Understanding a Dispersed System with an Application Model

ABOUT THE AUTHOR

Scott Reece is a huge fan of the greater visibility and communication enabled by DevOps. As a Friend of Astah Scott has been successfully closing project communication gaps by using models and team building. With more than ten years of project management experience, Scott currently serves as a Partner Director at Inedo.

As organizations adopt DevOps, they not only release smaller changes sooner, but they also change the way they develop software. Additionally, applications are no longer seen as giant monoliths, but rather as dispersed across dozens of services that may be built on different platforms (from Node.JS to .NET), depending on the team and technology-du-jour. To complicate matters even more, as technology is changing, so are business requirements.

This is where an Application Model comes in. It saves time, ensures quality, and increases organizational knowledge of the application and its components which are constantly being changed and rebuilt. This article will explain what an Application Model is, who is involved in creating one, and how to start using it.

WHAT IS AN APPLICATION MODEL

FIELD VS DOMAIN

For example, a core function of banks is to underwrite loans, but they do many different types of loans. A field of underwriting could be as specific as commercial underwriting for loans over $5,000,000. This field will have many similarities no matter the bank because underwriting is a regulated practice, therefore the information used to make a decision is mostly identical.
However, the field isn't specific enough to create an application model because it doesn't include the unique factors of a specific bank. Things like the amount of weight a system gives to any number of loan factors (credit score, cash on hand, etc.), processes that are unique to an individual bank, or even specific nomenclature that bank uses. All of these factors go into creating the application model.

BITE-SIZED CHUNKS

For example, a bank will have different criteria for commercial underwriting depending on size, income, and even type of business. While the overall application covers all the different types of businesses being underwritten, different sections could easily be divided by requirement and modeled individually as part of the whole.

The application model is NOT a specific set of UML diagrams or models used in a direct 1-to-1 relationship with the code. Rather, it is a high level document that furthers understanding of the application and helps provide insights about that application.

The model needs to be approachable to personnel from many different business roles, including: developers, project managers, IT administrators, and even end-users. The array of people who might use the model should give a rough idea about technical level.

What makes the Application Model so different, so useful, and so important is that it defines not just the field of the application but the specific domain, which is also why it can be called a Domain Model.

In the end, the team gathered will build a complete model of what the application is and how it currently works. **It's important that the model being created reflects the actual application and not an idealized version of it.** Because many business applications are quite large, it often makes sense to have a broad model for the entire application, and several more intricate models for specific sections or features.

Lastly, it's important that the model is reflected in the codebase for easy understanding by the development team. If the model is meant to serve as a central hub for conversations about changing and improving the application, the decisions and insights found during those conversations can be relayed and documented for the development team more easily and much better if there is a tight relationship between the model and the codebase.

HOW TO CREATE THE MODEL

COMMUNICATION MATTERS

Communication is an extremely important skill for any member of the model team to have. For example, there could be an incredibly gifted developer who has vast knowledge of the application being modeled, but who speaks the local language as a second language and without advanced, and nuanced proficiency. This is fine when changes that they are working on are documented through a ticket system and can be understood by reading the ticket and looking at the codebase; however, it's not appropriate for the modeling team.

At best, that developer will slow down the entire modeling process because they will have to ask many questions to fully understand what is being said by non-developers. At worst, that developer will not ask questions to get the understanding needed, and therefore will not be able to share their insights with the modeling team nor adequately explain the model to fellow developers.

As mentioned earlier, the model must be approachable and understandable to individuals across many different business roles and with different backgrounds because it is the thing around which conversations about change and improvements happen. So, the first logical step in creating the model is assembling the team.

The core team should share several attributes for success. They should all (no matter their business title) be very familiar with the application and the current processes; this is not the place to learn about how things work.

At a minimum, the team should include developers, operations, and project managers, but knowledgeable end-users can also be invaluable. Everyone should also be above average communicators because they will be working to create understanding with the core team and relaying that understanding to non-team members.

Once a team has been created, the best start is to talk about the application itself as it is. The goal is to create a shared understanding of how the application works and how it's being used. Some good starting topics are:

- What does it do well, what doesn't it?
- What's missing from the application?
- What's in the app that's not being used?
- What other applications does this interact with and which processes?
- Is the application being misused (e.g.: a field labeled one thing used for another)?
- What is the process and data flow?
- How much overlap is there between data and objects?
- How granular are security requirements?
- What are the application deployment specifics?
- Which users will need access?
- Which APIs will be used?
- What are the Performance and Health Monitoring requirements?

MISCOMMUNICATION

For example, if an underwriter who works on business loans was describing the process to a group modeling an underwriting application, they might inform the group that a credit rating is one of the factors that is used in determining loan worthiness and needs to be recorded in the application. Credit ratings are something that everyone who has ever had a credit card, home loan, or even auto insurance is familiar with. However, it also presents an opportunity for Miscommunication.

Individual loans (for things like a mortgage) use a credit rating score that can go up to 850. A developer might know this from personal experience and create a field on in the application to record a credit score. Restraints on the field could be numeric only, with a range of 300 to 850 (this would help ensure that only valid scores were entered). However, businesses are given credit ratings with letters and symbols—not numbers—like AAA.

In this example, if there were no follow-up questions and no examples used even though multiple people on the team were using the same term and thought they were using it correctly, there is still miscommunication which would result in an error in the final software that would then need to be fixed.

Conversing around a whiteboard is often helpful for quick sketches and lists that everyone can see. You'll note that there are both non-technical and semi-technical topics above, so a good Project Manager who is familiar with both ends of the application can be invaluable in facilitating discussion.

As discussions progress (and it will take time), what is being developed by the modeling group is a shared understanding of the application, as well as shared knowledge of how to talk about it. This allows words to be precisely used with the assurance that everyone on the team is using them in the same way.

Sometimes this understanding is referred to as a Universal Language because through questions, miscommunications, explanations, demonstrations, and discussions, what was created by the core group is a way to easily and fluently talk about the application without any misunderstanding.

Using this shared understanding documenting the application should be achievable. Because the Application Model isn't a set series of diagrams or maps, there isn't a definitive template for creating one, but many teams have found that a combination of Data-flow, Activity, Use Case, Sequence, and Component diagrams are at a high enough level that everyone can understand and use them, while still being technical enough to be useful for the development team. It can also be helpful for text documents of notes, snippets of sample code, and even photos of whiteboard drawings to be part of the model.

MODELING FOR DEVELOPMENT

MODELING TO SAVE TIME

For example, if a new federal regulation were introduced, and several new pieces of information had to be logged for loans of a specific size (or greater), the domain model might show:

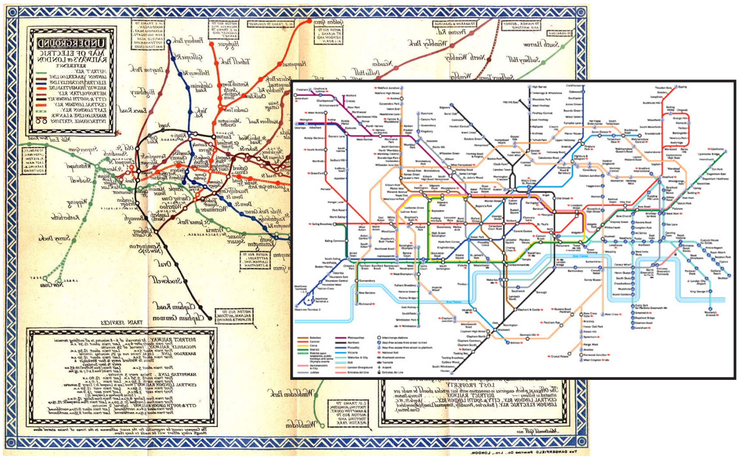
- That some information was already being collected and only needed minor updates.
- None of the information was being collected at that size loan; but it was for a different size, and that function could easily be added to that section of the application.
- That data wasn't being collected on loans of any size, and a conversation might ensue to determine if it was worth including the same data requirements on loans other than newly regulated.
- That information could be calculated based on existing procedures and would only need to be accurately logged and kept.

Of course, a series of diagrams and notes can be messy to use and organize, so oftentimes, a mind-map can be helpful as a method of keeping track of the disparate parts and diagrams, and maintaining ease of use.

Having a model is all well and good, but how does the model help with development? By defining the application not just from the developer point-of-view, insights will have been gained about what needs to be tweaked, what needs a major overhaul, and what needs to be demolished completely.

By not just knowing what needs to be changed, but why it's broken, and what lead to the break makes changing the software not just an issue on a ticket, but a complete picture. Also, noting application traits like data overlap, object reuse, and layers of abstraction allow the core development team to be able to develop and iterate quickly without worrying about changing functionality in an unplanned way, thereby saving on testing time and costs.

Surely, a model is only useful so long as it's current. Any changes to the application need to be entered into the model. An outdated model can be worse than having no model at all since conversations that occur around it are started with false data and assumptions. If a model isn't maintained, then all of the organizational time and effort that went into it and shared knowledge it created are lost.



MODELING FOR OPERATIONS

DESIGNING CHANGES WITH DEVS AND OPS

When evaluating an organization for a loan, the credit rating of that organization will need to be acquired. There are vendors that perform the task of rating other organizations, and individual banks will have preferred vendors for that metric.

If the task of gathering these ratings were to be automated, a developer might want to implement a simple process where the application queries the vendor through their API and waits for a reply. This is simple, straightforward, and only requires two pieces of data: the query and the report. However, Operations would certainly step in because this scenario would be a security risk. It would either involve two-way communication, or require leaving the communication channel open for an extended period of time.

A better solution that could be created with input from operations might involve an additional piece of data: a query ID. The interaction would then consist of multiple steps such as a query being made, a query ID getting returned, and then, at regular intervals, the vendor being sent the query ID until the report is ready.

Maintaining the model is key to ensuring its usefulness long after it is first ‘complete.’ Once a model is created, it can also be helpful for the members of the modeling team to present and discuss it with their department coworkers. Sharing knowledge beyond just a few people ensures that no matter what personnel changes happen, or what development trends and methodologies are implemented in the future, the Model can carry on and maintain usefulness as a teaching and documentation tool.

As applications and infrastructure become more and more intertwined, it becomes critical that both Developers and Operations are involved with creating an application model. Any changes that will be considered based on the model will have to take into account things like security, application monitoring, and the host of other criteria that Ops deals with.

Operations will be involved in the application development, either from the start where they can help define practices and procedures that the development team must follow, or before the application is released and they raise red flags and security issues which the development team must comply with before the changes are released. It’s much more efficient for them to be involved from the start.

GET MODELING!

Organizations have proprietary software because it is a competitive advantage for them to create their own applications based on their business needs. Because business needs are constantly changing and evolving, the software that encompasses them must change and adapt, as well.

Finding methods that allow for quicker changes while ensuring better change quality should be a goal in any organization. Creating an Application Model, and the shared understanding that goes with it, can be an invaluable part of any application that faces maintenance or overhaul.