Evolving Technology and the New Risk and Reward Picture in Information Systems, Part II: The Application Development Landscape

Description

After people and processes, applications are a primary enabler of business capabilities for most organizations of any size. In the early days of commercial computing, most companies had to develop their own applications and run them on their own computers. As things evolved, vendors entered the market in a variety of spaces, including:

- Operating infrastructure (service bureaus),
- Standardized application software packages,
- Hand-built customizations for them,
- Development and operational support services,
- Enabling technologies, such as database software,
- Specialty hardware with integrated application software, such as credit card processing services or ATMs

So, the options changed for HOW companies enabled their business models and operations but defining, implementing, operating and evolving systems was still expensive, time-consuming and risky. In fact, choosing to integrate outsourced software or services complicated things and increased some costs and risks while attenuating others. There are any number of stories about 8- and 9-figure systems failures from the â€˜70s, â€˜80s and â€˜90s. Here is a $44Million governmental failure and here is another one that was estimated to cost $4Billion. To be even-handed, here is a failure involving a private company that went out of business as a result of an error that caused a $440Million loss in just 30 minutes.

On the other hand, disruptive market players, such as Amazon, Google, Airbnb, Uber, Lyft and all of the on-line brokerages, travel sites and millions of others form a robust set of counter-examples, most of them relying on technology that is vastly different than what was available when most of the failures I cited occurred.

Usage Scenarios and Context

Back in the day, customers did not interact with a vendorâ€™s systems. One called a travel agent and the agent would purchase airline tickets and establish hotel reservations. Physical documents would be sent to or be picked up by the customers. Similarly, one called oneâ€™s broker to execute securities purchases or sales. These and other transactions were enabled by proprietary applications, running on infrastructure hard-wired to the offices of the people that used them. Many business models were predicated on barriers to entry erected on exclusive licensing arrangements or the investment in automation required to enter markets. Think of all the big-name retail brokerages (then known as â€˜wire housesâ€™) that no longer exist.

Intermediated process models, such as commercial business sales, operated in *batch.* Orders came in throughout the day, were entered by operators until a cutoff point in the evening and were then submitted, en masse, to be processed and passed along to the next step in the work stream. In such an environment, commercial sales orders could require a week to flow from receipt to fulfillment.

Today, customers perform transactions themselves on general-purpose hardware (anything from a smart phone to a desktop computer) over the internet. While this would have been impossible in the environment that existed much before the end of the â€˜90s, when smart phones didnâ€™t exist and very few had their own computers, it is clearly unthinkable that anyone would implement a customer-facing business model based on manual intermediation today unless it was impossible to avoid it.

Systems: Under the Hood Then

Understanding a little about how systems were built then and how they are now is, I believe, instructive. Architectures and usage patterns can illuminate specific pain points and stumbling blocks that create risks and costs.

Before databases were commercially viable, data was stored in structured files, similar in structure to the rows and columns of a spreadsheet. Often, a file would consist of several different types of records whose context was logically intertwined. For instance, an invoice might be represented as a header record (who the customer was and on what date the sale occurred), a record for each item sold (what it was, how many, at what unit cost and extended cost) and a footer record (what was the total cost of the sale and how much was still outstanding.)

Various processes and steps were enabled by different programs, usually designed to be run in a specified sequence. One might add new sales orders to the file, while another might find and mark line-item records for individual items that had been shipped. A third might sift through and remove completed orders from the file and place them in an archive file.

Data storage was so inflexible and expensive that strange strategies for optimizing its use were employed. It was common for unused fields of various data types to be salted into files and left blank at the time they were first created so that they could be used at a later date for new purposes as the need arose. This wasted expensive storage but pre-empted the need to make massive changes to existing programs in response to new requirements and so represented a calculated trade-off.

If changes to the processes were needed, it was critical that all of the programs that touched any of the files, the structure of the files and the transformation of the data in the files be made in lock step. After a copy of the system was made, changes could be made and tested and then a choreographed sequence of steps could be executed to move current live data to the revised structures and a set of reports generated to â€˜proveâ€™ that nothing had been lost along the way. If this sounds time-consuming, expensive and risky, well, it was.

Databases made transitions easier, largely as a result of what is known as *separation of responsibilities* . A system that employs a relational database as a repository can provide an interface of functions (via *stored procedures*) that a program can call to retrieve data in an expected format without knowing anything about how the data is actually structured behind the scenes. If there is a need to change the structure, it may be possible to rewrite the stored procedures to provide identical responses, obviating

the need to change any of the programs that call them.

So, the evolution of this architecture, circa the â€˜80s or â€˜90s, points to some important design issues that influence the need for and the costs and risks of evolving systems:

- Intermediated business models simplified design requirements by limiting the number of people who needed access to a system.
- Business models were changing fairly slowly, so threats and opportunities generally didnâ€™t force changes faster than systems could be evolved.
- Early architectures were rigid and changing them was expensive and risky.
- Failing to identify and understand requirements comprehensively had potentially ruinous implications so *measure twice cut once* was the order of the day.
- The evolution of technology enabled separation of responsibilities that could help to immunize some components of a system from changes in others.

Systems: Under the Hood Now

As you might imagine, things have changed substantially in the past 20 to 25 years. A lot of what is driving HOW things are done now evolved during this time. New approaches to development, such as Agile, evolved and new architectures have been enabled by advancements in software. What is central to how they impact the economics and feasibility of technology-driven business enablement relates to separation of responsibilities and options for outsourcing, integration and interoperation.

In the next post, we will explore some of these topics.

Date Created
2019/02/25
Author
howardmwiener