosity. We already have hints of unusual and violent events at great distances. Recently, a combination of observations with the Beppo-SAX satellite and two ground-based optical telescopes has produced proof that gamma-ray burst sources are at high redshift and so must be the most luminous objects in the universe. This must be new physics or a manifestation of physics in new and extreme conditions: attempts at a theoretical explanation using known physics have encountered major difficulties. More generally, our monitoring of and knowledge of transient sources in the cosmos is in its infancy.

## *Mapping the Milky Way*

The LSST is ideally suited to answering two basic questions about the Milky Way Galaxy:

* What is the structure and evolution history of the Milky Way?

* What are the fundamental properties of all the stars in the neighborhood of our Sun?

LSST will produce a massive and exquisitely accurate data. Compared to the the best currently available optical survey, Sloan Digital Sky Surey (SDSS) LSST will cover an area more than twice as large, making hundreds of observations of the same region (instead of just one or two) and each observation will be about 2 magnitudes deeper. The coverage of the plane of our Galaxy will yield data for numerous star-forming regions, and even penetrating through the interstellar dust layer. LSST will detect of the order of 10 billion stars.

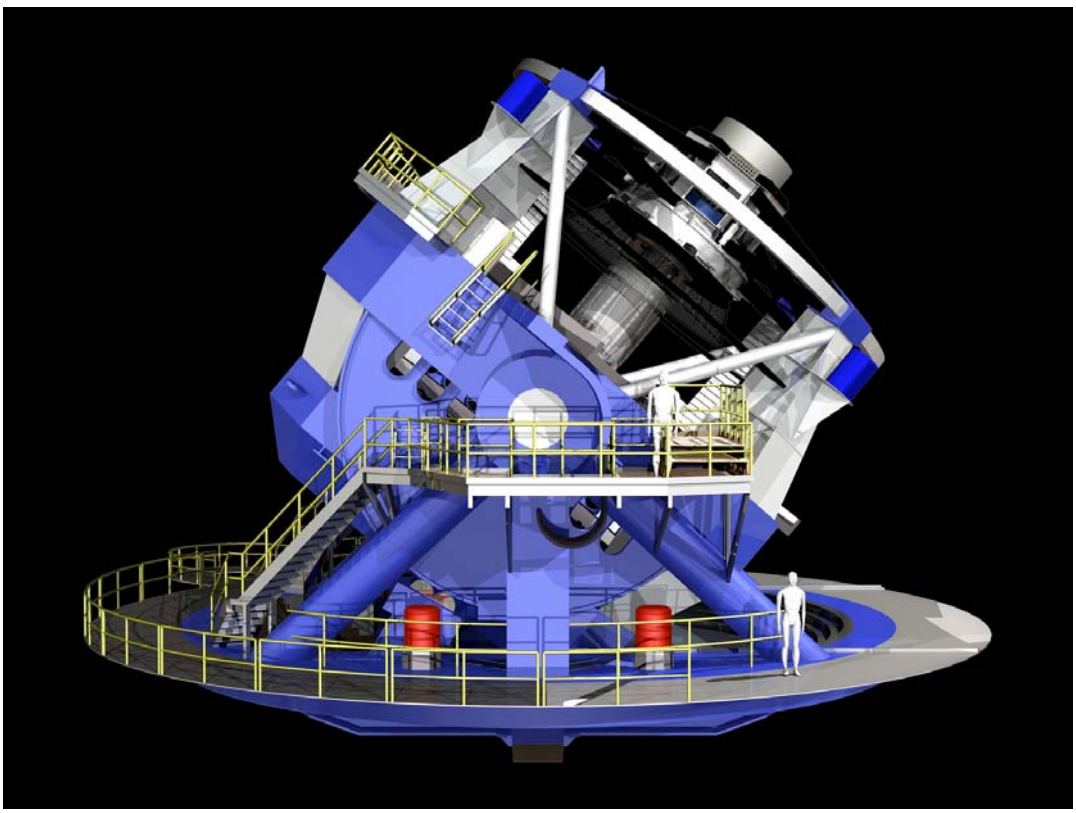


***Figure 4. The 8.4-meter LSST will use a special three-mirror design, creating an exceptionally wide field of view and will have the ability to survey the entire sky in only three nights. (Image credit: LSST Corporation)***

## *LSST Data Products*

The LSST data products are organized into two groups, distinguished by the cadence with which they are generated. Level One products are generated by pipeline processing the stream of data from the camera system during normal observing. Level One products are therefore being continuously generated and updated every observing night. This process is of necessity highly automated, and must proceed with absolutely minimal human interaction. Level One data products are divided into Image, Catalog, and Alert categories, as shown in Figure 5.

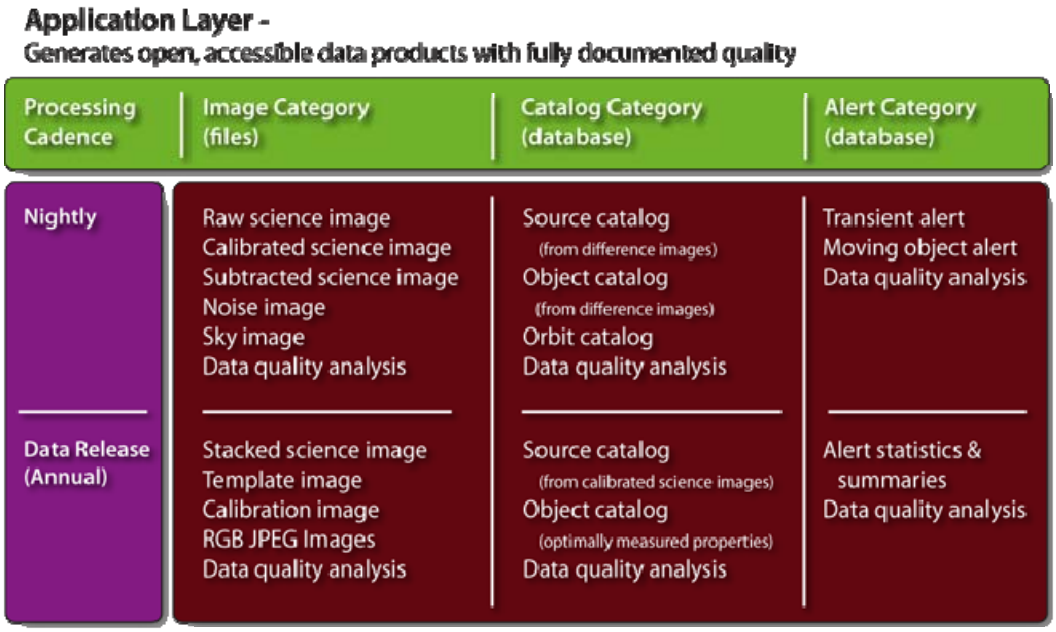



***Figure 5. LSST Data Products***

## *High Level Requirements*

The Functional Requirement Specifications of the data management system include the following:

1) The incoming stream of images generated by the camera system during observing is processed to generate and archive the nightly data products

* Raw science images
* Catalog of variable sources
* Transient alerts

2) Periodically the accumulated nightly data products are processed to

* Generate co-added images of several types
* Optimally measure the properties of fainter objects
* Perform astrometric and photometric calibration of the full survey object catalog
* Classify objects based on both their static properties and time-dependent behavior

Level Two products, including calibration images, co-added images, and the resulting catalogs, are generated on a much slower cadence, and their release will be driven by data quality assessments. Although many of the steps that generate Level Two products will be automated, they need not all be so, and significant human interaction may be tolerated.

## *LSST Data Management Pipelines*

The pipelines process the images to produce the catalogs, which are then made accessible to the community via open interfaces in a Virtual Observatory model. Since new data is being collected nightly throughout the LSST's 10-year survey period, and scientific algorithms will evolve during this time frame, significant re-processing will occur. This must be taken into account in sizing the LSST technology resources and making the LSST middleware easily extendable.

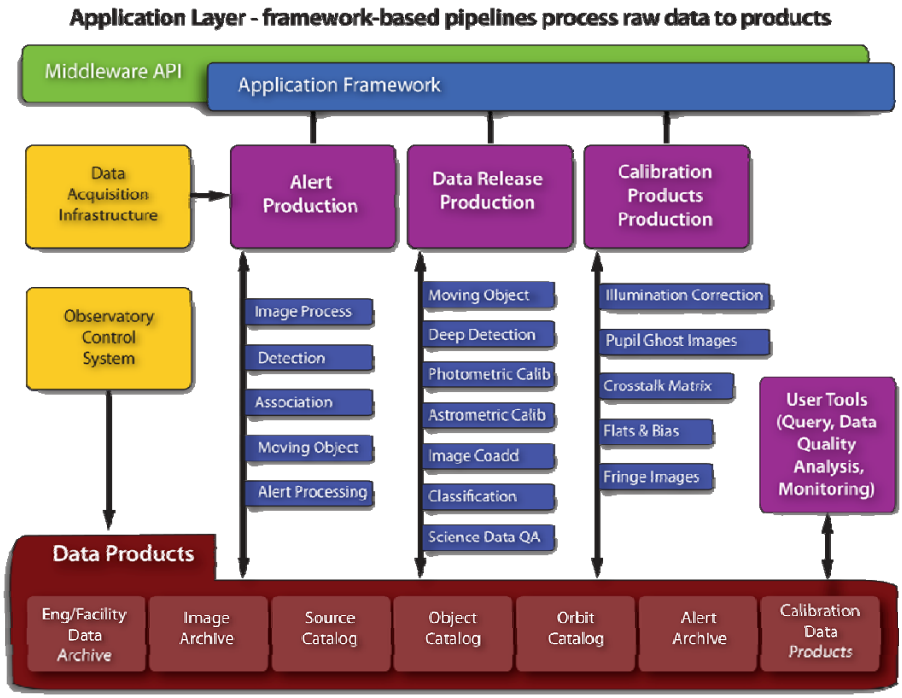



***Figure 6. LSST's Image Processing Pipeline software is the topic of this book***

The pipelines can be categorized in terms of either being near real-time or "static", depending on how stringent the associated latency and throughput deadlines are. Examples of near real-time pipelines include data quality assessment providing feedback to telescope operations, instrument calibration processing, and time-domain or transient science analysis. These pipelines execute at the mountain base facility in order to avoid the latency associated with long-haul transmission of the raw data. The static pipelines include deep image co-addition, weak lensing shear processing needed for dark energy - dark matter science, and object cataloging. These pipelines execute at the archive center. The archive center also performs re-processing of the near real-time pipelines.

## Chapter 3

# Data Challenges: From 0 to 20 Terabytes a night

The LSST website[21] summarizes the challenge facing Jeff and Tim very nicely:

> *The science archive will consist of 400,000 sixteen-megapixel images per night (for 10 years), comprising 60 PB of pixel data. This enormous LSST data archive and object database enables a diverse multidisciplinary research program: astronomy & astrophysics; machine learning (data mining); exploratory data analysis; extremely large databases; scientific visualization; computational science & distributed computing; and inquiry-based science education (using data in the classroom). Many possible scientific data mining use cases are anticipated with this database.*
>
> *The LSST scientific database will include:*
>
> ** Over 100 database tables*
>
> ** Image metadata consisting of 700 million rows*
>
> ** A source catalog with 3 trillion rows*
>
> ** An object catalog with 20 billion rows each with 200+ attributes*
>
> ** A moving object catalog with 10 million rows*
>
> ** A variable object catalog with 100 million rows*
>
> ** An alerts catalog. Alerts issued worldwide within 60 seconds.*
>
> ** Calibration, configuration, processing, and provenance metadata*

## *Sky Movies—Challenges of LSST Data Management*

The Data Management (DM) part of the LSST software is a beast of a project. LSST will deal with unprecedented data volumes. The telescope's camera will produce a stream of individual images that are each 3.2 billion pixels, with a new image coming along every couple of minutes.

In essence, the LSST sky survey will produce a 10 year "sky movie". If you think of telescopes like LBT producing a series of snapshots of selected galaxies and other celestial objects, and survey telescopes such as Sloan producing a "sky map"[22], then LSST's data stream is more analogous to producing a 10 year, frame-by-frame video of the sky.

## *LSST's Use Cases Will Involve Accessing the Catalogs*

LSST's mandate includes a wide distribution of science data. Virtually anyone who wants to will be able to access the LSST database. So parts of the LSST DM software will involve use cases and user interfaces for accessing the data produced by the telescope. Those data mining parts of the software will be designed using regular use-case-driven ICONIX Process, but they're not the part of the software that we're concerned with in this book.

---

[21] www.lsst.org/lsst/science/technology

[22] www.sdss.org/dr7/products/index.html

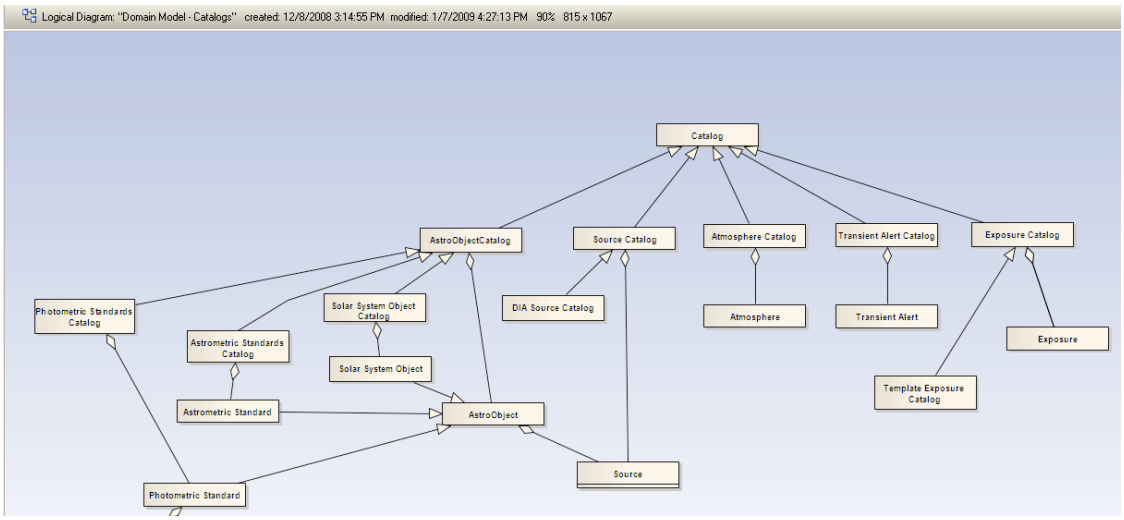



***Figure 1—LSST will produce many catalogs, which will be widely accessible by the public***

## *Lots of Brains and a Fair Amount of Time*

There are a couple of things that Jeff and Tim do have working in their favor: plenty of brains (not only their own, but a widespread and largely brilliant team of astrophysicists that are experts on various pieces of the problem), and a fair amount of time (LSST is scheduled to go operational in 2015, and is currently in an R&D phase). However, it's safe to say that most of the astrophysicists on the team wouldn't consider themselves software engineers, although most of them are programmers. In this situation, a good strategy is to make extensive use of rapid prototyping (in this case algorithm development via prototyping) in addition to the UML modeling. So a two-pronged strategy of prototyping and modeling has been underway on LSST for a few years now.

The LSST prototyping strategy involves annual *Data Challenges* (see Figure 2). These Data Challenges are development efforts with a limited functional and performance scope, and somewhat relaxed modeling requirements. During LSST's Construction Phase, prototyping will switch to incremental development, where the actual system will be developed in a sequence of incremental releases, and somewhat more modeling will be expected.


***Figure 2. LSST's R&D Phase is being conducted as a series of Data Challenges***

In the next Chapter, we'll take you into a modeling workshop that I helped to conduct, for Data Challenge 3 (DC3), where the need for some process tailoring became obvious. But first let's look at some of the challenges faced by the LSST modeling team.

## *Let Coders Write Code*

A very important part of the development strategy for LSST is to implement key algorithms and validate them, in parallel with developing an overall UML model for the LSST software. This is important for many reasons, among them (as you'll see in the next chapter) that lots of programmers prefer writing code to modeling in UML[23]. But in order to make all of these independently developed prototype pieces work together when LSST gets to Construction, we need to be able to import the prototype code into the model. Enterprise Architect's reverse engineering capabilities make this work quite smoothly (see Figures 2 and 3).

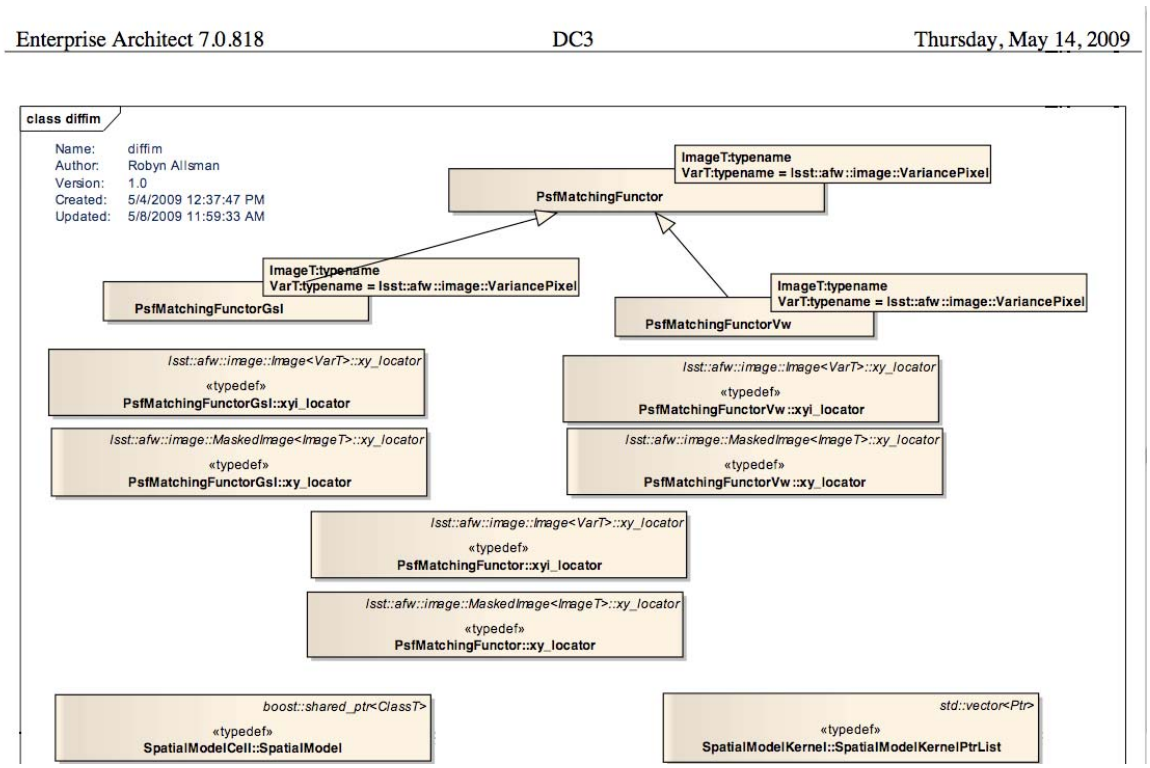



***Figure 2. LSST's schedule allows an R&D Phase that's long enough to allow lots of prototyping.***

Enterprise Architect can inhale code as fast as it can be written, so Jeff and Tim are able to allow programmers to prototype various critical pieces of the data management software during the Data Challenges, with full confidence that when the time comes to begin work on the real production software, the prototype code can be "mined" as needed and made to work within the overall system architecture.

### Modeling Tip: Combine modeling with prototyping

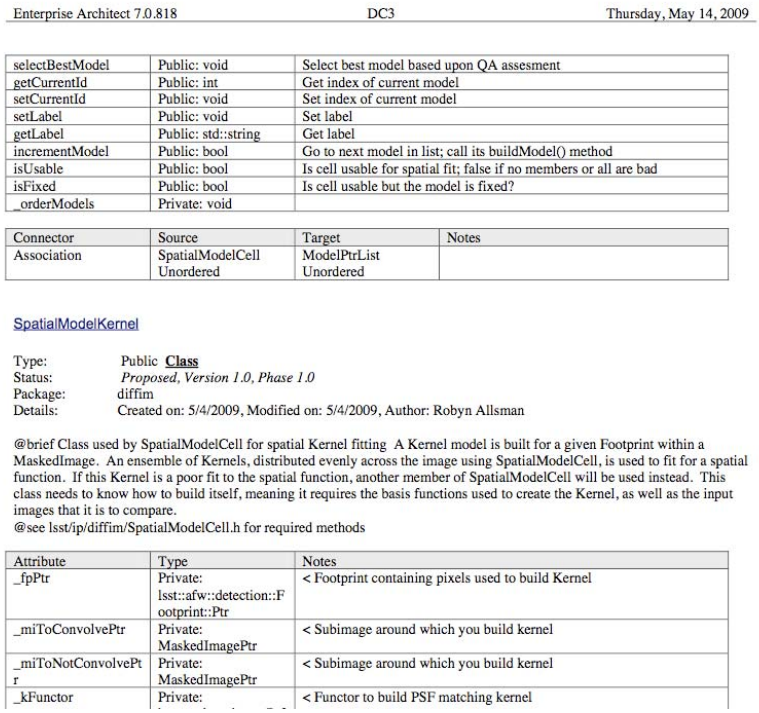
***Don't*** be afraid to build prototype code, but

***Do*** make sure to reverse-engineer it into your production model.

***Make sure*** your modeling tool is effective at reverse engineering.

Combining modeling with prototyping is an effective ***risk mitigation strategy***.

[23] We know this will come as a shock to many readers.



Enterprise Architect 7.0.818 DC3 Thursday, May 14, 2009

| | | |
|---|---|---|
| selectBestModel | Public: void | Select best model based upon QA assesment |
| getCurrentId | Public: int | Get index of current model |
| setCurrentId | Public: void | Set index of current model |
| setLabel | Public: void | Set label |
| getLabel | Public: std::string | Get label |
| incrementModel | Public: bool | Go to next model in list; call its buildModel() method |
| isUsable | Public: bool | Is cell usable for spatial fit; false if no members or all are bad |
| isFixed | Public: bool | Is cell usable but the model is fixed? |
| _orderModels | Private: void | |

| Connector | Source | Target | Notes |
|---|---|---|---|
| Association | SpatialModelCell Unordered | ModelPtrList Unordered | |

SpatialModelKernel

Type: Public **Class**
Status: *Proposed, Version 1.0, Phase 1.0*
Package: diffim
Details: Created on: 5/4/2009, Modified on: 5/4/2009, Author: Robyn Allsman

@brief Class used by SpatialModelCell for spatial Kernel fitting A Kernel model is built for a given Footprint within a MaskedImage. An ensemble of Kernels, distributed evenly across the image using SpatialModelCell, is used to fit for a spatial function. If this Kernel is a poor fit to the spatial function, another member of SpatialModelCell will be used instead. This class needs to know how to build itself, meaning it requires the basis functions used to create the Kernel, as well as the input images that it is to compare.
@see lsst/ip/diffim/SpatialModelCell.h for required methods

| Attribute | Type | Notes |
|---|---|---|
| _fpPtr | Private: lsst::afw::detection::Footprint::Ptr | < Footprint containing pixels used to build Kernel |
| _miToConvolvePtr | Private: MaskedImagePtr | < Subimage around which you build kernel |
| _miToNotConvolvePtr | Private: MaskedImagePtr | < Subimage around which you build kernel |
| _kFunctor | Private: | < Functor to build PSF matching kernel |

***Figure 3. Enterprise Architect's reverse engineering capabilities allow complex algorithms to be prototyped, then incorporated into the model.***

## *Distributed Development Team*

A further complication in modeling LSST's DM software is that the development is being done in multiple, independent, geographically distributed research organizations—places like Princeton, Stanford, University of Washington and Caltech, among others. This places some significant requirements on the modeling tools that are used by the project. Fortunately, Enterprise Architect supports distributed development using version-controlled packages, allowing someone at Caltech or Stanford to check out a package, add information to the model, and check it back in to the central repository in Tucson.

### Modeling Tip: Manage distributed teams with version control

***Make sure*** that your modeling tool allows you to manage large projects by providing interfaces to version control system and allowing check-in/check-out on a package by package basis.

## *Here Today, Gone Tomorrow*

My involvement with LSST has been ongoing but sporadic. Due to budget constraints and the incremental nature of the various Data Challenges (and to relaxed modeling requirements during LSST's R&D phase), we never did a full 5-day LSST JumpStart workshop as we did with LBT. So my involvement has been more of an occasional visitor to the various workshops on the project, on the order of a few days a year for the last few years.

My approach has always been to show up when requested, and do the best I can to help out. In some cases this has been as simple as being "Tim's personal modeling assistant". In other cases it has involved strategy sessions with Jeff and Tim about the overall approach, and, as you'll see in the next chapter, tailoring ICONIX Process to make it work better for LSST. We'll spend the rest of the book exploring the why and how of the result: **ICONIX Process for Algorithms**.

## Chapter 4
# Tailoring ICONIX Process for Algorithm Development

Those of us who define software processes do our best to make a process as generally useful as possible. However, no process can be a perfect fit for every project, and while ICONIX Process has been useful to thousands of software projects over the last 15 years, there's no disgrace involved in tailoring it to fit a specific problem, or class of problems. In this case, tailoring it to fit LSST Data Management resulted in a process that's applicable to all kinds of algorithm-intensive projects.

The need to consider tailoring ICONIX Process to better fit LSST's data management software can be traced back to an email discussion between Jeff, Tim, and I back in August of 2008, in preparation for the DC3 Design workshop which was held at Caltech.

## *The Pasadena DC3 Workshop—How the Need for Process Tailoring Surfaced*

The DC3 workshop had a very specific purpose: to bring key players from the LSST team together to brainstorm and capture ideas for Data Challenge 3 in the model. Jeff was much less concerned with the precise form that the information capture took, than with getting as many algorithms captured as possible in a short time period, because he was on the hook to produce estimates for the upcoming Preliminary Design Review (PDR).

Digging through my old emails finds this discussion, beginning with an email from me to Jeff and Tim regarding how we wanted to approach the workshop:

> *Jeff and Tim*
>
> *Here are a few things that I believe help to define "the right way" to do these use case / UML models. If left to my own devices I would teach the workshop attendees these principles, but you need to give me marching orders as to whether you want me to do this or not. I think one of the key values of robustness analysis is that it helps to ensure "well formed" or "proper" use cases. To be honest I don't know what the robustness diagrams will look like if we don''t follow these guidelines*
>
> *- use case diagrams are not Data Flow Diagrams (a use case doesn't have INPUTS: OUTPUTS: and drill down to see lower level functions)*
>
> *- to restate the previous point, use cases are a SCENARIO decomposition, not functional decomposition*
>
> *- scenarios are generally distinguished by having an actor, and represent event/response dialog between a system and something outside of that system, whereas functions are algorithmic (all one system)*
>
> *- to restate that one, algorithms are not scenarios (algorithms are generally represented as controllers within scenarios)*
>
> *- functional requirements are not scenarios (but they should be linked to the scenarios). Requirements can also be linked to controllers.*
>
> *Doug*

Tim's response to this email was as follows:

> *Hi Doug,*
>
> *I think there's a long-term and a short-term response to your concerns. First, the long-term. I've gradually been coming to the view that a variant of the Iconix process needs to be developed that is*

*better tuned to people like us (astronomers, physicists, scientific application programmers) and the systems we need to construct. I think that the difficulties that we've run into with LBT and LSST are not only due to our own lack of discipline and cat-like natures. Even those of us who support the use of a formal design process (like me) have not really used it as intended, as you are seeing. At least sometimes, the process just doesn't seem a natural fit to the way we think about our systems. The part of LSST we're working on now is dominated by complex pipeline processing, and when we scribble on our whiteboards we always end up with inputs and outputs and boxes that transform data. And so, when we go into UML we want to continue thinking that way, and end up torquing the design process into something that doesn't work right. So, perhaps after PDR is complete (and we've survived it), it would be worthwhile to devote some serious thought to creating a modified Iconix process ("The Iconix Process for Scientific Cats" by Rosenberg, Kantor, and Axelrod??)*

*In the short term, I'm not sure how to proceed. Jeff has some pretty definite ideas of what we need to get through PDR, and somehow we need to mobilize the team to produce what''s needed. And there is the minor matter that we have a lot of code to produce as well! For the modeling, it might actually make most sense to take a subteam (say Jeff, Nicole, and me) and notionally lock us in a room to produce the model, drawing as needed on the other application folks and translating what they tell us into UML. If that's the right approach, we should try it on for size next week. Maybe there's some small modification/addition to the Iconix process that could help in the short term - you mentioned DFDs. I'll in good part leave this set of decisions up to you and Jeff, though I'll certainly want to kibitz (and maybe protest!).*

*Keep thinking,*

*Tim*

So here's where it started. A suggestion from Tim that maybe it was time to think about tailoring the process. Since Tim's one of the smartest guys that I know, and I have never ever known him to speak without a good deal of forethought, I took this email very seriously, and I started mulling over what an algorithmic variant of ICONIX Process might look like.

We decided to take a day for the three of us (Tim, Jeff, and myself) to work through an example which we'd use as a model for workshop attendees. We proceeded to produce a "stretched ICONIX Process" example, which captured high-level algorithms as use cases, and lower level algorithms as controllers on robustness diagrams. I don't think any of us felt completely comfortable with the example, but Jeff felt pretty strongly that trying to introduce something other than ICONIX Process as published in my books at this particular workshop would cause more problems than it might solve, when what they really needed was some forward progress capturing algorithm detail within the model. So this wasn't the time to be thinking about process tailoring.

But all three of us knew that we were stretching "textbook ICONIX Process" further than we felt comfortable with.

As we worked through our example, the thought that kept growing in my head was… trying to model algorithms as use cases was the fundamental problem, because scenario modeling (which had worked on LBT) and algorithm modeling were different, and needed to have different guidelines. In the LSST image processing software, scenarios were scarce, and algorithms were plentiful. To further complicate matters, the algorithms were organized into workflows and data productions[24].

As it turns out, the Pasadena workshop reinforced these ideas quite strongly in my mind. But it was during our day of pre-workshop preparation that I first suggested we stop using use cases to model algorithms, distinguish between workflow and science algorithms (and use activity diagrams to describe workflow, and "controllers" on robustness diagrams to capture details about science algorithms).

[24] In LSST terminology, "workflows" are the image processing pipelines, and "productions" involve running a set of pipelines to produce a set of data.

## *"We're Not Building Internet Bookstores"*

I had been warned on several occasions about the level of anti-modeling sentiment in some parts of the LSST team, and in particular that one highly-brilliant "superhero programmer" was the most vocal, and that I'd be meeting him for the first time at this workshop. I won't use his name in this discussion but since he is a legitimate "superhero programmer" (being largely responsible for another major survey telescope's software), I'll refer to him as superCoder which is a fairly accurate description of his skill level. There was also another individual who, having been in the industry for quite a long time, dated back to the days of FORTRAN, and who had dubbed the robustness diagram notation "Martian" and proudly pronounced that "I don't speak Martian." For this discussion we'll call him FORTRAN_JOCK.

So Jeff and Tim had warned me to expect active resistance from superCoder and FORTRAN_JOCK during the workshop, and I was not to be disappointed. Jeff (and to some extent Tim) has spent many hours banging heads with these folks regarding the value and necessity of modeling. superCoder, having more or less single-handedly programmed the other survey telescope didn't see why he couldn't do the same with LSST. LSST, of course is massively bigger and more complex than any previous survey telescope and this would be a complete impossibility, even if it were desirable.

The first morning of the workshop turned out to be quite contentious. I had a couple of hours on the meeting agenda to present the example we had developed and give a quick overview of the plan-of-attack for the next 2 days. Jeff had predicted that this time would be used by superCoder and FORTRAN_JOCK to basically attack the whole approach we were using, and open a debate about modeling that, if left unchecked, would suck up the entire 2 days that were allocated to capture algorithms into the model. Jeff's predictions were entirely correct, led off by superCoder's proclamation to me (and the assembled group) that he had read my book (whose example happened to be an Internet bookstore), and that while it contained lovely guidance for those who might be building Internet bookstores, that the LSST DM group was most emphatically *not* building internet bookstores, and thus the guidance presented in my book was generally not useful to him. This was followed by FORTRAN_JOCK's pronouncement that he didn't speak Martian, which got chuckles from everybody, including me. I now routinely introduce the robustness notation to students as "Martian".

Jeff, having fully anticipated this sort of speech from superCoder, was absolutely in no mood to tolerate it. Jeff is responsible for the LSST project passing PDR, and continued funding for the project is contingent upon the cost and schedule estimates that he needed to produce, and he needed information captured into the model during this workshop for that purpose. In a performance worthy of General Patton[25], he stood up and basically out-shouted the naysayers and declared that we would NOT spend this meeting debating the approach, and that we WOULD spend the meeting capturing algorithm detail, end of debate, LET'S GET TO WORK.

I was privately somewhat sympathetic to the "Internet bookstore" argument because I knew we were stretching the process farther than I was comfortable with, and by then I had some ideas on how to tailor it to be a better fit to the LSST Data Management problem, but I certainly understood where Jeff was coming from, and I was quite impressed with his performance, which he had forewarned me might be coming. In any event, it worked exactly as he intended, and the workshop proceeded to capture algorithm detail for the next 2 days.

---

[25] Jeff's performance actually reminded me of one of my favorite scenes from the movie *Patton*, which is one of my favorite movies. In the scene, Patton's army is marching 100 miles through the snow trying to rescue a battalion at Bastogne during the Battle of the Bulge. His officers are lamenting the bad weather and the fact that they can't get any air cover. Patton launches into a tirade that ends with: "If we are not victorious, LET NO MAN COME BACK ALIVE." His aide pulls him aside and says: "General, sometimes the men can't tell when you're acting." To which General Patton responds: "It isn't important for them to know… it's only important for me to know."

## Modeling Tip—Don't confuse use cases and algorithms

Many people confuse use cases with algorithms because both are generally named with a verb-phrase and consist of a sequence of steps. In most systems, algorithms are represented as steps within use cases (controllers on robustness diagrams). LSST's image processing pipelines don't really have any use cases.

I would rate the result of this workshop as a qualified success. We captured a lot of algorithms, but we captured them in a less-than-optimal form. Essentially, the conclusion of our pre-workshop meeting was that we knew it was going to be messy, but that we'd clean the mess up later. If it had been anyone but Jeff promising that we'd clean it up later, I would have been pretty skeptical about it ever happening, but I knew he meant it. And as you'll see shortly, Tim and I were destined to become the cleanup crew, during which time the actual process tailoring took place, on a whiteboard in Tim's office.

We organized the workshop into teams, each specialized in the functional area of their particular laboratory. There were people from Princeton, University of Washington, Caltech, Stanford, and several other locations. I took charge of one lab team, Jeff took another, and Tim another, and we all floated around as best we could during the lab sessions.

The approach was to model high-level algorithms with use cases, and lower-level algorithms as controllers on the robustness diagrams for those use cases. Since the software didn't have any user interface, we didn't expect to see any boundary objects on the robustness diagrams, but we did want to use the robustness diagrams to discover missing domain objects and to identify lower-level algorithms within the higher-level algorithms. The problem with this particular project is that LSST image processing has many levels of algorithms-within-algorithms-within-algorithms-within-algorithms and the guidance wasn't real clear when to model with a use case and when to use a robustness diagram.

To save time, we decided to have each lab team produce its own "mini domain model" before trying to describe any use cases. I was much more rigorous about this with the teams that I worked with than Jeff and Tim were, and I think it made a significant difference in the amount of progress made by the various teams. Jeff had initially penciled himself in to work with superCoder's lab team, but during the lunch break I approached him and suggested that I work with that team instead because the friction level between them was obviously way too high. I think that was the quickest agreement I have ever received from Jeff on anything.

As it turns out, having me work with superCoder's team was a pretty good idea. I've been developing software with Very Smart People since my college days (often PhD physicists whose science and math abilities go way over my head), and it has exposed me to some very interesting projects. I have an approach that I often use when I'm working with Tim, and it pretty much involves me starting a discussion with: "So, tell me about X" (if I remember correctly, X in this case was the Deep Detection Pipeline), and letting the Very Smart Person start explaining how it all works, while I "take notes" in UML.

There's slightly more structure behind my questions than just "tell me about the Deep Detection Pipeline" in that I'm fishing for specific information with my questions. The first thing I fish for is names of things (nouns). So while the VSP (in this case our good friend superCoder) is telling me "what are the things that are involved in deep detection", a domain model diagram is assembling itself in Enterprise Architect (see Figure 1).

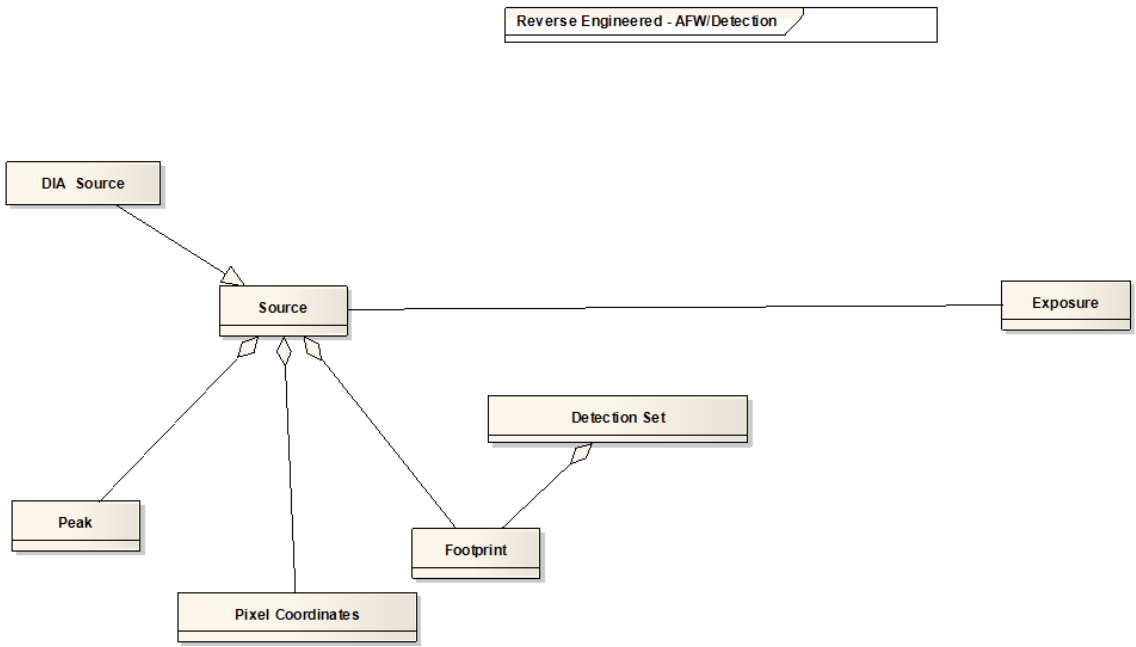



***Figure 1. Some of the domain classes that are involved in deep detection. Note the link to a separate diagram showing reverse engineered code from the prototype Deep Detection Pipeline.***

Occasionally I'll stop and ask whether the domain model I'm assembling looks right or not. The key to this approach is that *I need to be able to capture information without slowing down the narrative explanation* of the VSP. In this case, superCoder had already built a deep detection pipeline before, knew exactly how he wanted it to work, and was perfectly capable of explaining it at a mind-bending pace.

## Modeling Tip—Always Start with a Domain Model

Failure to define your terms will always cause problems when modeling. So you should always start with a domain model. But don't get stuck trying to make it perfect upfront.

I don't like to spend too much time on domain modeling, but rather try to nail down the 80% most important domain objects as quickly as possible with an understanding that we'll continually refine this as we discover new domain classes. So after 15 or 20 minutes of domain modeling, I switched my questions over to algorithms. As it turns out, the "magic questions" for modeling use cases[26] work reasonably well for algorithms too—and I make sure any detail that I capture uses the nouns from the domain model. In this case I was trying to separate pipeline workflow from science algorithms, so I started with "tell me about the high level algorithm for the Deep Detection Pipeline" and proceeded from there. Once again, the key is that I have to be able to capture information as fast as a Very Smart Person can explain it to me, so the CASE tool that I'm using (Enterprise Architect) had better not get in my way while I'm working (which it didn't).

In a sense, I use UML and visual modeling the way a good secretary uses shorthand to capture the text of a memo or business letter. So the notation and tool that I'm using are analogous to shorthand and a steno pad. The good news out of all of this was that within 5 minutes of starting

[26] As described in Use Case Driven Object Modeling, the magic questions are: 1. What happens? 2. And then what happens? (repeat as necessary) 3. What else can happen? (repeat as necessary). These questions work well for capturing algorithms, too.

the lab portion of the workshop, our friend superCoder was standing up at the whiteboard explaining algorithms (which he was very good at and liked to do), and the algorithms he was explaining were getting captured into Jeff's model, by me. For the moment, we had overcome the resistance to modeling, but I knew that the need for process tailoring was an issue that wasn't going away anytime soon.

## *Tim and I Become the Cleanup Crew, and ICONIX Process Gets Tailored*

After the August Pasadena DC3 workshop, I wasn't involved with LSST for several months. But as the PDR deadline started creeping closer, getting the model cleaned up started moving higher up the priority list for Jeff and Tim. So it was in mid-December that I found myself in Tucson again for a 2-day working session with Tim to try to get the model in better shape. I think all of us would have been quite happy for me to be involved with the modeling on a more frequent basis, but this was what the budget permitted. My approach was to try to be as helpful as I could within the limited time available.

I hadn't seen anything of the LSST model since August, and it didn't look like Tim had really gotten any real traction on doing much with it. Jeff had gotten the results of the Pasadena workshop inhaled into the existing LSST model, but it was sort of just sitting there off to the side, waiting for the cleanup that we all knew was necessary. My arrival was the catalyst for this cleanup to start, and Tim and I were the cleanup crew.

The first issue we ran into was the domain model, which was actually not a single domain model anymore but instead a collection of multiple, fragmented, and inconsistent domain models. Because of the split into multiple teams (each with a "mini domain model") in Pasadena, there was no single place to look for "the domain model" which definitively established a consistent set of names for domain classes, and the relationships between them. The domain model included reverse-engineered code from prototyping efforts, and (as on many projects that haven't discovered the power of domain modeling) names were used inconsistently throughout the model.

Tim and I ran head-first into the need to clean this up within 10 minutes of getting started. I had asked him some question which caused him to try to find something in the domain model, and it was much too difficult. It became clear to me that despite Tim having every intention of trying to make the model work, he was continually having to fight with it.

### Modeling Tip: If you're fighting the model, figure out why

UML models exist to help define and communicate a development problem and its solution. If you find yourself having to fight the model, something's wrong. Take the time to figure out what the problem is, and fix it. Models should be a help, not a hindrance.

So we decided to attack this problem first, and spent a few hours printing out each and every domain model diagram, making notes on them with markers of various colors, and one by one adding dozens of objects that had been discovered in Pasadena into the projectwide domain model (renaming many of them as we went as name usage had been inconsistent previously).

We also removed reverse-engineered code from the domain model into a separate package, but grabbed the names of some of these classes and added those to the domain model. Within a few hours we saw a noticeable improvement.

## Modeling Tip: Keep reverse engineered code separate from the domain model

It's OK to prototype code and reverse engineer it into a UML model. On systems like LSST, it's essential. But don't mix the reverse engineered classes with the domain model.

As the consolidated domain model grew, I suggested to Tim that it was really pretty important for him to add definitions to all the domain classes, and he agreed (with some reluctance, because it was a lot of work). He was heading to a conference in some remote place the next week and promised to spend some time adding the definitions, because he saw the value in having a project glossary. I received an email from him while he was in the middle of this exercise that said:

> *Would you believe it? The exercise of writing the definitions is turning up some more domain objects... Tim*

He later told me that "some more" was really about a 20% increase in the number of domain objects discovered, just by adding definitions to the classes. So, if you look for high-leverage modeling activities, like I do, this is definitely one to make a note about. Define your project vocabulary, it works. Thanks to Tim's continued efforts (with some help from Robyn), the LSST Domain Model is now an extremely useful section of the model.

## Modeling Tip: Use your domain model as a project glossary

Trust me, you'll be glad you defined your domain classes unambiguously. It generally only takes a few hours, and pays dividends over the lifetime of your project.

Once the domain model got consolidated, the next thing we needed to do was to bring all of the "use cases" and robustness diagrams from the Pasadena workshop in-line with the (now standardized) domain terminology. This was no small task, certainly not one that could be completed by Tim and I in less than two days, and it was during our initial attempts at this that I became convinced that the process tailoring couldn't wait much longer.

I hadn't seen the model since August, and during the workshop in Pasadena I was mostly focused on the models produced by the teams that I worked with. As Tim and I started digging, here's what we found:

**Pipeline workflows were modeled as use cases.** Use case diagrams were used to describe the various pipeline stages and how they related together.

**Algorithms (pipeline stages) were also modeled as use cases.** So when you looked at a diagram it wasn't easy to tell if you were looking at a pipeline, a pipeline stage, or a science algorithm. Everything looked the same.

**"Schizophrenic" use case descriptions which had "inputs/outputs" like an algorithm and a "basic/alternate" structure like a use case.** Not surprisingly, with use cases being used to represent algorithms, the model was full of "algorithm use cases" that had a split personality (half algorithm, half use case). The problem here is that the best-practice guidelines for writing good use cases are not the same as the guidelines for describing algorithms. Thus representing algorithms with use cases tends to be confusing to both creators and readers of the model.

While none of this was particularly surprising to us, it struck me right between the eyes that it was time to do something about it, and that the compelling reason to do it now was that resistance to modeling (which didn't need any further ammunition) was being fueled by the fact that the model simply wasn't working (i.e. was not easily understandable) as well as it should.

## Modeling Tip: Watch for signs that your software process may need tailoring

Here are some telltale warning signs to look out for:

### *The arguments against modeling seem compelling*

There is inevitably some resistance to any sort of UML modeling. Generally there is a "hardcore" group of disbelievers who think that any sort of pre-coding modeling activity is a waste of time, and some others who understand the need for a more disciplined development approach and will make some attempt at making modeling work. When the "disbelievers" present arguments that seem compelling to a large percentage of the team, and when those who are trying to make modeling work are finding that it's tough sledding, it's a clue that the process needs tailoring.

### *The model feels like a hindrance rather than a help*

The purpose of UML models is not to slow a project down for no good purpose (we realize this may come as a surprise to some). The purpose of the model is to capture all the information that needs to be captured in order for the entire team to develop a common understanding of the problem as completely as possible before coding begins. This enables not just the developers to proceed with their work in a more consistent way, but allows for parallel work by test designers and project management, essential for timely project delivery and managing. For the great majority of systems which involve users and user interfaces, a lot of this information takes the form of use cases. For systems that run with little or no user interaction, use case techniques are not necessarily the most appropriate.

### *The model is confusing to those who try to read and understand it*

A good model should explain the requirements and intended behavior of the system clearly and concisely. If the model is not easy to follow and understand, it may be a sign that tailoring is required.

As Tim and I sat in his office discussing the reasons that *we should tailor the process for LSST right now*, it was somewhat problematic because Jeff had made it very clear that he didn't think there was enough time to take on any sort of process modifications before PDR. So it was a bit of a dilemma. Finally I suggested to Tim that he and I consider working a small example and showing it to Jeff in the hopes he would agree that it was obviously an improvement. Tim, who you may recall had suggested process tailoring in the first place, agreed. We initially undertook to convert one pipeline (Instrument Signature Removal) into the new format to "try it on for size".

By this time, having had several months to reflect on the problem, I had pretty well formed an idea of how I thought the process should be tailored. I had a precedent for tailoring ICONIX Process for Business Process Modeling where we had used activity diagrams to describe workflows in the ICONIX Business Modeling Roadmap.[27]

This had proven to work pretty well, and the more I looked at the use case diagrams for the LSST pipelines the more convinced I became that *there was a fundamental difference between the pipelines*, which defined the workflow for a stream of images, *and the science algorithms*, which perform mathematical operations on the images. I thought the model would be much less confusing if we modeled the workflows on activity diagrams.

[27] www.iconixsw.com/BPRoadmap.html

## Modeling Tip: Guidelines for tailoring a development process

The overriding rule in process tailoring is ***"fit the process to the problem"*** (and not vice-versa). ICONIX Process uses a core subset of UML diagrams, not because the other parts of UML are never useful, but because this subset has proven to work consistently and effectively while avoiding analysis paralysis on a wide range of projects.

When a project has need for additional UML diagram types, ICONIX Process has always recommended "tailoring up" to include them. If we overzealously stick to the 4 diagram types (class, sequence, use case, and robustness) in the core ICONIX subset, we run the risk of "everything looks like a nail because all we've got is a hammer". In this case the nail is a use case, and the hammer is a use case diagram.

Note that starting from a minimalist modeling subset and tailoring up is a very different approach from that taken by processes such as Rational Unified Process (RUP), which start with everything and require chipping out everything that's not needed. Generally, tailoring down involves a lot more work than tailoring up.

The overriding issue with applying ICONIX Process to the LSST DM software is that ICONIX Process is fundamentally a use-case driven approach to object-oriented design, and the LSST DM software has very few real use cases but is instead heavily algorithm-intensive.

The tailoring I proposed to Tim was actually a pretty simple extension which retained all the benefits[28] of use-case-driven ICONIX Process while being a lot less confusing to everyone involved. These changes were:

1) **Don't describe anything other than a real use case with use cases.** Specifically, don't describe workflow as use cases, and don't describe algorithms as use cases. LSST will have "real use cases"; many of these will involve using the data produced by the image processing pipelines. But the part of the software we modeled is almost completely devoid of them.
2) **Describe workflow using activity diagrams.** Essentially, for LSST, this meant that Pipelines and Productions would be described using activity diagrams.
3) **Describe high-level, policy-directed science algorithms on robustness diagrams.** Recognize that "policy" is essentially a proxy for a human sitting in front of a GUI driving the image processing, and treat it as an actor. This allows the robustness diagram rules to be used correctly.
4) **Describe more "atomic" number-crunching science algorithms as controllers on robustness diagrams.** So each controller will specify Inputs, Outputs, Algorithm, and Exceptions.

This tailoring of ICONIX Process is widely applicable beyond LSST to an entire class of algorithm-intensive systems. We present it here as *ICONIX Process for Algorithm-Intensive Systems* (or, *ICONIX Process for Algorithms*).

[28] Another useful principle to follow is: *Fix what's broken, but* ***don't try to fix what isn't broken***. What wasn't broken about ICONIX Process was using robustness diagrams to identify missing domain objects and identify "controllers" (science algorithms), and using sequence diagrams to do a responsibility-driven allocation of behavior to objects. Jeff, Tim, and I were all unanimously agreed on not sacrificing these benefits. We just needed to find a way to make them work when there were no real use cases.

## *A Few More Thoughts About ICONIX Process for Algorithms as Used on LSST*

***Modeling pipelines as activity diagrams*** involved not only "transmogrifying" the diagram from a use case diagram to an activity diagram, but also incorporating "Policy" as an actor which defined paths through the various pipeline stages. Although the LSST DM software will run without human intervention, various predefined Policies act as proxies for how a human user would guide the software. As it turned out on LSST, there were two parallel sets of image processing pipelines that differed only in the policies to guide them, so making the pipeline activity diagram "policy driven" immediately allowed us to cut the number of "pipeline use case diagrams" in half. This was an encouraging sign as an immediate simplification of the model resulted from the process tailoring we did.

***Modeling pipeline stages as high-level algorithms*** meant replacing the "schizophrenic" algorithm-use case template of

*Inputs:*

*Outputs:*

*Basic Course:*

*Alternate Courses:*

With an activity specification template more suited to algorithms, namely:

*Inputs:*

*Outputs:*

*Algorithm:*

*Exceptions:*

Not surprisingly, writing algorithm descriptions as algorithms and not as scenarios made the model much easier to understand. This simple process modification went a long way towards addressing the lack of semantic consistency in the model.

***We used robustness diagrams to elaborate activities (is that legal?)***[29] The "algorithm-use cases" that had been written in Pasadena had been elaborated on robustness diagrams, and we made the non-standard process enhancement to elaborate the pipeline stage activities with these robustness diagrams as well. Enterprise Architect was flexible enough to support this.

### Modeling Tip: Good modeling tools are flexible

I've been bending the rules (and writing new ones) of software development processes for more than 20 years. One of the key attributes that I look for in a tool is ***flexibility***. Over the years, I've found that I can make Enterprise Architect do almost anything. It helps me, but doesn't get in my way.

Keeping this elaboration of pipeline stage algorithms on robustness diagrams was important for a number of reasons, one of the primary reasons being that we wanted to maintain the decomposition into "controllers" (lower level algorithms) and "entities" (domain classes). Another important reason was that project estimation tools and techniques relied on the number of controllers within a given pipeline stage (and an estimate of level of effort for each controller) for cost and schedule estimation.

[29] Luckily I have a license from the UML police to make these sorts of non-standard process enhancements.

Note that since the domain model had not been consolidated when the original robustness diagrams were created, these diagrams had to be re-visited anyway (since the domain classes appear as "entities" on the robustness diagrams). Not surprisingly, numerous other errors were uncovered during this cleanup pass through the pipeline stage algorithms. All of this work was absolutely necessary in order to get the project ready for PDR.

## *Some Final Thoughts on Process Tailoring*

While it's generally a good thing to give a process a fair chance to work, and we've found that ICONIX Process "out of the box" works very well for a very wide range of projects, it's also advisable to be alert for any fundamental issues that might require tailoring. In the case of LSST DM, it proved too cumbersome (although not impossible) to bend the rules and model algorithms as use cases.

While we knew from the beginning that we were stretching and twisting the rules during the Pasadena workshop, the team still managed to produce a large amount of useful work. In hindsight it might have been advisable to tailor the process a bit earlier, but, as with many things, the issues became clearer as more progress was made.

***The most important conclusion is to not be afraid to tweak the process if it's not working as well as it should***. Neither ICONIX Process nor any other software process is chiseled on stone tablets, and they should never be treated as if they were.

If it ain't broke don't fix it, but if it is… don't hesitate to do what's necessary to make your process work for you.

## *All's Well That Ends Well…*

As things turned out, the LSST PDR date was slipped by the funding agencies, allowing time for Tim, with assistance from Robyn and a couple of working sessions with me, to update the entire model in this fashion with excellent results—some of this updating is still in progress as I'm writing this. We'll use the algorithmically tailored version of the model to show how to detect asteroids that might hit the Earth (and other moving objects) in the next chapter.

## Chapter 5

# How Do You Detect an Asteroid That Might Hit the Earth?

This chapter pulls together a few pieces of the LSST model to tell the story of how the LSST software will identify moving objects from its torrent of images.

## *Domain Model*

Let's start with a small fragment from the LSST Domain Model, focusing on "Astronomical Objects" (AstroObjects). The domain model, in addition to it's large main "almost everything" diagram, also is organized into subset diagrams showing clusters of related objects. It's often a lot easier to look at the subset diagrams, and that's the case here.

You can see from Figure 1 that we've identified a Solar System Object as a distinct kind of AstroObject, and that Solar System Objects have Orbits. We've also identified something mysterious called Ephemerides. As it turns out, Ephemerides are important to our story. But... (as I asked Tim on a recent visit to Tucson) what's an Ephemeride?

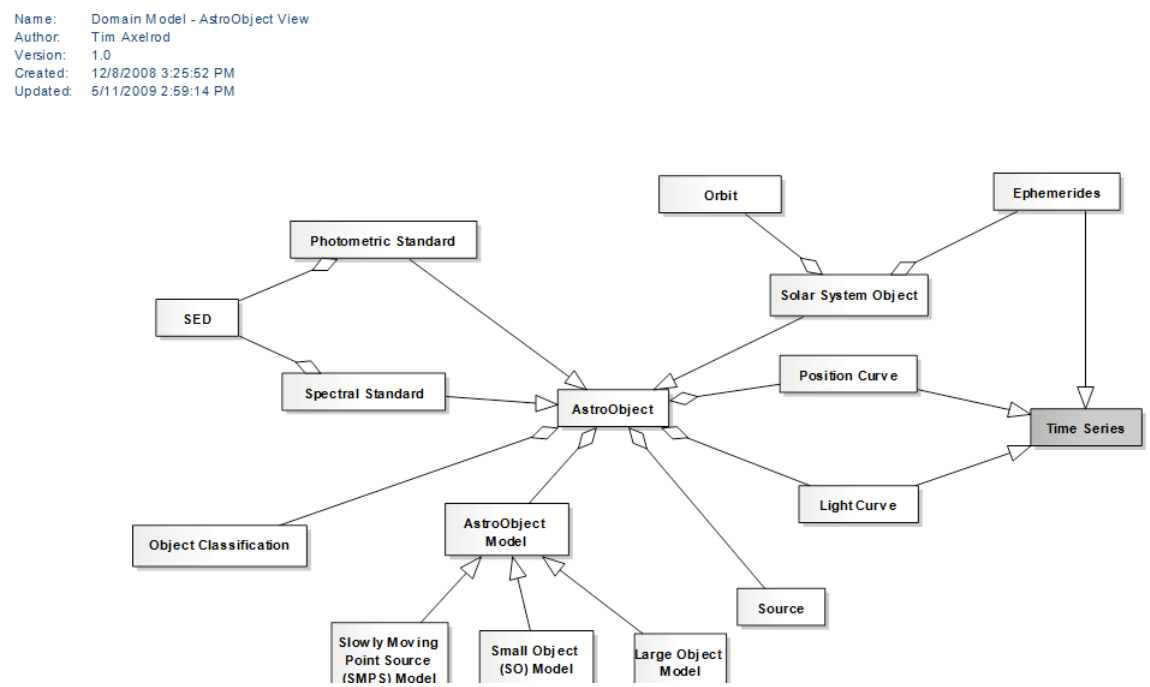



***Figure 1. Domain model showing different kinds of Astronomical Objects.***

Here's where Tim having defined his domain objects comes in handy. We can just look it up under E for Ephemeride. Using the domain model as a glossary is useful on any project, but when you have a distributed team like LSST, it's absolutely essential (see Figure 2).

### 3.1.34 Discontinuity Mask

A **Mask** associated with a **CCD**. It has four mask planes: X-, X+, Y-, Y+. If a pixel has the X- plane set, the corresponding **CCD** pixel has a geometrical discontinuity with the pixel in the negative X direction. Similarly for X+ (positive X), Y- (negative Y), and Y+ (positive Y) planes.

### 3.1.35 ECG List

### 3.1.36 Electrical Geometry

The **Electrical Geometry** specifies the order in which the **Segment** pixels are read out, including not only the physical imaging pixels, but also the overscan pixels that can be located before and/or after each row of physical pixels.

### 3.1.37 Ephemerides

A **Time Series** of predicted **Sky Coordinates** for a **Solar System Object.**

### 3.1.38 Ephemeris Collection

A table of values which gives the positions of astronomical objects at a given time or times.

***Figure 2. Using the Domain Model as a Project Glossary is a very useful technique.***

As you can see, an Ephemeride is a **Time Series** of predicted **Sky Coordinates** for a **Solar System Object**. In other words, the predicted track of an object in the solar system (like a Martian Spacecraft, or an asteroid that might be heading for Earth, or the planet Neptune, or one of Saturn's moons, or even the Hubble Space Telescope). In the words of an old electromagnetic theory professor of mine, this is "intuitively obvious to the casual observer."

## Modeling Tip: It all starts with Domain Modeling

In complex systems like LSST, the Domain Model can become quite large. When this happens, create "subset view" diagrams that show a relate set of objects. Don't forget to define all your domain classes unambiguously.

So far, so good. The LSST Domain Model has an object that exists specifically for the purpose we have in mind. Now let's take a look at how we're going to compute these Ephemerides.

## *Image Processing Pipelines*

LSST's image processing software uses a "pipeline" architecture. Images go in one end of the pipeline through an Input Queue, and are analyzed as they pass through various "processing stages", then exit through an Output Queue. LSST's middleware defines a general purpose architecture for pipelines which allows for parallel processing of the image stream. Parallel processing is an absolute necessity when you're dealing with a stream of 3 gigapixel images with a new image coming through every few minutes. We're going to be looking at one of many LSST pipelines later in this chapter, a pipeline called "Day MOPS".

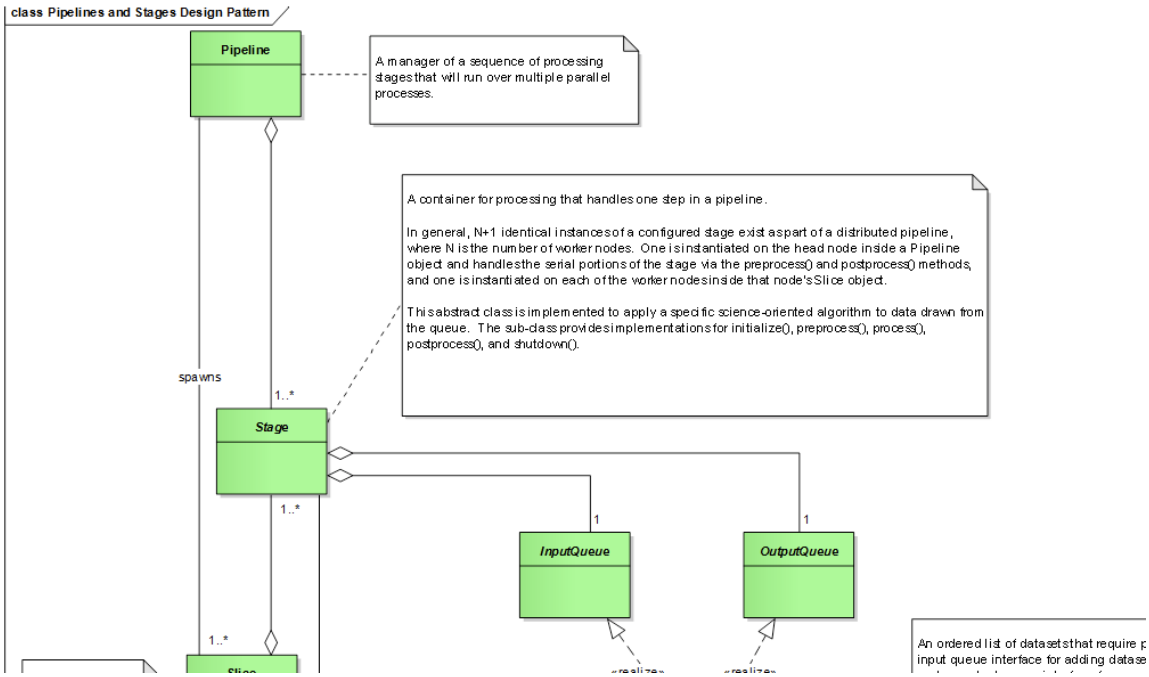



***Figure 3. LSST's middleware manages the image processing pipelines.***

## *Policies*

LSST's software will operate at much too high a rate for there to be human guidance and direction during the execution of a pipeline.However, there are many occasions where human guidance is necessary. LSST pipelines can be controlled by Policies, which are sets of parameters that human experts (astrophysicists) can define. So a Policy is really like a proxy object that replaces a person who would be guiding the image processing software if you slowed down the processing by a couple of million times. (See Figure 4).

#### 3.2.2.6.1 Policy

Policy is a container for holding hierarchical configuration data in memory. A policy is a set of named parameters that can be used to configure the internal data and behavior of an object within an application.

An important feature Policy objects is that the parameters can be loaded in from a file. Thus, it allows applications fine-grained control of objects even if much of the configuration parameters they provide are normally set to defaults and otherwise do not change. The Policy interface allows an application to pull out parameter values by name. Typically, the application "knows" the names it needs from a Policy to configure itself--that is, these names and the use of their values are hard-coded into the application. The application simply calls one of the get methods to retrieve a typed value for the parameter. (Nevertheless, if necessary, the parameter names contained in a policy can be retrieved via the \c names() member function.)

***Figure 4. LSST's pipelines are "policy driven". Policy objects are sets of parameters that effectively act as proxies for expert astrophysicists. These parameters guide the image analysis, since the image processing runs far too quickly to allow for actual human intervention.***

## *The Day MOPS Pipeline*

Since LSST Pipelines are used to define workflow (and are not really use cases) we're using an activity diagram to describe the workflow, instead of trying to describe the pipeline detail on a use case diagram. So it's clear in the model that when we see an activity diagram, we're working at the upper levels of the model.

Figure 5 shows the activity diagram that describes our moving object detection pipeline.

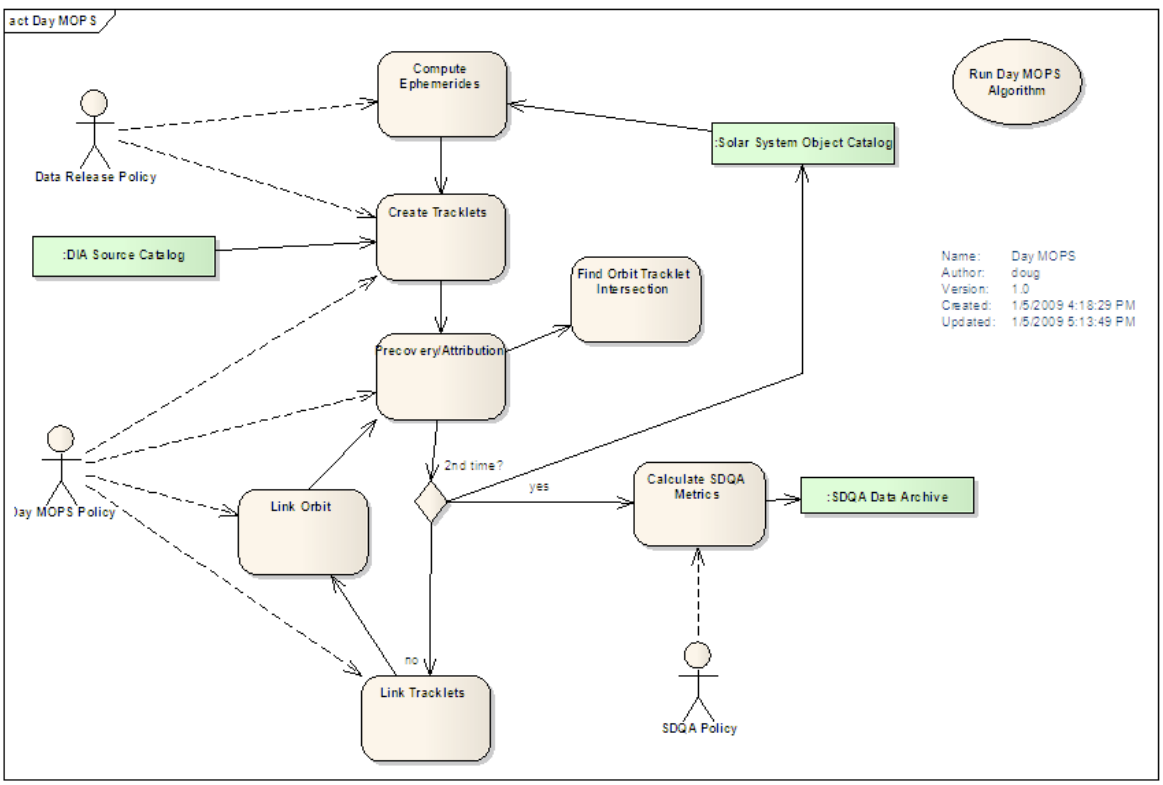



***Figure 5. The Day MOPS pipeline detects moving objects and updates several of LSST's Catalogs, including the Solar System Object Catalog.***

## Near Earth Object Detection

LSST's Day MOPS pipeline is being developed in close collaboration with a project called PAN-STARRS[30] (Panoramic Survey Telescope & Rapid Response System). The PAN-STARRS website has an interesting discussion about Near Earth Object (NEO) detection and the potential threat from asteroids.[31]

Also, NASA JPL maintains a Near Earth Object website if you'd like to take a peek at which asteroids will be making close passes to us in the near future[32]. The PAN-STARRS website lists their digital cameras as the world's largest, at 1.4 gigapixels[33]; this won't be true anymore after LSST's 3.2 gigapixel (that's 3200 megapixels)[34] camera gets built.

## *A Look Inside the Pipeline Stages*

Within each activity, we're elaborating the processing on a robustness diagram as if the activity were a use case, and we're representing Policy as an actor, and (to make our robustness diagram rules work correctly) defining Policy Readers as boundary objects. Since the robustness diagram is a conceptual design diagram, it doesn't really matter whether a policy reader object will ever be implemented. We're using it in the model *as a device to help us identify what the different policy parameters should be*.

## Modeling Tip: Robustness diagrams help "Object Discovery"

Robustness diagrams (aka "Martian") are conceptual design diagrams that are useful to help you discover details about your object model. Since it's not an implementation model, it's OK to add conceptual objects like Policy Readers to help us discover what data we're reading from the Policy. **See Figure 6 for an example of a robustness diagram.**

By elaborating pipeline stages within activities, and allowing for "controllers" (lower level algorithms) within the robustness diagram, we're putting a limit on how many levels of "algorithms within algorithms within algorithms within algorithms" we're going to show in the model. The controllers that are connected to the Policy Readers help us to identify exactly what needs to go into the Policy.

## Modeling Tip: Avoid deep trees of nested functions

One of the problems with functional decomposition approaches that are typically used in algorithm-intensive systems is that there tend to be many levels of nested algorithms. *ICONIX Process for Algorithms* represents high-level (workflow) algorithms on activity diagrams, elaborates policy/parameter-driven activities on robustness and sequence diagrams, and lower-level "number cruncher" algorithms as controllers within the robustness diagrams.

So it's much easier to figure out what level of algorithm you're looking at when you read the model.

---

[30] http://pan-starrs.ifa.hawaii.edu/public/

[31] http://pan-starrs.ifa.hawaii.edu/public/asteroid-threat/asteroid_threat.html

[32] http://neo.jpl.nasa.gov/

[33] http://pan-starrs.ifa.hawaii.edu/public/design-features/cameras.html

[34] http://www.lsst.org/lsst/gallery/camera/suzanne

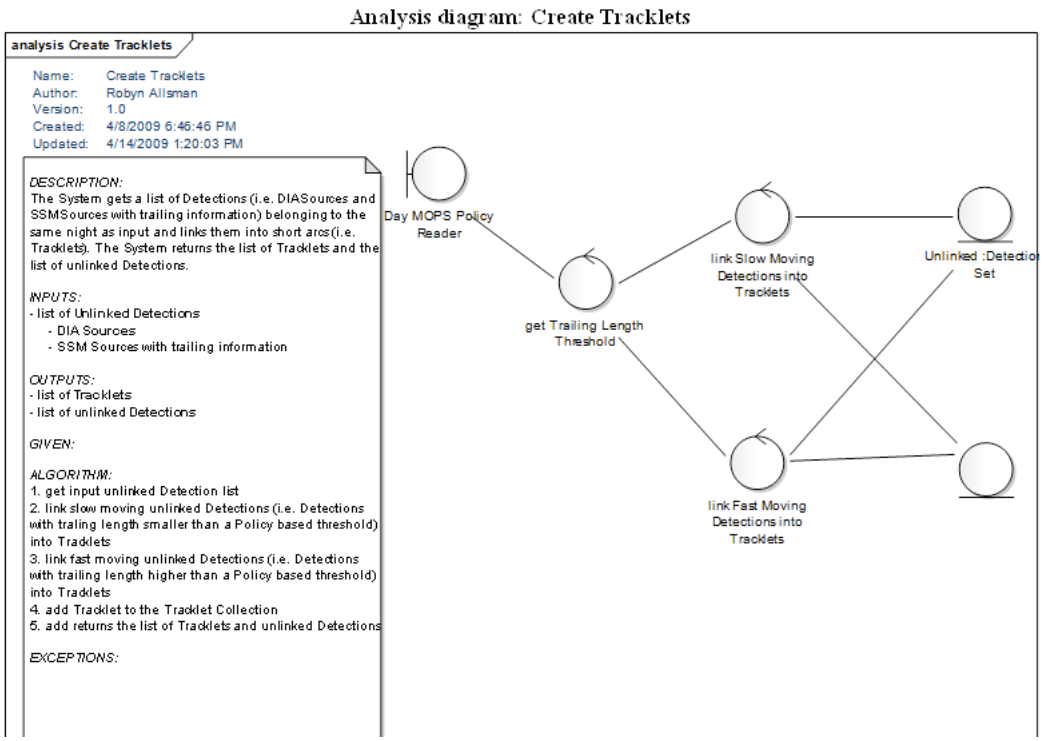



***Figure 6. A robustness diagram for Tracklet Creation. "Martian" notation is actually pretty simple.***

Notice that even though we're using it to describe a policy-directed algorithm instead of a use case, the robustness diagram still works. This allows us to leverage other capabilities of ICONIX Process and the Sparx Agile/ICONIX add-in to generate skeleton sequence diagrams and test cases automatically.

## Modeling Tip: ICONIX/Algorithms retains all the benefits of ICONIX Process

The standard, use-case-driven, ICONIX Process discovers details about an object model using robustness analysis, and does a responsibility-driven allocation of behavior using sequence diagrams. Additionally, the Agile/ICONIX add-in from Sparx Systems supports automatic generation of test cases and JUnit/NUnit test code from robustness and sequence diagrams. ICONIX Process for Algorithms retains all of these benefits.

ICONIX Process for Algorithms is useful for a wide range of algorithm-intensive systems.

Figure 7 shows an algorithm for a pipeline stage that hasn't been elaborated (yet) on a robustness diagram. Exactly as with use-case-driven ICONIX Process, drawing the robustness diagram helps us discover additional domain objects that may still be missing from the model. So our process tailoring has retained the benefits of use-case-driven ICONIX Process, without trying to force-fit algorithms into use cases.

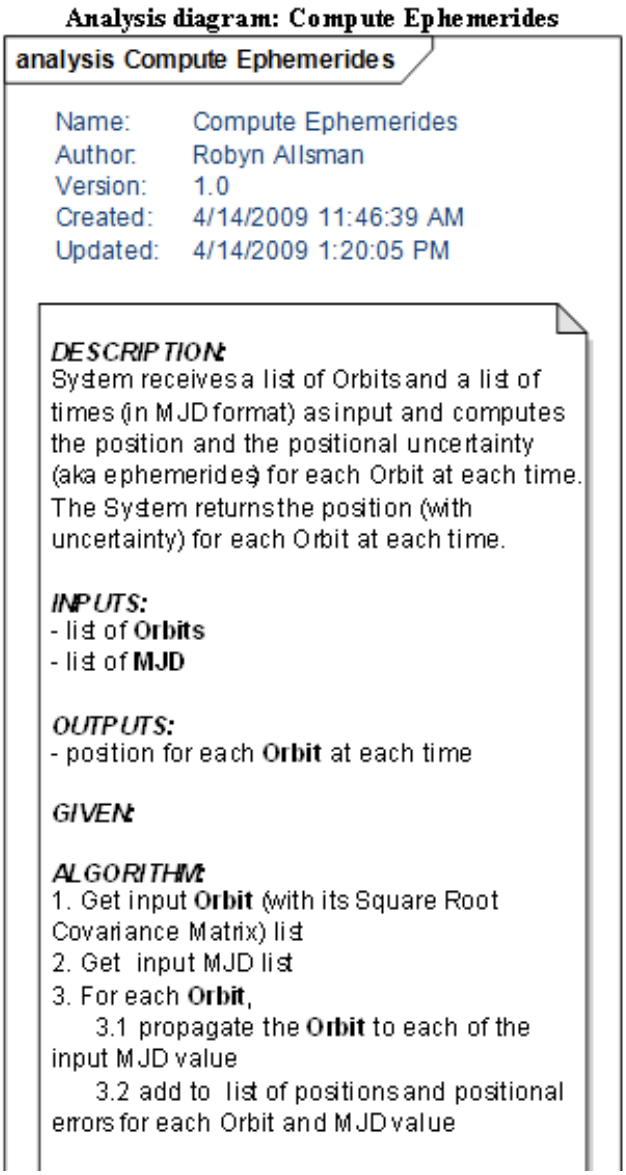



***Figure 7. This algorithm for computing Ephemerides will soon be detailed on a robustness diagram.***

## *Converging Towards a Solution*

You can see the benefits of a forward-modeling and reverse-engineering approach here, with a reverse-engineered database schema for MOPS (see Figure 8).

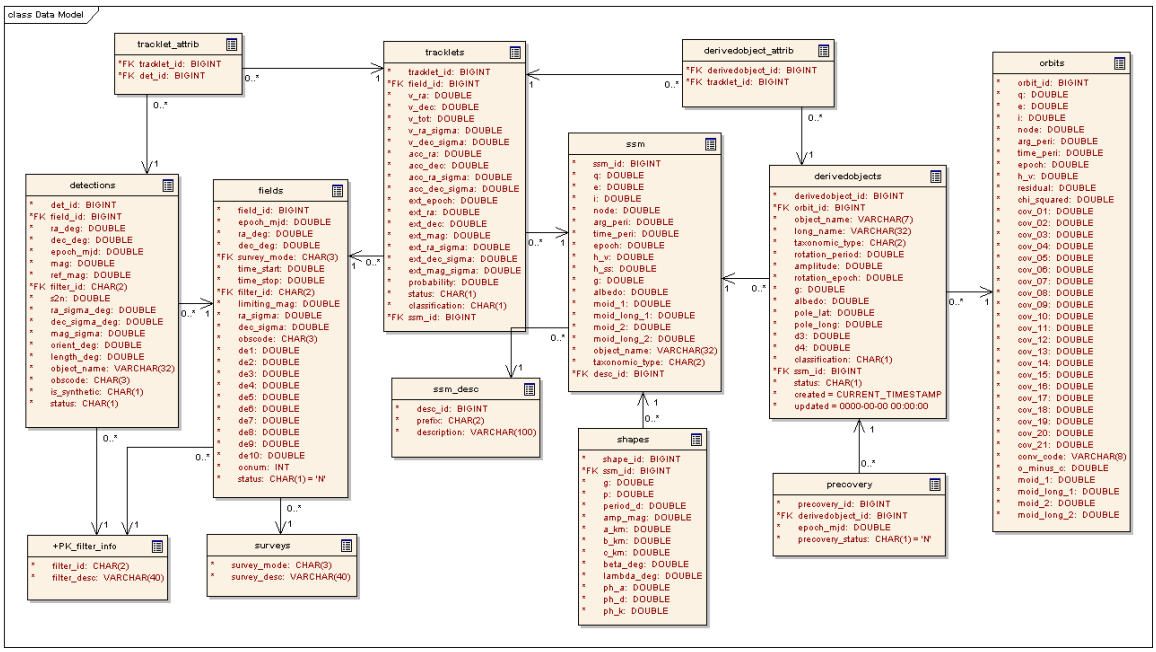



***Figure 8. Reverse-engineered MOPS database schema.***

By using LSST's R&D Phase for both modeling and prototyping different areas of functionality, the project minimizes risk. When LSST reaches the Construction phase, there will be very little doubt of a successful outcome.

### Modeling Tip: Modeling + Reverse Engineering minimizes risk

Once again, you can see how the capabilities of the modeling tool (in this case, EA's ability to reverse-engineer database schemas in addition to a wide range of programming languages) allows for a risk-mitigation strategy that's crucial for a project of the complexity of LSST.

## *Conclusion*

Detecting asteroids which might impact the Earth is a small, but important, portion of LSST's overall science mission. Hopefully, you've been able to follow the discussion presented in this chapter that explains how this capability will be provided by the Day MOPS image processing pipeline. If you have, that's a good sign that our tailoring of ICONIX Process has resulted in a UML model that successfully and unambiguously explains the design of the LSST software.