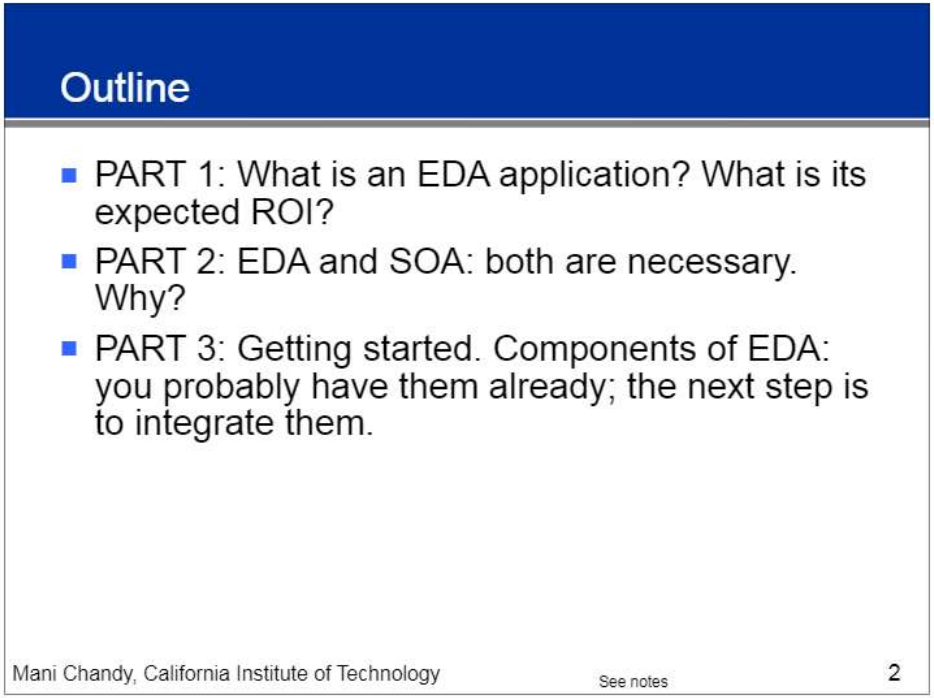
Thanks to Gartner and Roy Schulte for the opportunity to speak today. I especially want to thank Roy and David Luckham for the excellent presentations they've given on Event-Driven Applications. I am going to talk to you about somewhat different aspects of EDA than they did. I hope our collection of presentations is synergistic and helps you to determine whether and how EDA is useful to you.

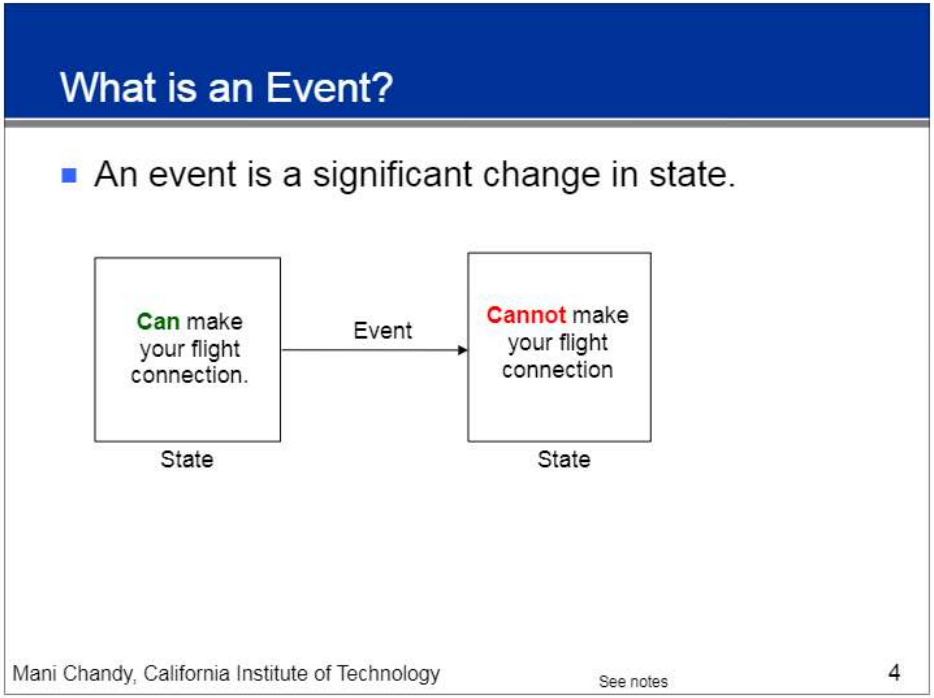
Dave spoke to you this morning about business intelligence that is obtained by analyzing a repository of event-messages from the event cloud. Roy has talked to you about the role of the enterprise service bus, the EDA asynchronous loose-coupling patter of application integration, and the benefits of EDA for the enterprise. I'd like to help round-out these discussions by having a conversation of ROI for event applications.



My primary goal is to help dispel the misconception that EDA is of interest to you, the participants in this conference, as a technology of exclusively research interest that may become useful only after 2008. Many of you are already using EDA. Most of you can benefit from EDA applications now. So, a focus of this talk is on understanding EDA and its ROI… trying to evaluate the true costs and the true benefits.

The talk is in 3 parts. The first part describes basic terms and then does an evaluation of the costs, benefits and ROI. A key point in the analysis of costs is that the cost of tooling for EDA is small for 2 reasons: (1) the cost of your time and the time of business users in specifying and developing the application dwarfs the costs of tools, and (2) the tools you need are incremental --- a layer on top of existing tools. So, a second part of the talk deals with the complementary nature of SOA and EDA. The third part of the talk suggests steps to get started.

The business proposition of EDA is that it helps enterprises respond to events. These events may be threats, opportunities or other kinds of situations that require timely responses. Keep these questions about the business proposition in mind: How important is it for your enterprise to respond to situations as they occur? And how much can IT in general, and EDA in particular, help respond in an appropriate and timely fashion?

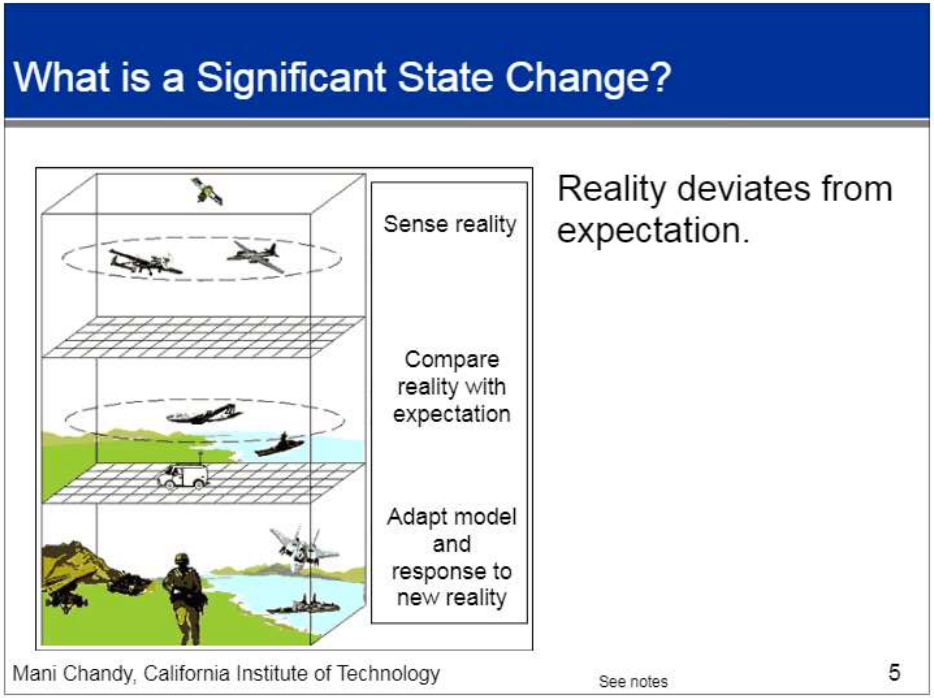


An event is a significant change in state. I want to emphasize that an event deals with CHANGE.

For example, you have a flight back home tomorrow with a connecting flight in Dallas. Delays in San Diego due to airport congestion result in your being unable to make your current flight connections in Dallas. Something CHANGED. Previously you could make the connection, and then a change --- airport congestion in San Diego --- caused the connection to be unavailable.

Note that the event is NOT that you cannot make the connection. The event is the change in state.

How do you know that the event occurred? How do you respond to the event? These are parts of an event-driven application and are discussed in the talk.



An event is a significant state change. So, what is a significant state change?

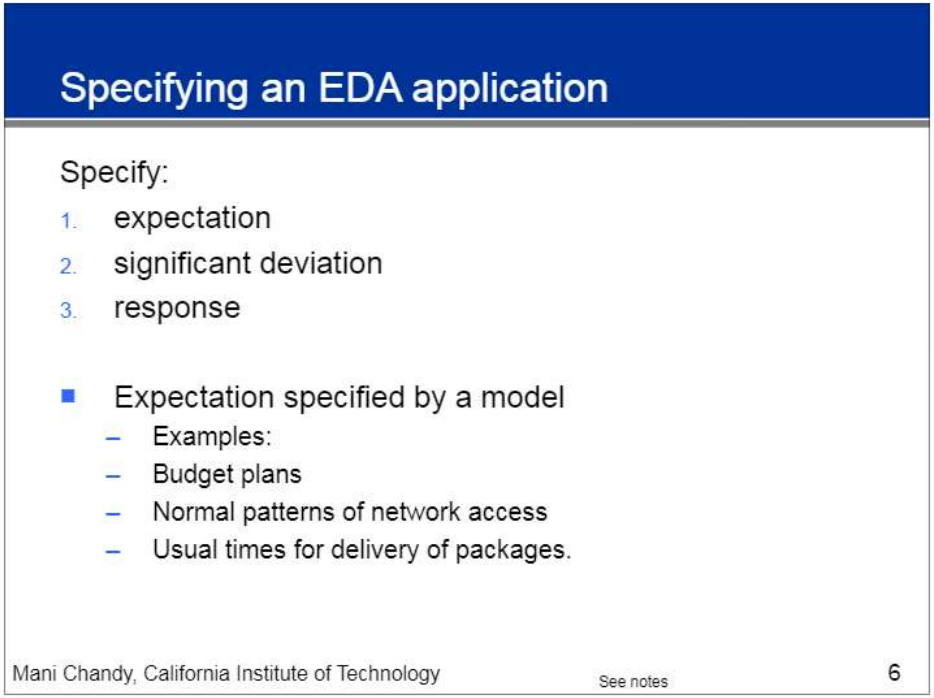
A significant state change is a change in the "real" state that deviates from the "expected" state, where the deviation is significant enough to cause a change in plans and necessitates some sort of response.

Continuing with the trivial example of making connections on your trip back home tomorrow… consider a state change where because of a change in head winds you are likely to land 10 minutes early. That is a state change. But, it is not significant if it does not change your behavior or plans.

The military did the earliest and most detailed work on event-systems as part of C-cubed I (Command, Communication, Control, and Information). They sense reality, the "real state" of the environment by sensors on planes, satellites, ships, and information from war-fighters. Information from multiple sources is integrated or fused to obtain an estimate of the real state of the system and its environment. Then, if this real state deviates from expectations, an event is generated that causes the previous expectations to be updated, and perhaps a response to be undertaken.

I'll keep coming back to the concept of an event as being a deviation of reality from expectation, and contrast it with the more generic notion of events.

For now, let's stay with the definition of events as: an event is generated when reality deviates significantly from expectation.



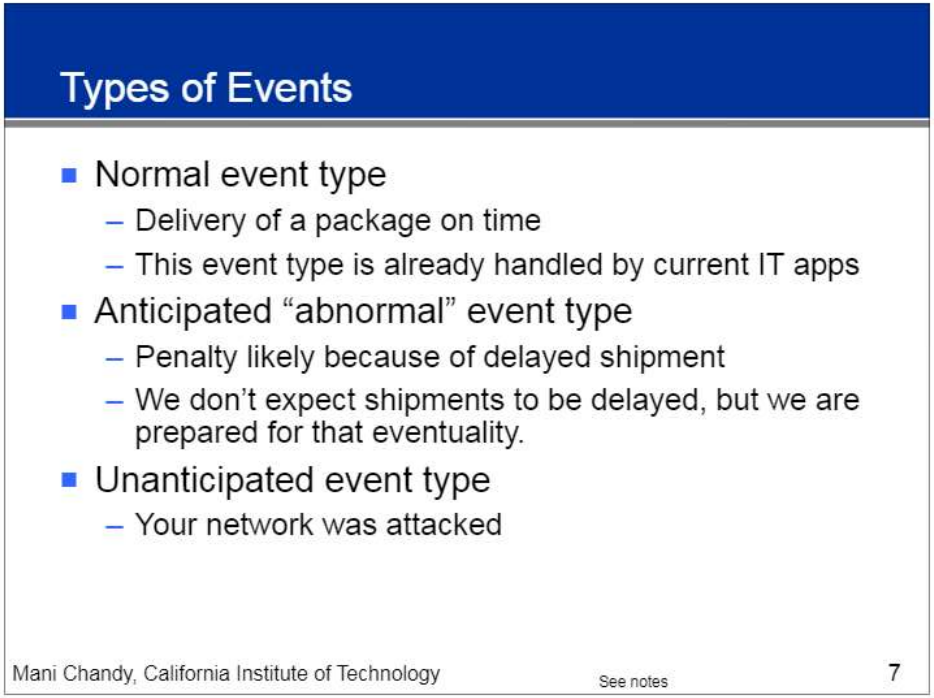
Specifying an EDA application, therefore, is a specification of an expectation of the state of the world ---- the system and its environment, a specification of what constitutes a significant deviation of reality from this expectation, and a specification of the response to be undertaken when reality deviates from expectation.

An expectation is specified by some form of model. In accounting, a model is a plan for income and expenditure. Deviations are the differences between "plans" and "actuals."

For applications dealing with intrusion detection into networks, an expectation is specified by "normal" patterns of network access. Reality deviates from expectation when there are abnormal access patterns.

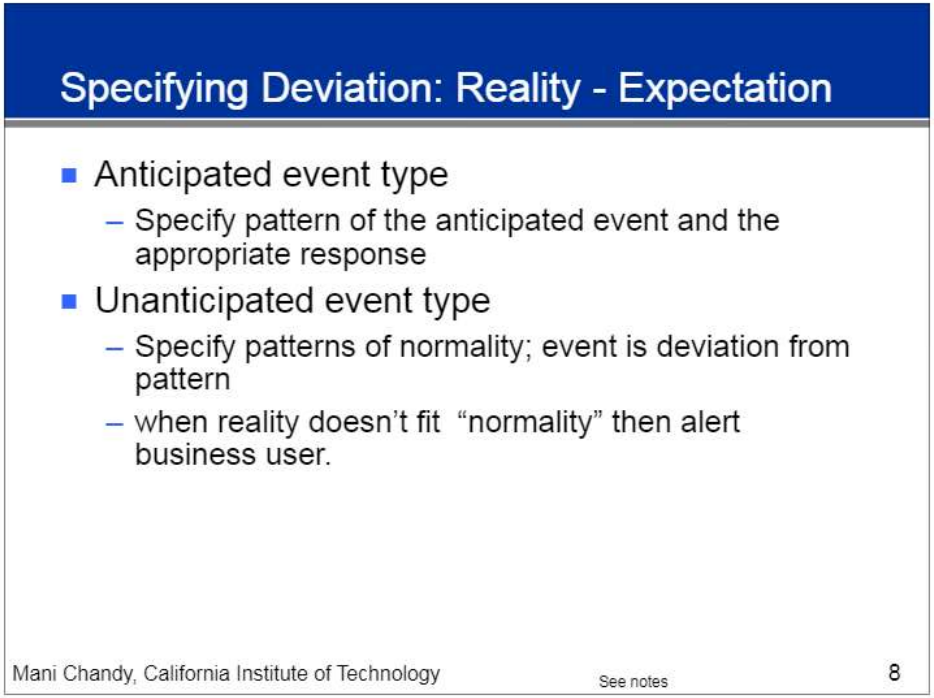
For logistics applications, expectations are specified by a model of delays in package delivery. Reality deviates from expectations when packages delivered later than predicted by the model.

Of course, there are "normal" events too… the delivery of a package on time, which is completely expected and fits the model perfectly, is also an event in the traditional sense; and the delivery of a package is logged, and this log may be used later in audit trails and billing. Certainly, the information that a package was delivered should not be lost by the information system, even if the delivery of the package is completely anticipated. It helps, however, to separate the two types of events: the normal type which is already handled by existing IT applications (possibly using SOA) and the abnormal type which isn't handled adequately today.



I'll separate events into 3 categories. Normal events already handled by your IT applications. Abnormal events that may not be handled by IT applications, where the enterprise anticipates that these abnormal events may happen on occasion. The third type of event is one that is totally unexpected by the organization: it is not only abnormal but it is an event for which the enterprise has no contingency plans.

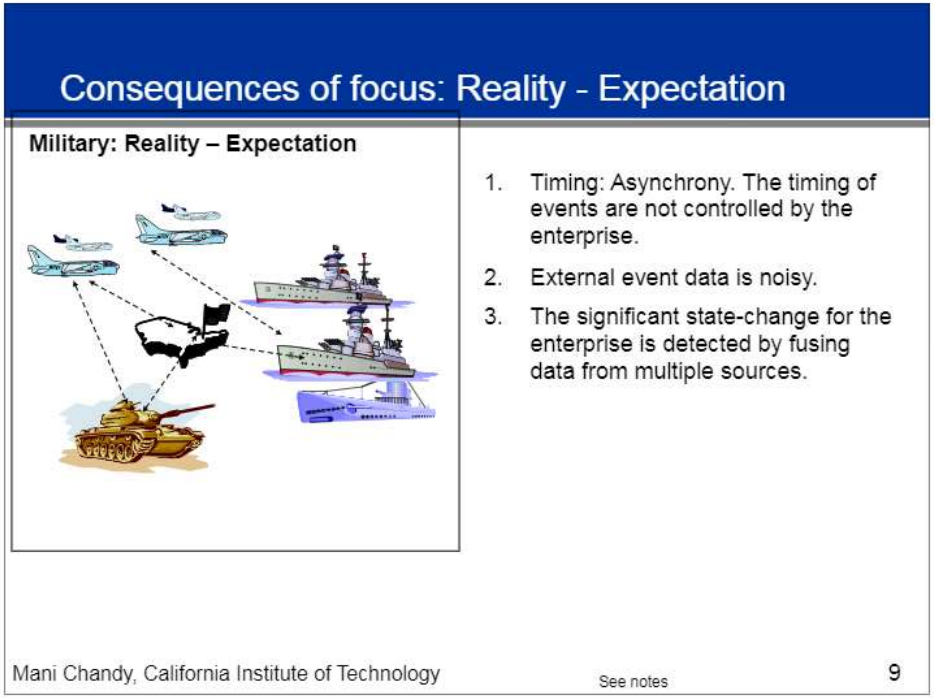
This talk deals with the latter two types of events: the anticipated abnormal event and totally unanticipated event.



You specify the anticipated event type by specifying the pattern that you anticipate that indicates this event type. The pattern may deal with a history of events and with data in a database. The key point here is that you KNOW what you are looking for. For example, if your enterprise is a health-care insurance provider and the application deals with detecting fraud from opticians, you know certain patterns that need to be investigated.

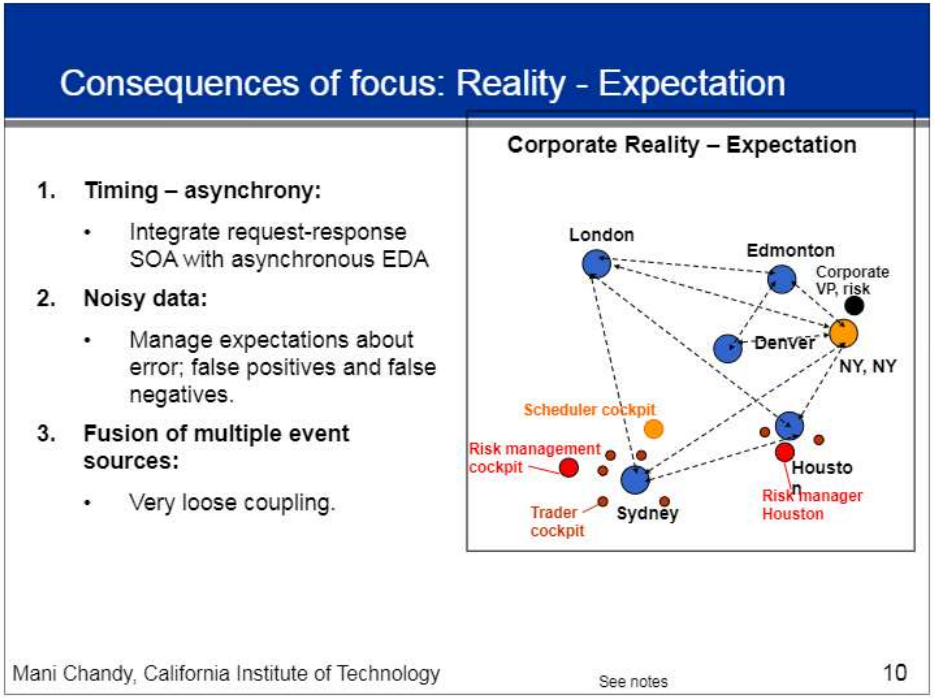
For the unanticipated event type, you don't know what patterns indicate an abnormal situation. In this case, you specify patterns that indicate normality. Any pattern of events that does not fit this normal pattern is considered unusual. For example, if your application deals with intrusion detection into a building, you may specify normal patterns of behavior in entering and leaving the building.

The response for the anticipated event type can be tailored for that event type. Obviously, the response for the unanticipated event cannot be tailored for the event; so the response is to alert a person who can then determine whether the unanticipated event is a threat or an opportunity or a false alarm.



Situational awareness, and UDOP (user-defined operational pictures) are terms widely used in the military. All components of a task force need to be aware of the global situation of a battlefield. Each user needs to have an accurate picture of the aspects – relevant to his or her role – of this operational picture. Likewise, all components of certain enterprises need to be aware of the global situation and different people and applications need to be aware of different aspects.

Estimating critical events in the external environment has three important consequences: firstly events happen when they happen --- the timing of events is not controlled by the enterprise. Secondly, external event data is more noisy --- more error prone --- than data within the enterprise. Thirdly, a significant state change of the external environment is detected by fusing data from several data sources. This is quite different from publish-subscribe on a message queue where the subscription is on a single message. In EDA applications, an event may be created based on HISTORIES of messages from MULTIPLE sources.

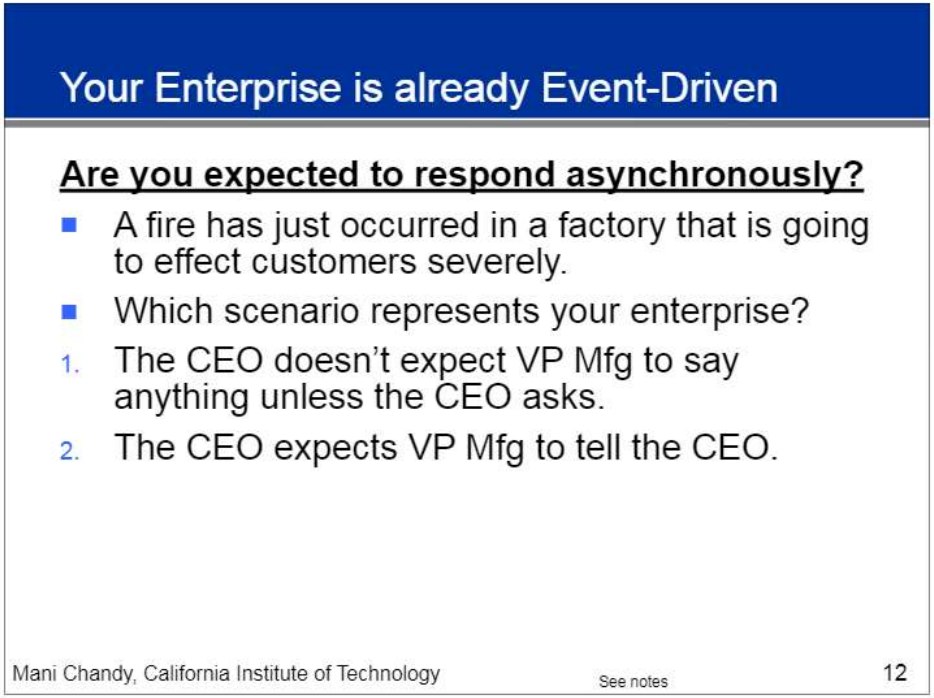


The picture on the right shows situational awareness in the commodity trading unit of an energy company. The company has overall goals for its Value at Risk, each location (such as the Houston office) has its own goals for VaR, and different people such as a trader in Edmonton has his or her own UDOP represented here by a "cockpit" tailored for that person.

The essence of situational awareness is that absence of communication implies that reality matches expectations (models). All the components and people just cannot keep communicating all the time --- that would result in too much traffic.

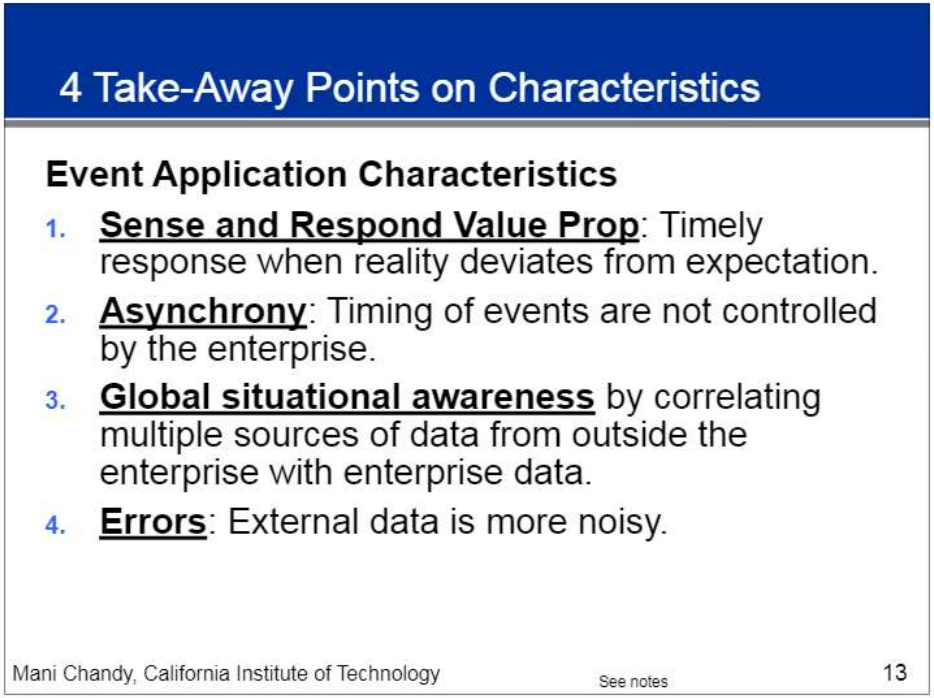
The three consequences of dealing with the external environment have an impact on the architectural pattern used for event applications. Firstly, since the timing of events is asynchronous we use EDA combined with SOA. Secondly, since the data is fundamentally noisy, we must accept the possibility of error --- both false positives and false negatives. Error is not a critical concern in SOA applications, but is in externally-facing event applications. Thirdly, a consequence of fusing data from multiple sources, especially multiple external sources, implies that the coupling between components must be even looser than for SOA.

Here are two rhetorical questions for you. Of course, all of know what the answers are for everybody in this room. Of course, your enterprise is aware of what is going on in the world outside the enterprise. And of course, there are people in your enterprise who are putting 2 and 2 together to get 4. The point of these questions is that you already have EDA in which the components are people.



Here's another rhetorical question. Of course, the CEO expects VPs to inform the CEO when something unusual happens. The point of this question is to emphasize the contracts that are explicit or implicit in collaborative human relationships: I have a responsibility to tell my collaborators, in a timely fashion, when something they don't expect happens to me or my environment.

These rhetorical questions emphasize that your enterprise has always been event-driven. You don't need to convince lines of business to BECOME enterprise driven; they already are.



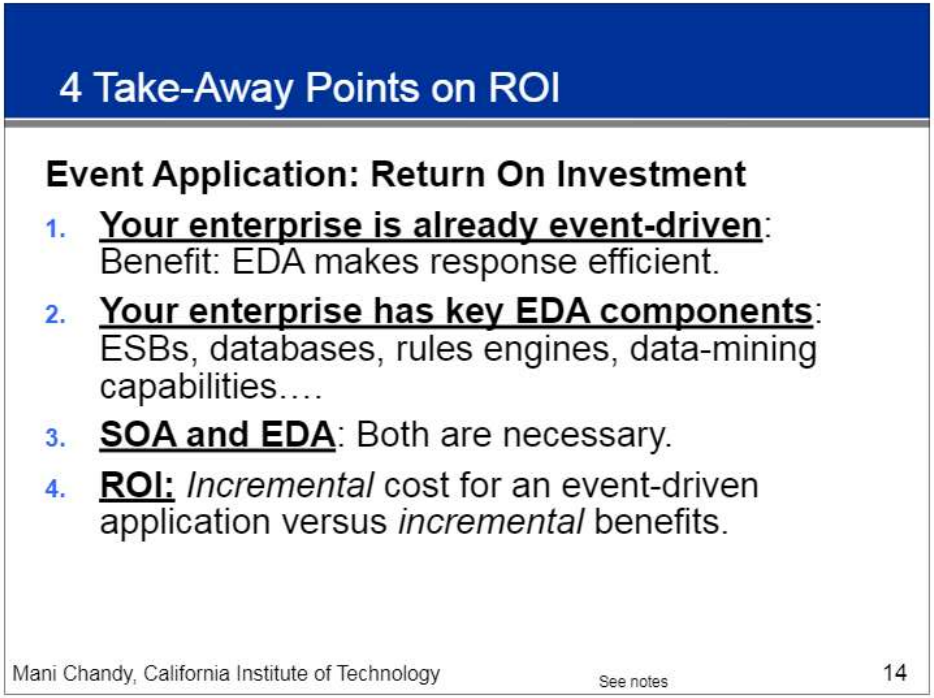
Let's review the 4 key characteristics of event applications.

Firstly: The value proposition is responding in a timely fashion when reality deviates from expectation.

Secondly: Asynchrony is a fundamental aspect of the problem; it is not an artifact of the technology.

Thirdly: These applications require correlating HISTORICAL data across multiple event sources.

Fourthly, we should acknowledge the possibility of error.



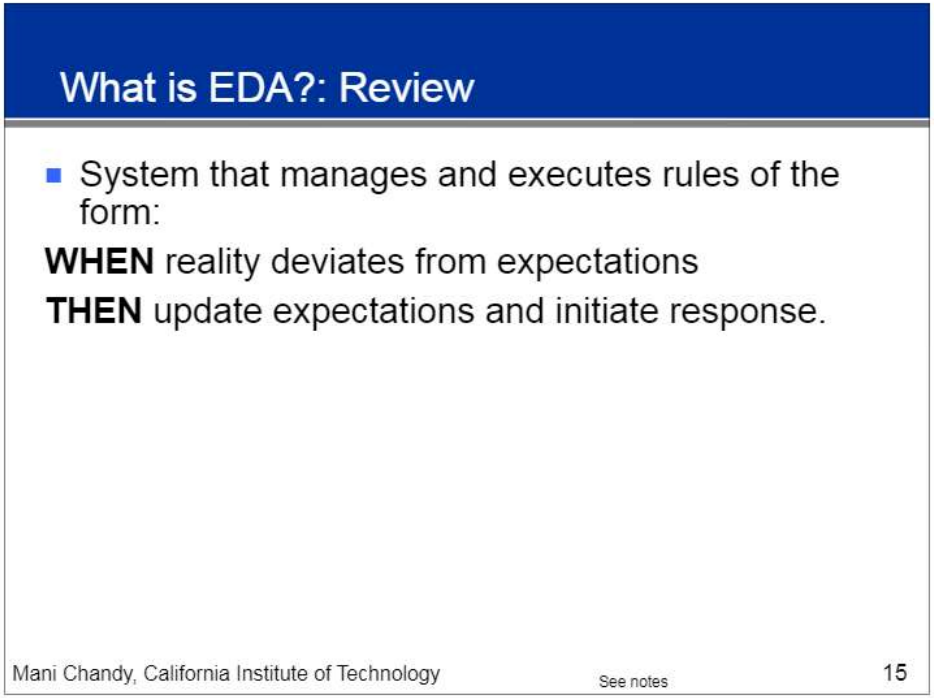
Let's review the 4 points about ROI.

Firstly, your enterprise is already event-driven. So the return is in making your enterprise more efficient. You aren't changing the fundamental nature of the enterprise.

Secondly, you enterprise probably has the key components of EDA already. You don't need to invest huge amounts in new tooling. The challenge is to repurpose the IT tools you have to make your enterprise more efficient in responding to events.

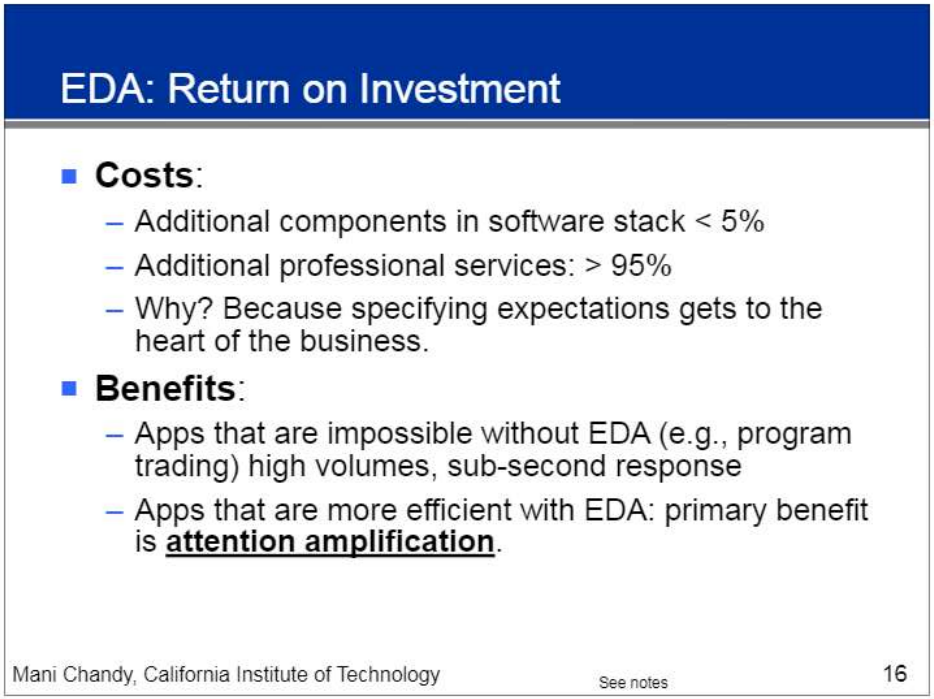
Thirdly, your investments in SOA are going to pay off with EDA as well. EDA and SOA are complementary technologies, not competing technologies.

Fourthly: the ROI is the ratio of the **additional** cost of repurposing your existing tool-set and the additional benefits you derive by making your enterprise more efficient. For most enterprises, EDA applications don't require totally new tools and totally new applications.



An event-driven application can be defined as one that executes when-then rules. When an important situation occurs, then respond appropriately.

This definition generates lots of questions. Firstly, who specifies the rules? How many types of rules? Could rules be at cross purposes? How are when-clauses evaluated? What sorts of then-clauses are used in enterprises?



The development of an EDA application is not cheap. But this is not because tools are expensive. The cost of added tooling is probably at most a tenth of the total cost. Most of the cost deals with the time required by IT and business users in specifying the application, trying it out, and tuning it.

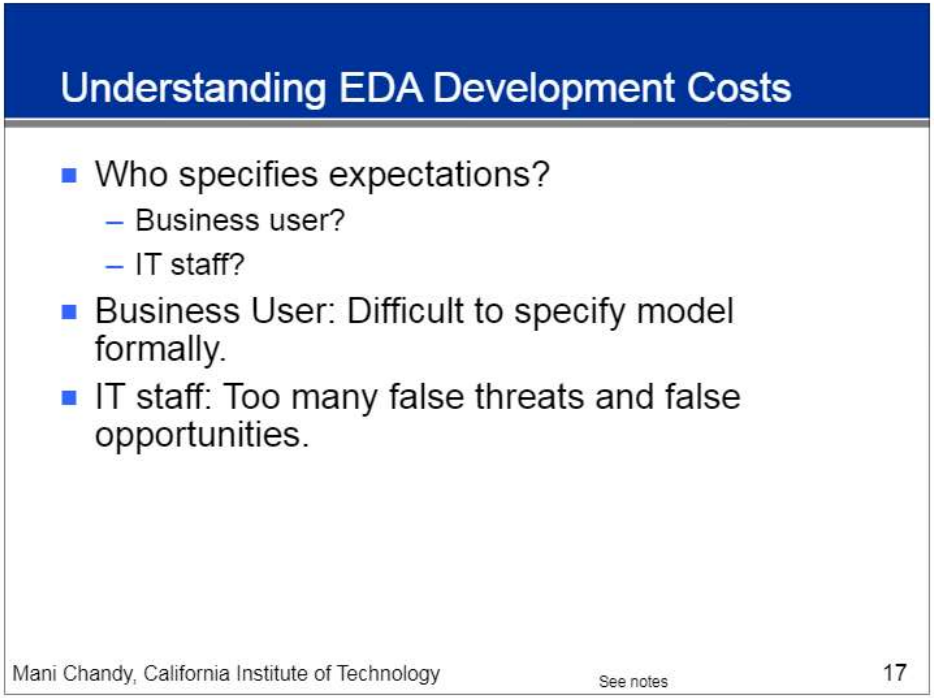
Why do EDA applications take so much of business user's time? Because it deals with the heart of the business: how are plans specified? What constitutes acceptable performance? When actuals deviate from plans what should happen? There's no easy way to answer these questions without getting the business user's time.

This time is often the most expensive part of application development. I'll talk more about this shortly.

The benefits of EDA are different for two kinds of apps: firstly apps that cannot be developed without EDA and secondly apps that can be developed using traditional methods but that are more efficient with EDA.

Apps that require specific design strategies to deal with events are those for which volumes are extremely high and where response times must be in sub-seconds. Traditional database-centric solutions don't work for these situations.

In many applications, however, loads and response times are not severe. The benefit to using EDA designs is in making the system more efficient. I've heard, as you must have, that a pilot of a plane spends 99.99% of the time in sheer boredom and 0.01% of the time in sheer panic. A value proposition for EDA in the second kind of app (non-severe volume and response times) is attention amplification: the business user, like the pilot, is alerted and given the right tools to deal with the panic situations. This frees the business user to deal with other issues the remaining 99.99% of the time.

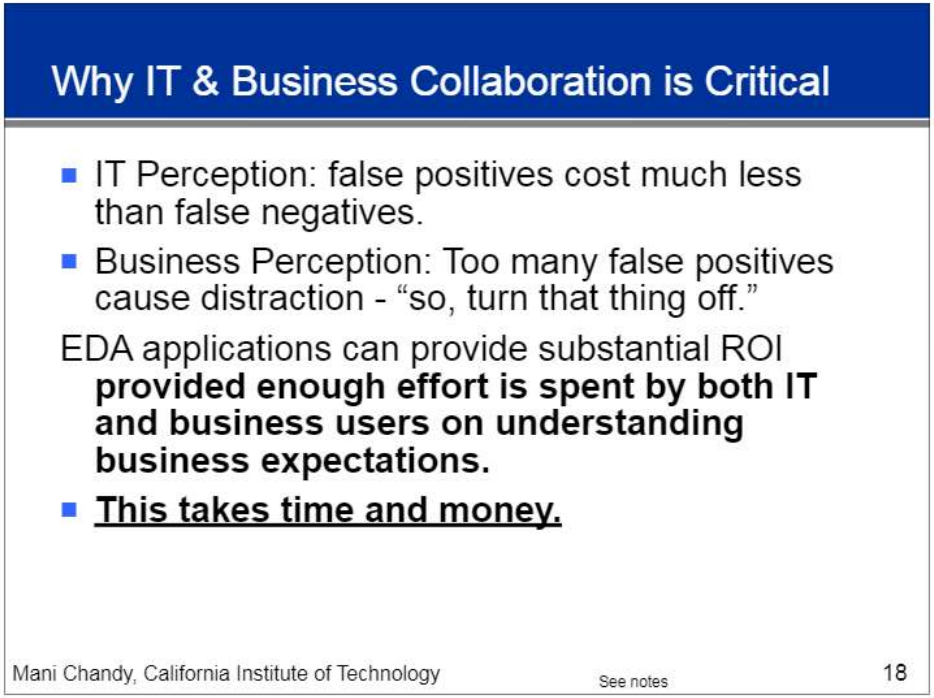


Let's get back to why development costs are significant. Why do these apps require the business user's time?

The value prop of EDA apps is that the enterprise responds in a timely fashion when reality deviates from expectation (a model, a plan). Who specifies these expectations? Consider two possibilities? The expectations are specified by (1) the business user or (2) by an IT person.

Business users don't have the time to learn notations that help them specify expectations (models) in a formal way so that an application can run them.

IT staff may not know the business well enough to specify plans, deviations and responses accurately. So, an IT person is likely to specify systems that result in more false positives ---- false threats and false opportunities ---- than is acceptable. Let's consider that in more detail next.



In most of the event-driven applications that I've seen, responses are mediated through human beings. Today, there are few applications that automatically invoke a mechanical response – say automatically carry out a trade on an arbitrage opportunity. Because human beings mediate responses, IT people tend to think that the cost of a false positive is low…. After all, what is the cost of sending an alert to a business user? The business user looks at the alert, perhaps analyzes it for a short while, and then discards it. If however, too many false positives are generated, the business user will turn the system off.

So, the system cannot be specified by the IT person alone. Nor can it be specified by the business user alone. It takes cooperation between the business user and the IT person. And that takes time and money.

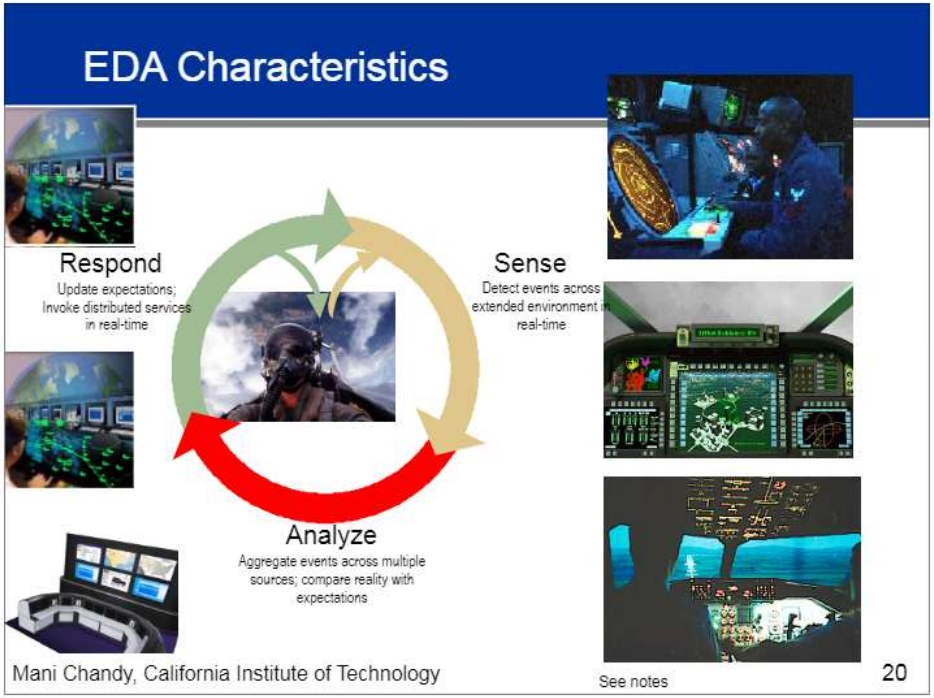
Yes, the investment is substantial. But, its an investment in time not in tooling.

Tooling will improve in the future. And this will reduce time and effort…. But the cost of time, effort and collaboration between the business user and the IT person will continue to be the large percentage of the overall cost of development of EDA applications. So, don't wait for better tools. Start now.

Outline: PART 2

- PART 1: What is an EDA application? What is its expected ROI?
- **PART 2: EDA and SOA: both are necessary. Why?**
- PART 3: Getting started. Components of EDA: you probably have them already; the next step is to integrate them.

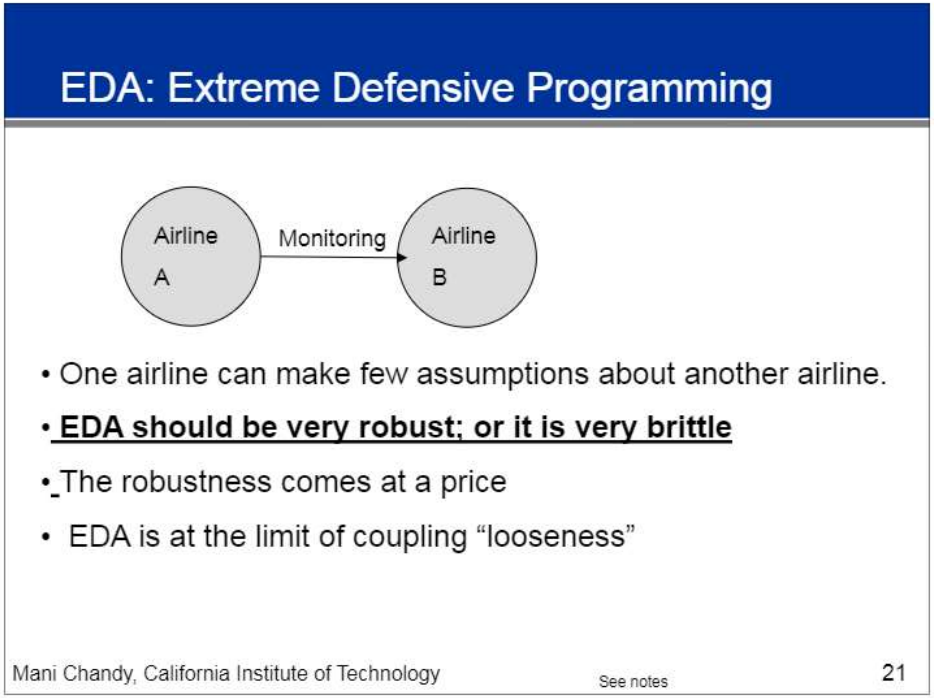
We've finished the first part of today's discussion. Now I'd like to emphasize the point that EDA and SOA are complementary and both are necessary.



A central characteristic of event-driven applications is its composition of three basic features: sense, analyze and respond. The sensing part obtains information from within and outside the enterprise. This part obtains the data that defines "reality." The analysis part correlates or fuses data to determine an estimate of the current state, and to compare the real state with the plan. In effect, the analysis part continuously evaluates the **when** clauses of when-then rules by fusing information from all the sensors and determines if the **then** clauses should be executed. The responders execute the **then** clauses. This part responds when reality deviates from expectation by modifying expectation (the plan) and executing responses such as sending alerts, invoking applications, and initiating operations by actuators.

This is a depiction of the warfighter's approach to managing threats in his/her global environment:

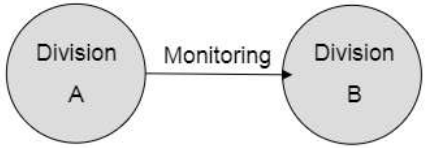
-He/she has sensors that detect threats –enemies, mountains, low fuel- friendly forces, air traffic control commands. The system has responders – such as alerting the pilot or initiating an automatic response without pilot mediation. And the IT system coupled with the pilot continuously analyze the streams of data arriving from sensors.



An airline monitoring another airlines web sites has to poll the competitor's web site, and has to deal with unexpected changes to the site. The application must be designed to be extremely robust or it will be very very brittle. I know from sad experience working in the trenches.

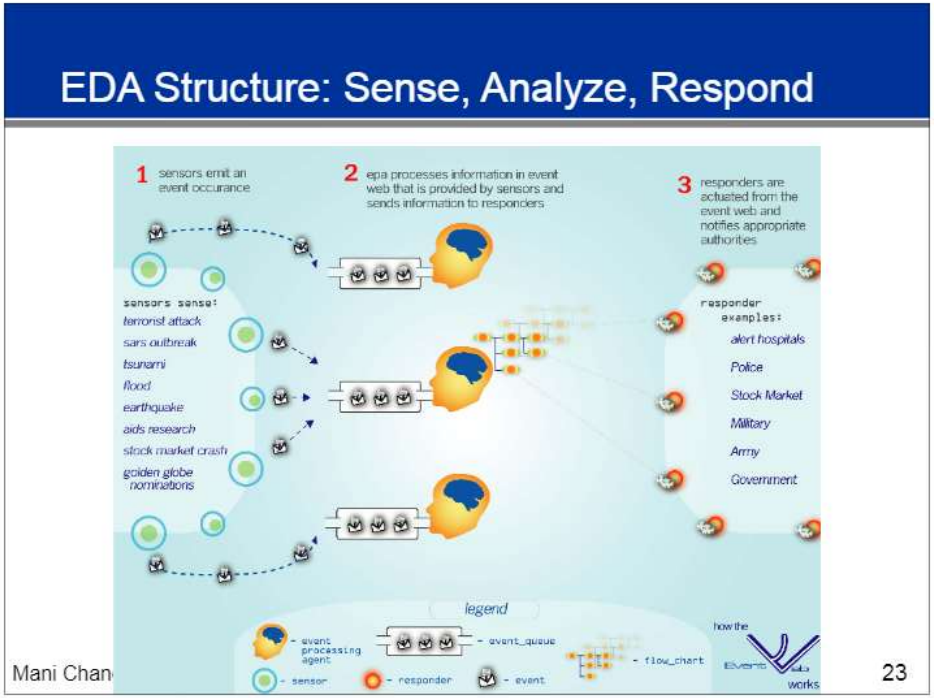
In this sense we can think of EDA as taking defensive programming to the extreme.

EDA: Extreme Defensive Programming

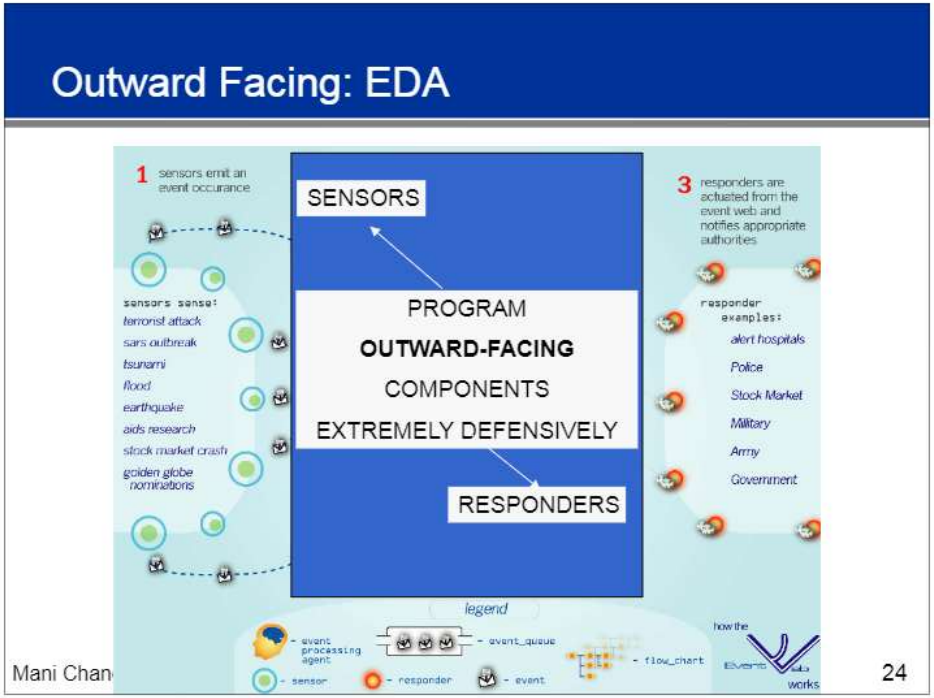


- One division can make few assumptions about another division.
- **EDA should be very robust**
- The robustness comes at a price
- EDA is at the limit of coupling "looseness"

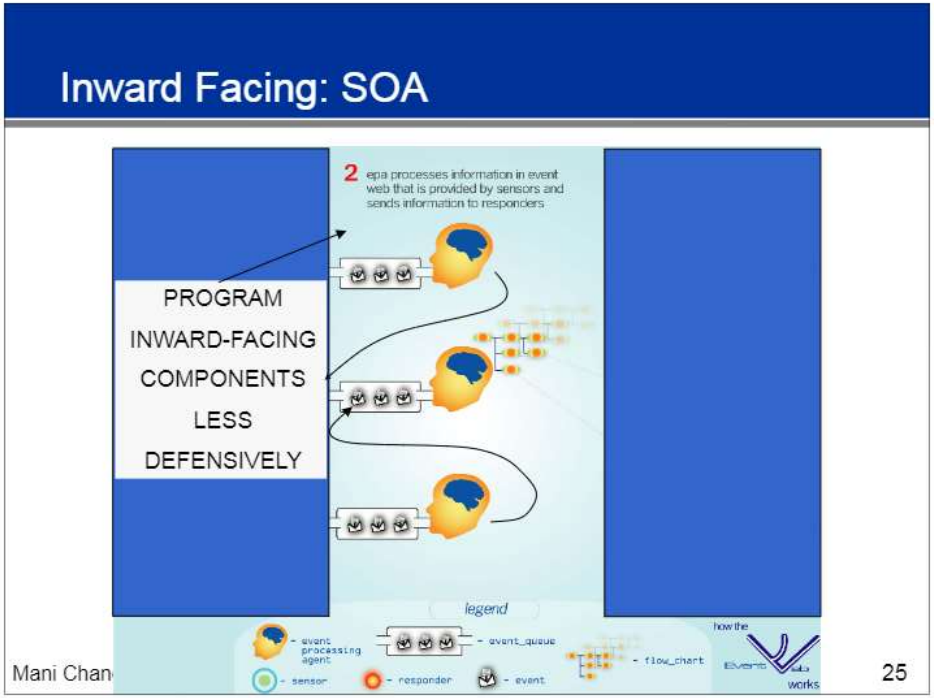
One division of an enterprise can treat another division just as one airline treats another. In this sense, coupling within an enterprise could be extremely robust. But, we must remember that this robustness is achieved at a substantial cost. Developing applications that are based on contracts between components is much easier than when no contracts exist.



This slide is from articles by Jonathan Lurie and me in the online magazine developer.com. The articles are on sense and respond systems. Sensors are shown on the left, the event processing agents (EPAs) that analyze sensor data are shown in the middle, and the responders to the right. Let's look at the roles of EDA and SOA in this picture.

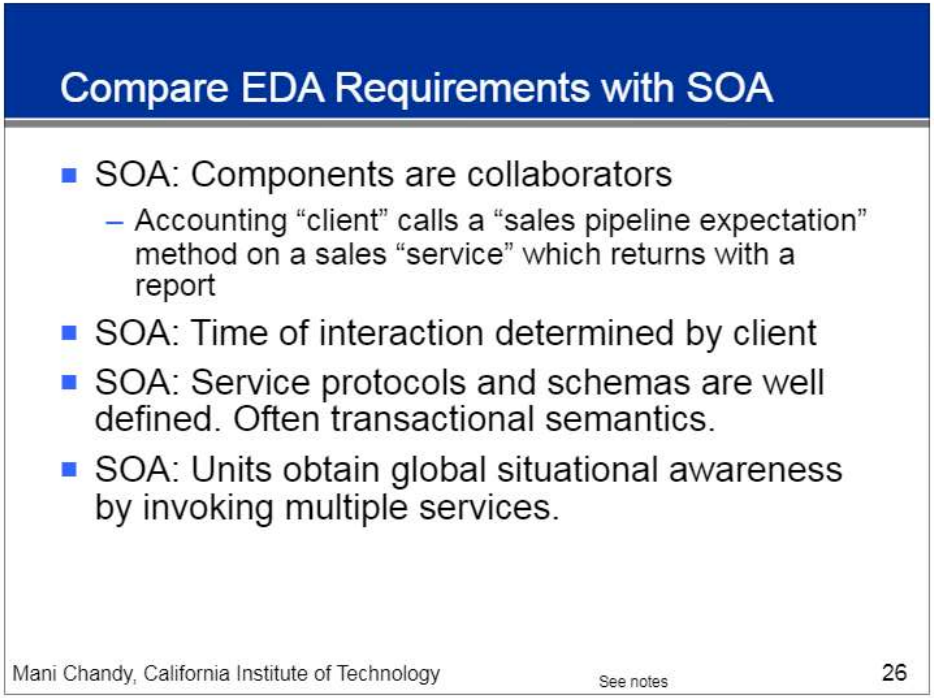


The "outward-facing" components of an event-driven application are programmed extremely defensively. The sensing portion may have to poll the environment rather than have data pushed to it. The outward-facing part of the application has to use EDA.



The inward-facing components can be, and perhaps should be, designed using SOA. This is because within-enterprise applications can depend on each other. They can provide services that are invoked by other services, with well-defined interfaces. Also, operations may need to be transactional.

EDA for outward-facing and SOA for inward-facing is somewhat simplistic, but gets across the idea of the appropriateness of each kind of architecture.



SOA and EDA are complementary and compatible. Use SOA and EDA for different goals. Let us take an extreme – a caricature – view of each to highlight the differences between the two.

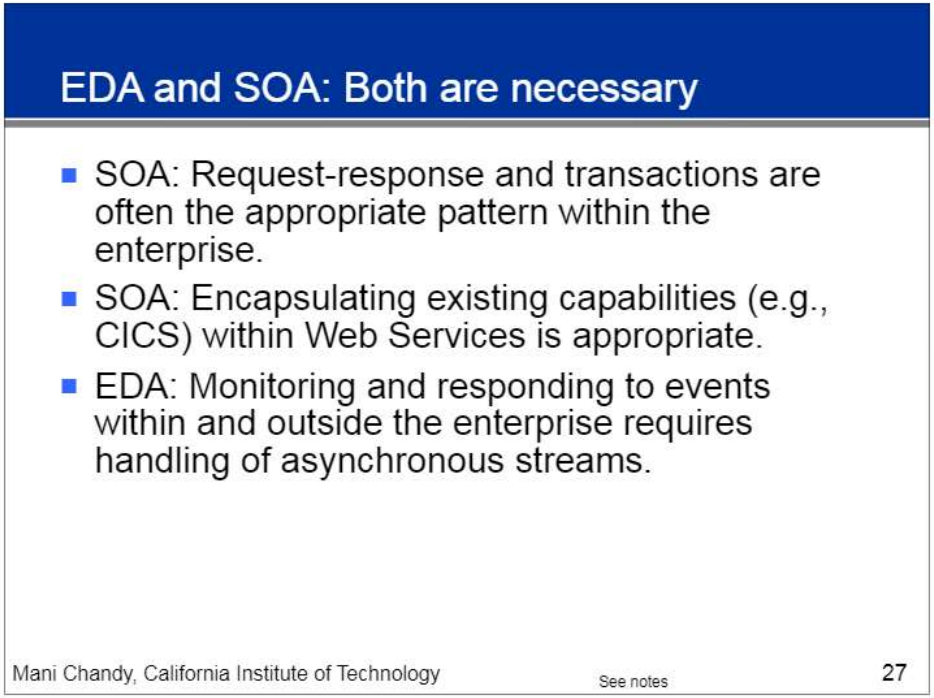
The components of SOA are collaborators. SOA is an integration framework among collaborating entities. EDA may interact with components outside the enterprise – and these may not be collaborators.

In SOA the time of interaction – the time at which a service is invoked – is determined by the client. In EDA the times at which significant events occur may not be controlled by the enterprise.

In SOA service protocols and schemas are well defined That, after all, is the goal of WS*. By contrast, in EDA, protocols and schemas may be less well defined.

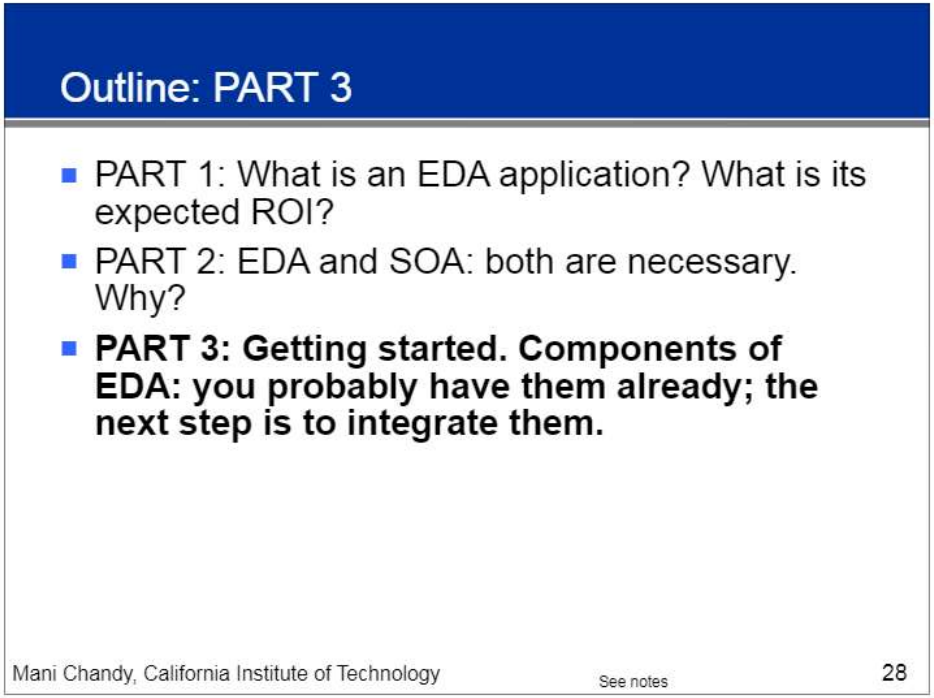
In EDA absence of communication conveys information: reality fits current expectations. In SOA absence of communication conveys nothing: the server does not have a contract to tell the client when reality deviates from the client's expectations.

I want to emphasize that the points in the slide take an extreme view of EDA and SOA to delineate the differences. They are complementary: Use SOA where synchronous interactions make sense; use EDA where asynchrony makes sense.

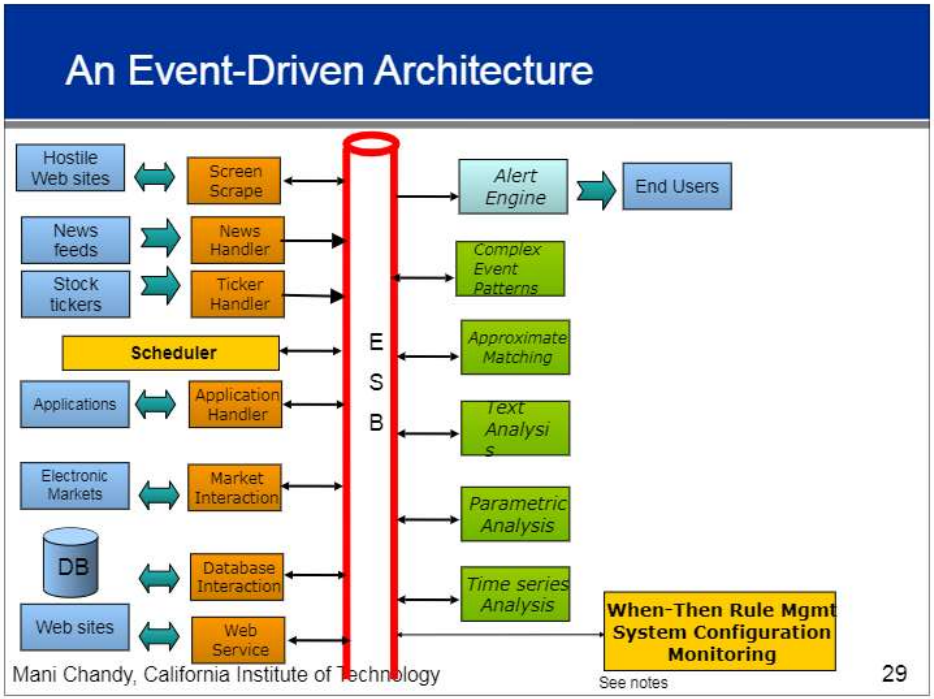


To summarize, SOA and EDA are both necessary. Encapsulating a CICS application falls naturally within SOA. Likewise wrapping client-server apps as Web Services is also naturally SOA. Integration of services within an organization may require transactional semantics, and that fits SOA better than EDA.

EDA is necessary, however, to deal with the "global situation."



The final part of the discussion, today, deals with getting started with an EDA application. The first two parts discussed EDA, its ROI, and its complementarity with respect to other IT tools. The next part deals with examining ROIs for applications and starting the first one.



I'm going to spend some time on this slide for two reasons. (1) To explore architectures for event-driven applications and (2) to emphasize my points that your enterprise already has many of the components of the architecture and that SOA is complementary to EDA.

I'm going to discuss the components of the architecture from left to right.

At the extreme left are the sources of data. These data sources are categorized along two axes. The first axis is the degree to which they provide well-defined interfaces, and the second axis deals with whether data is pushed by the data source or whether data must be pulled from it.

Consider the first axis: how well-defined is the interface? Web Services, databases, and applications within the enterprise (the lower blue components on the left) usually have well-defined interfaces. In some cases, these interfaces are defined using WSDL.

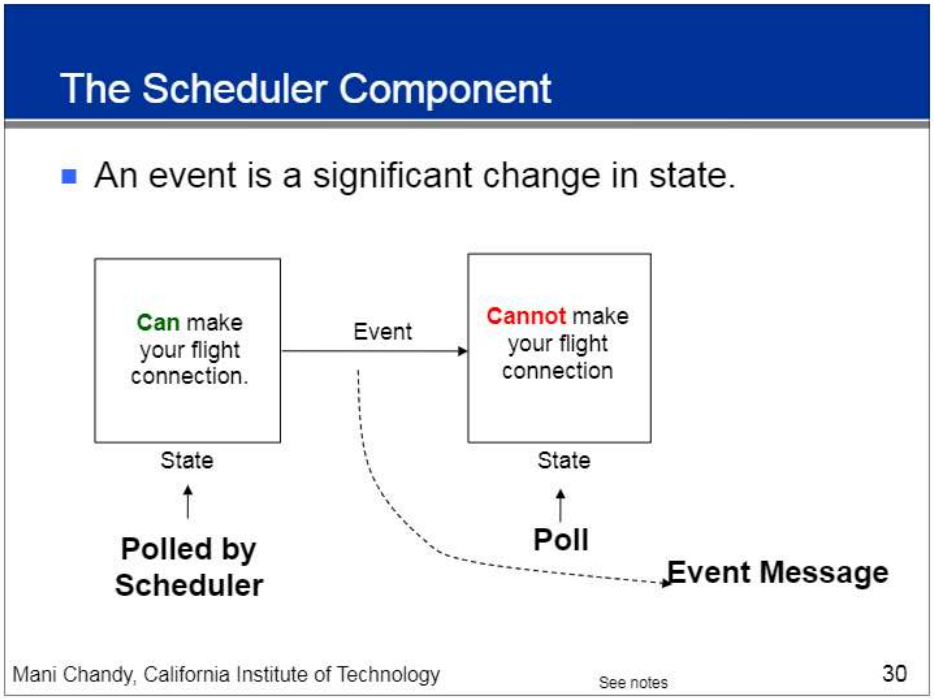
Data sources such as news feeds and stock ticker feeds are also (usually) well defined. The degree of meta-level description may not, however, be that of interfaces within the enterprise. For example, RSS provides a basic degree of meta-level description, and tags in blogs provide valuable contextual information; but, it's not schematized to the same level as calling a Web Service with certain parameters and getting responses of specified types.

Hostile Web sites may have well-defined interfaces but they are under no obligation to inform your enterprise before the interfaces are changed. In some cases, the only way to get information may be through "screen-scraping" with all its difficulties and inaccuracies.

Now let's look at the second axis: news feeds and stock-tickers may push information to the enterprise. Most of the other data sources need to be polled according to some schedule. The scheduler may need to be sophisticated. For instance, certain information – Fed setting interest rates, disclosure of gas inventories – may be made available at specific times such as Wednesdays at 1PM Central Time. These sites may need to be polled frequently just before the time at which the information is usually made available and then not polled at all. The scheduler is important enough to deserve its own explicit component in the architecture chart.

The orange components that interface with the data sources are sensors – they poll or accept data from external sources and convert them to standard schemas.

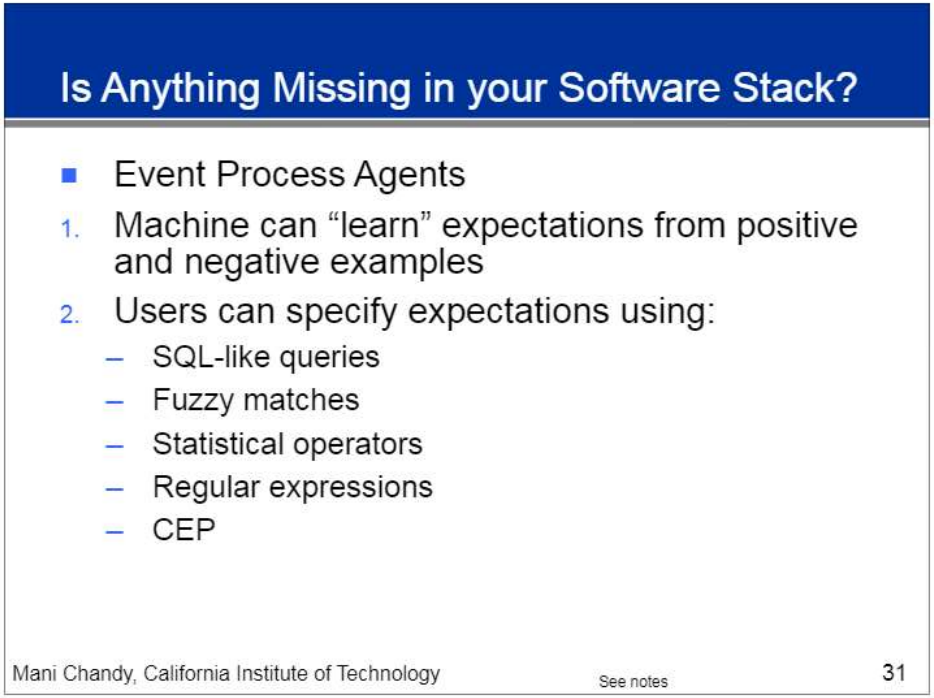
Events from the sensors are passed to the Enterprise Service Bus. The ESB may carry out further translation of the schemas. The next several steps in processing are carried out by Event Processing Agents (EPAs) shown in green. Often an event passed to the ESB by a sensor, say a news source, may be processed by a sequence of EPAs – for instance first by an EPA that analyzes natural language text and then by a Time Series EPA that carries out analyses on the history of results generated by the text analyzer.



An event-message is the message that indicates that an event occurred. An event message has information about the event. The generation of an event message may require polling. Consider the example that we started with: the event that indicates the change from "yes you can make your flight connection" to "No, you can't make your connection." The generation of the event may require polling of the airlines' web sight or invocation of Web Services. The times of the polling are determined by the scheduler.

Schedulers may need to be quite sophisticated. For example, assume that gas inventories are announced every Wednesday at 1PM in Houston. So, it makes no sense to poll the site every minute throughout the week. But, it does make sense to poll the site from 12:58 till the announcement is made.

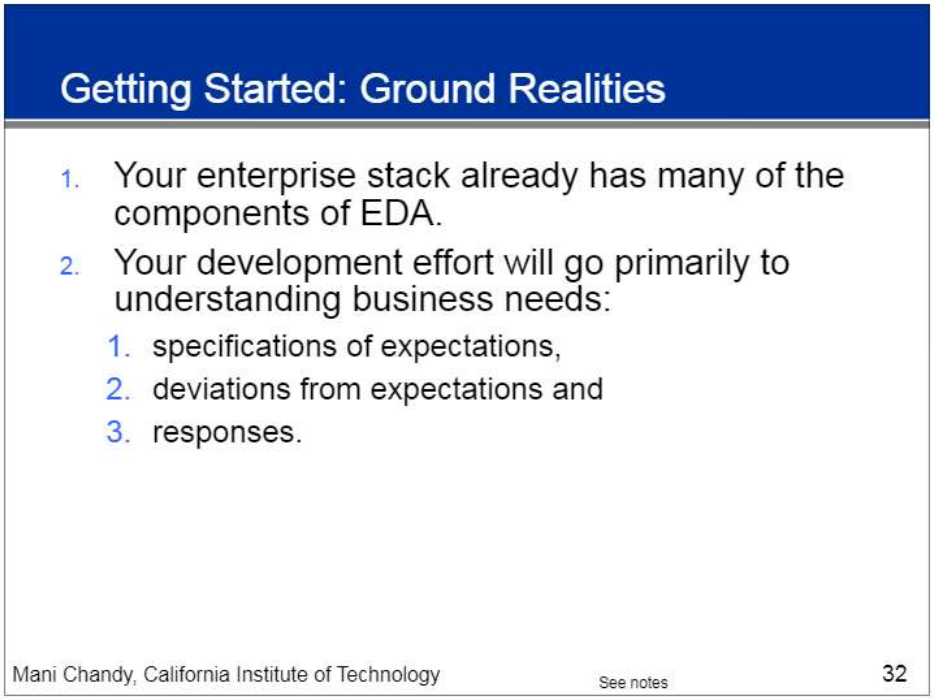
The scheduler is not a difficult piece of code. And indeed a very simple scheduler may be adequate.



Now let's look at another aspect of the incremental cost: the development of components needed for EDA and the effort in integrating these components to provide EDA functionality.

The components that are most likely to be missing are the "cognition" components – the EPAs. You should think through what kinds of EPAs are appropriate for your application. Start with one type, start with one or two rules, show value and then keep adding capability.

Again, you don't need sophisticated tools to get started. It's much more important to understand the business problem, and then create a template for the business user that is useful in the context of the business. The user can change values in the template, but you don't expect the user to write SQL or regular expressions, or specifications in a CEP notation.

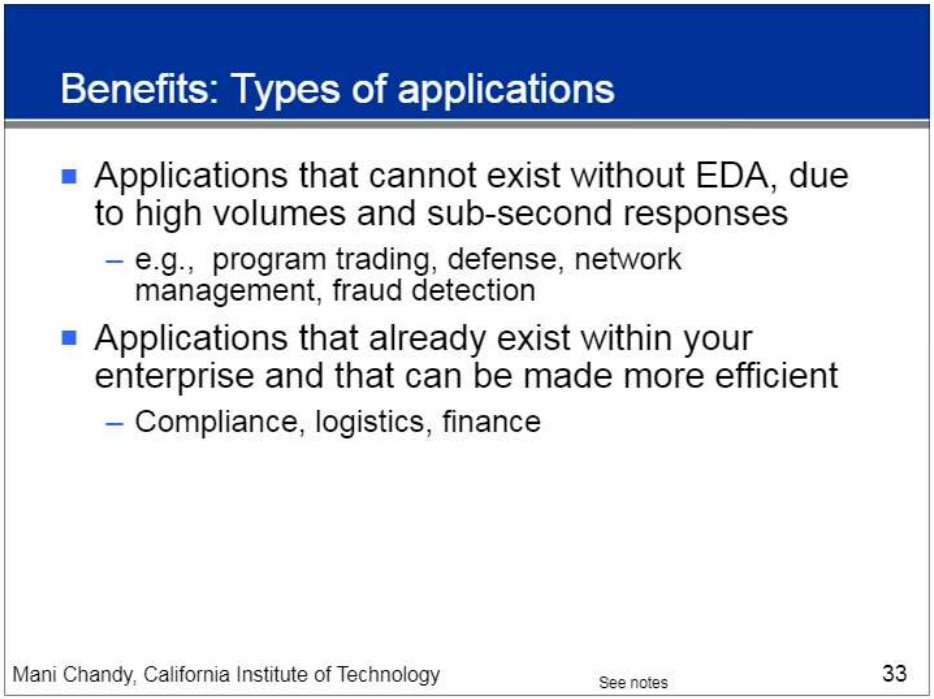


Let's review the realities about getting started.

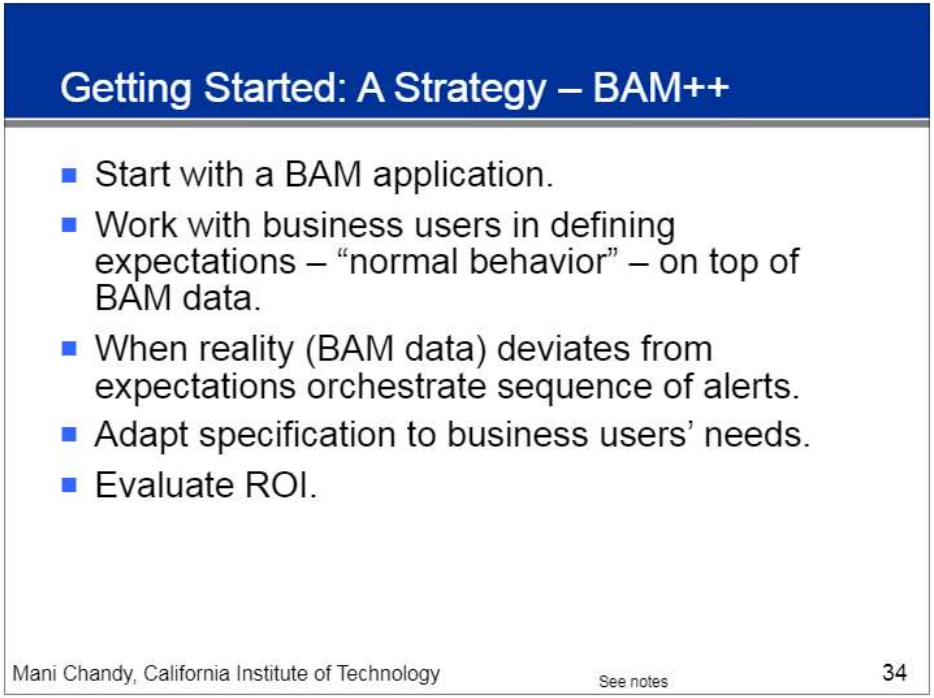
Firstly your enterprise stack already has many of the components you need for a new EDA application.

Secondly your development effort will go primarily towards working with business users in specifying the problem….. How are expectations specified? How are deviations from expectations specified? What sorts of responses are appropriate?

You will develop a UI and a template for the business user where the business user specifies parameters in a set of when-then rules that you've both agreed on. This will take time, of course, but you should consider doing this now with a single application even before you find the ideal tool. Yes, you may have to write some code, but get started with something simple.



The benefits vary depending on the type of applications. The first class of app is the class where the app cannot be built without explicitly considering EDA technology. The second class of app is one that may already exist in your enterprise, but that you can make more efficient.

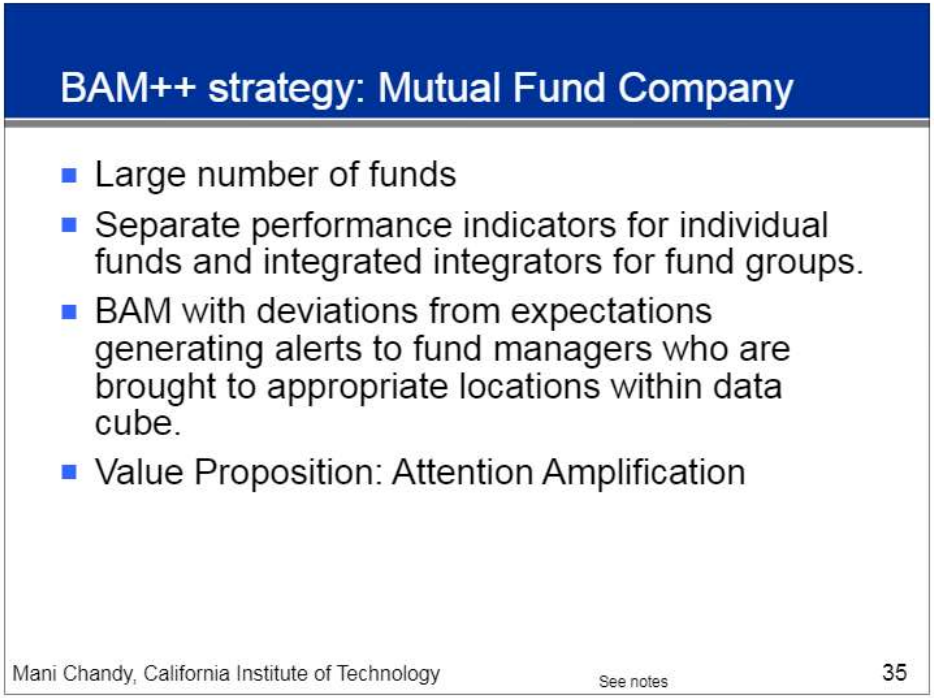


Here's a strategy that will work for many of your enterprises. I call it a BAM++ strategy. The idea is to start with BAM and then add a function: determine if reality deviates from expectation.

The value proposition is that instead of having the business user, say the CFO, continuously monitor the portal, the EDA system will monitor the portal for the user. When something significant happens then (1) alert the appropriate people, and (2) bring links to the appropriate tools into the portal with links to the appropriate data, so that the business user can immediately respond to the threat or opportunity.

The value proposition here is the attention amplifier. This helps the business user amplify his/her attention, and respond rapidly with appropriate tools when a situation arises.

The advantage of the BAM application is that the sensor data --- the data that identifies reality --- is already present; it's sending data to the portal. So, you don't need to connect to new data sources. Secondly, the issue of error is already understood. If the data shows up in the portal, it is sufficiently accurate to be useful. Thirdly, improving BAM seems less radical than developing an event-driven application.



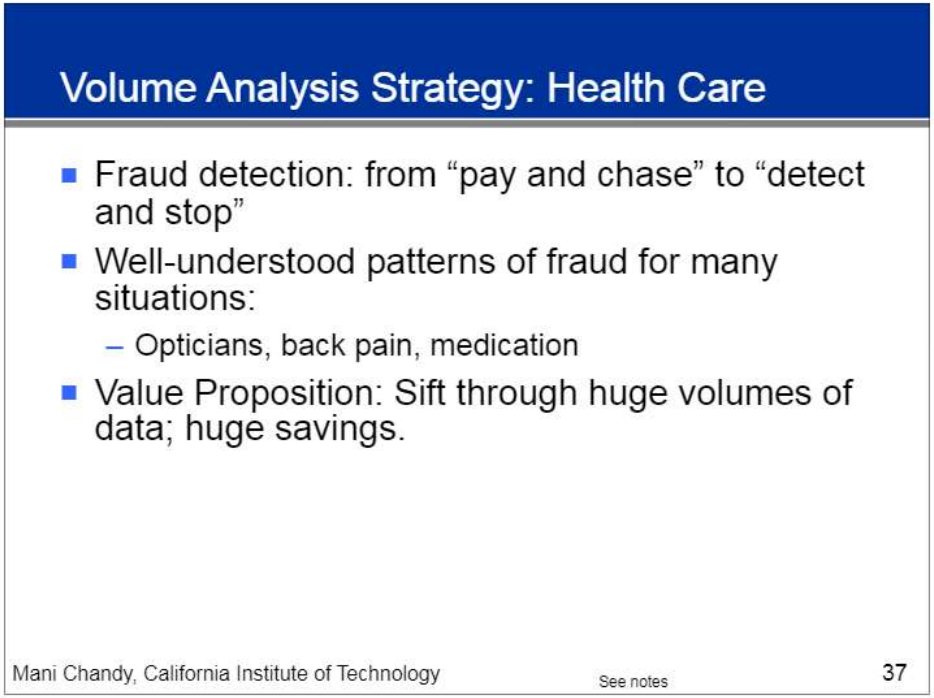
Here's an example of the BAM++ strategy used for a mutual fund company with several funds. Performance indicators for the funds can be complex, and different indicates are aggregated in different ways across groups. The application determined when "actuals" deviated from "plans" for the different indicators, and then brought the business user --- the fund manager or CFO --- to the place in the data cube that contained the data appropriate for the event. The business user was alerted and given the location in the data cube with the date to understand and deal with the deviation of actuals from the plan. (The app also had a mechanism for sending alerts up the management hierarchy, and this can be done by integration with your existing messaging tools.)

The value proposition here is attention amplification. The business user could have monitored the BAM portal, determined when there were situations that required analysis, and found the place in the data cube with the data helpful for the analysis. The tool helps amplify the business user's attention and response.

Energy traders monitor a great deal of information including the power flowing along the main trunks of the grid, the power at generators of their company, data from different markets and other information sources.

Traders made and lost significant amounts of money in relatively short periods. There were important periods during the day when things happened. And there were long stretches when opportunities and threats were not significant. Since traders were monitoring a lot of sources of data they missed out on opportunities and threats.

A company built an application spending a lot of time with traders, and parsing data from several information sources. Much of the data was unstructured and didn't have schemas. Though the development of the application wasn't cheap, the energy trading opportunity got its return on investment in less than 3 months.

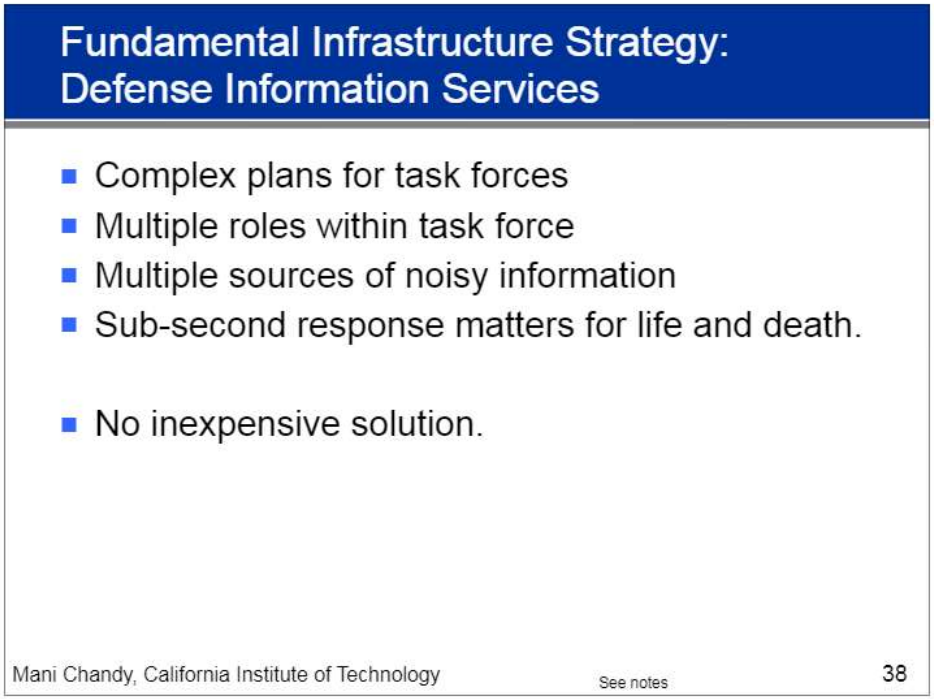


A different strategy is followed by healthcare insurance companies in dealing with potential fraud by healthcare providers or patients. Rather than “pay and chase” i.e., pay first and then attempt to recover funds later, the goal was to detect the fraud quickly and stop further payments.

There are a few patterns of fraud that are relatively common. Of course, there may be fraud going on that the insurance company doesn’t know about or for which the insurance company doesn’t have a pattern. Nevertheless, catching the more common patterns as they occurred can save a great deal of money.

The value proposition here is in sifting through huge volumes of data, and alerting insurance experts on situations that may require further analysis. Ultimately, the value proposition here too is attention amplification ---- instead of searching through thousands of transactions the agents can focus on a much smaller number.

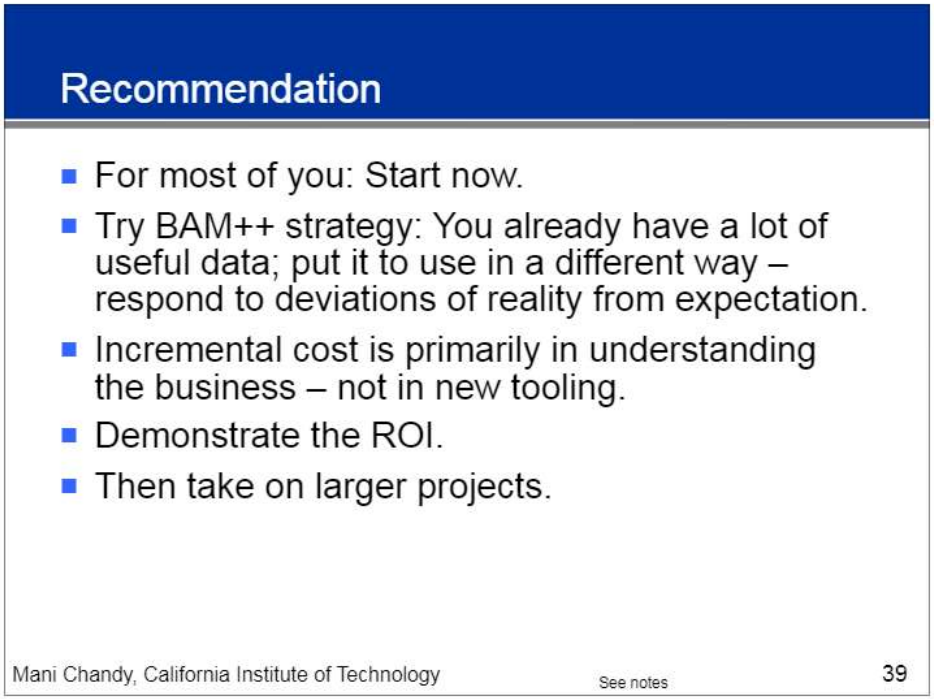
This application doesn’t fall into the BAM++ strategy because the patterns of transactions that indicate fraud don’t usually show up in BAM portals. An advantage of this application is that the data sources are known.



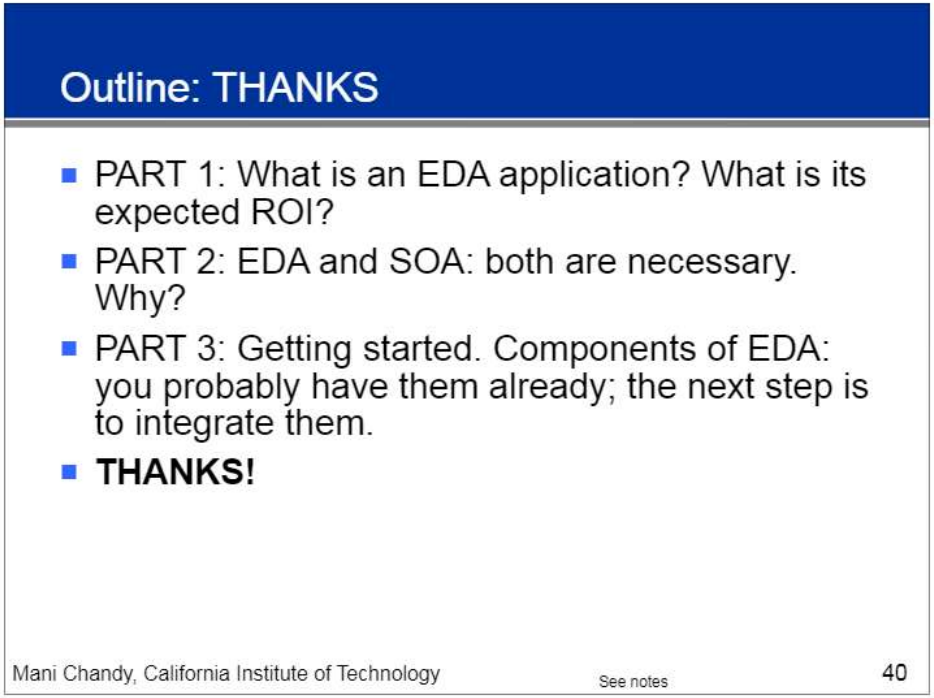
The problem for DISA – defense information services agencies – is fundamentally and profoundly more difficult because they have to deal with complex plans for multiple units of task forces, multiple sources of noisy information, multiple roles, large volumes of information, and sub-second responses are critical.

There is no inexpensive solution. The solution has to be tailored for this specific application. Though COTS tools can be used, the application has unique requirements. So, a great deal of time and effort is required to develop the application.

Most of us don't have applications that are like a DISA application. Most of us have much simpler problems. So, we can start soon with small steps and evaluate the ROI of the application in operation before taking on bigger projects.



My recommendation is to start now and start small with a BAM++ strategy. Evaluate the costs --- particularly your time and the business user's time. Evaluate the benefits, particularly the business user's attention amplification. If the ROI is satisfactory, then consider a bigger project.



We've been through a fast-paced tour of a particular kind of value proposition of EDA ---- responding in a timely fashion when reality deviates from expectation. We've discussed the complementarity of SOA and EDA, and the utility of the existing components in the enterprise stack. We looked at how we can go about evaluating the ROI for EDA. And we looked at first steps that we can take.

Thanks very much for your time. And I look forward to questions.