

Understanding Tech Options and How They Impact Operations

Written by: Eric Malouff, CEO and Founder, TimeSuite Software

Software selection can be a daunting task, whether it's for accounting, project management, scheduling, estimating, timecard capture, contact management, or another process. Software becomes part of your infrastructure.

Business is about doing better than the competition. Quality, efficiency, customer service, and other processes are impacted by software. Also impacted are cost and organizational controls and, in turn, the risk of losing money on a project or falling short of optimal profit on a project.

This article is meant to provide perspective when choosing software options. Do you select software specific to the various tasks? Do you select a modular system that facilitates each task through subsystems that are integrated? Or do you select a relational system where data and application controls are centralized?

Multiple Software Applications

Different software applications are selected to maintain data and tasks for different facets of your business. Developing software is like a design-build construction project. The vendors have developed their systems independently over periods of time. Mixing and matching software components from different software suppliers can have varied results. Integration between different applications is never seamless, and data generally has some level of incongruence.



Modular Software Applications

Modules are subsystems within the same software application. Integration is how data is shared between the subsystems. Transactional data is posted between modules.

If you take a database class, in the first week you learn about normalizing your data structure (eliminating duplicate data). If you have an accounts payable (AP) invoice that is entered into an AP module and posted to both job cost and general ledger modules, then this simple rule is broken. There is a difference between accounting and other processes such as estimating, project management, schedule, timecard tracking, etc. These other subsystems generally interact with one or more of the accounting modules. Thus you still have the limitations presented by modular data.

A major component to reducing code and improving quality and efficiency involves centralizing controls and components. For example, the code to draw a grid should be in a system one time. Every time a grid is used, this exact code should be used. With separate pieces of software that are integrated, it is less likely that the interface uses centralized controls and components.

Posting accounting transactional data between modules is redundant and unnecessary. This process also makes it difficult to provide automated audit trails and to report where data is maintained under many modules. Modular transaction data that is generally maintained in the subsidiary, job cost, and the general ledger also consumes a lot more space than a relational system.

As a modular system matures, the complexity becomes harder and harder to contain. A beast of a code base makes maintenance and enhancements difficult. A well-coordinated army of developers is required to pile additional code on top of code. Over time, these systems become massive. They are characteristically rigid and less efficient to maintain.

Relational Software Applications

A relational system doesn't have modules. Instead, a relational system will have facilities that use centralized data. An AP invoice will be stored in a centralized transaction structure. Each facility will use the same centralized address books, item lists, transaction structure, task lists, etc.

Relational systems have other advantages. Dynamic setup becomes possible. If you've ever implemented a system where implementation decisions are irreversible, then you will understand this concept. Relational systems have a dynamic architecture, and thus setup decisions can be changed at any time. Removing the coordination between modules and subsystems simplifies the setup. Thus, a system can be set up initially to reflect most use-case scenarios, and can be adjusted during implementation.

Centralizing controls and components simplifies training because each facility in the system utilizes the exact same code and functions in the same way.

Normalized (nonredundant) data structures provide for ease-of-use features such as automated audit trails, automated over/under billings, automated accrued wages, and automated retroactive allocation of over/under allocated indirect cost.

Also facilitated is reporting that isn't limited because data is centralized, such as percentage-of-completion financials (including a full summary of contacts that report wages as of the work date, not the check date).

Relational system architectures provide for a much larger feature set.


As the system matures, the complexity is contained. A smaller development team is preferred where production is high and team coordination is facilitated. So, relational systems evolve faster.

Gained efficiencies are achieved when a process is refined. Relational systems generally provide for pliable dynamic processes that conform to processes that are chosen by your company – rather than processes that are dictated and include lengthy checklists.

Conclusion

Relational architectures are advanced. Why doesn't every system have an advanced architecture? Because modular, full-featured systems have been developed over the last 40 or 50 years, and it would take rewriting the software's core to change the architecture.

Relational architectures require software infrastructure, and this process takes time. But development on the other side of that infrastructure is simplified, and a software development team can be significantly more productive.

You have many choices with the software available, and the types of systems to incorporate into your infrastructure. Your competitive advantage/disadvantage and, ultimately, your reputation and profitability are impacted by the infrastructure you put in place and maintain. A relational system provides efficiency, advanced feature sets, ease of use, and the ability to evolve. Relational software can be significantly less expensive than modular software when it comes to implementation, maintenance and upfront/ongoing vendor costs. 



About the Author

Eric Malouff is a certified public accountant (CPA) who grew up in the construction industry. His father was a dirt and asphalt contractor. Growing up, he worked as a laborer, foreman, project manager, and estimator. In college, he worked part time for two years as a software developer while finishing up accounting and computer information systems degrees.

Malouff started Malouff & Company, P.C. CPAs in 1990 and practiced in public accounting specializing within the construction industry for 18 years. He is the CEO and founder of TimeSuite Software (founded in 1994). Visit timesuite.com.

About the Article

Republished from [Construction Business Owner](#). Construction Business Owner (CBO) is the leading business magazine for contractors and is designed to help owners of construction firms run successful businesses. Founded in 2004, CBO provides real-world business management education and knowledge that is of real value to the owners of construction companies.

Any views and opinions expressed in this article may or may not reflect the views and opinions of the Construction Management Association of America (CMAA). By publishing this piece, CMAA is not expressing endorsement of the individual, the article, or their association, organization, or company.