It's a Simple Matter of Programming

Build vs. Buy Considerations for File Workflow and Job Scheduling

Reading Time: 5 minutes

Your developer has just told you "*it's just a simple matter of programming*." Really?

According to Wikipedia, a

> "Small Matter of Programming (SMOP) or Simple Matter of Programming is a phrase used to ironically indicate that a suggested feature or design change would in fact require a great deal of effort; it often implies that the person proposing the feature underestimates its cost. Such underestimated costs are common during cost estimation, particularly near the beginning of a project."

Conversely, developers will sometimes propose an unrealistically small amount of effort to accomplish something in the hope they get assigned to something they consider "interesting." Unfortunately, often what interests the developer may not necessarily be something key to the business. Some managers can gauge their development experience by simply counting the number of times they've been sold the need for a new build process, or a new development methodology, or some software library, or some new piece of technology – each claiming to resolve deficiencies in their current development practices while in actuality the development manager is being sold a developer's "interest."

Filtering fact from fiction

You're a development manager, IT operations manager, system architect, or development lead. In arrives some requirement for workflow or task management, or some scheduling need. You manage some core technology component of your enterprise. It may for example be report generation, or media review and cataloging, or claims processing, or payments fraud detection.

You surface the requirement for team discussion. Someone raises their hand. "Hey. That's easy. Give me a few days and I'll be done. And hey. It'll be free." He proposes integrating some open source of their own choosing, writing a little code and voila! Problem solved. But really?

Unbeknownst to you, on the forums he has posted the following (very real) posting:

> "In my new Java project, we have to develop a feature of Job scheduling, for example I have to create a program in java to execute certain task in a specific time or after a regular intervals of time. Can you please suggest me how it can be done? But first of all please explain me what is job scheduling in Java and how one can use this feature in their application."

Besides hearing about cron expressions and job scheduling, he gets feedback and insights regarding clustering, fail-over, parallel processing, state machines, job pinning, concurrency management, monitoring, API design, database schemas, triggers and events, business calendars, and a litany of other seemingly esoteric commentary. "But all I need is a scheduler?" he sighs.

But he pulls off his job scheduling effort - or so it appears. He does indeed integrate some open source, writes a few classes, makes a few (or more often than not, many) simplifying assumptions and voila! Problem solved. For now. Occasionally things go bump in the night, but he's there to make it all better.

Time Marches On

It's now some time later. That developer has left for bigger and better things. That system you addressed that scheduling requirement for - well it's now mission critical. And the scheduler- well it's going bump in the night - a lot! And now you need some new feature added to that scheduling code. You bring it up to your team - now a new team of course - everyone has gone on to bigger and better things. Some new hire raises their hand. *It's a simple matter of programming.* Give me a few weeks and I'll be done.

And again - voila! Well, not so much. That bit of code is now replaced with a framework and an API and a ton of code. More time is spent documenting the API than was spent in the original development effort. And nothing in the new code is working as expected or being delivered on schedule.

Ask Yourself

Ask yourself. Is this software key to your business? Is it what customers' actually pay your company to do for them? Say you work for an Analytics company. Your company's customers' pay for new insights from your big data software. They are not paying you for your workflow, job scheduling, or file transfers.

Go through your 'build vs. buy' checklist. If you don't have one – consider the following buy considerations:

- Will a purchased solution significantly reduce the amount of coding required to incorporate advanced functionality into an offering?

- Will a purchased solution help the solution stay current with new functionality requirements?
- Will a purchased solution support existing interoperability standards to ease integration efforts?
- Will a purchased solution come with extensive documentation and experienced technical resources that help reduce the time to deploy and roll-out your enhanced solution?
- Will a purchased solution ease customization of the look-and-feel of the solution?
- Will a purchased solution make it easy to operate and administer the solution?
- Will a purchased solution provide a seamless, single source of technology components to meet your requirements?
- Will a purchased solution reduce the ongoing cost of support and maintenance of the solution and cost of future functionality?
- Will a purchased solution scale to support significant increases in capacity?
- Is the purchased solution an area of core competence for the provider? Is the solution backed by ongoing R&D investment?

And then of course - Will the purchased solution cost less over time than an in-house developed solution?

In evaluating the last question, diligently challenge your own and your developer's development assumptions. Is there available a plan, task list, or to-do list in place to do the following fundamental tasks? (The below list is not exhaustive but supplied as a starting point for challenging the assumption "*it's simply a matter of programming*."

- Define the problem
- Identify entities and relationships
- Map out the business process, including inputs, outputs, all steps, and all participants
- Design, develop, and test system components
- Design, develop, and test algorithms/configurations
- Design, develop, and test input screens
- Design, develop, and test output screens and reports
- Design and test integration points
- Estimate disk storage requirements
- Design and test deployment options for hardware, bandwidth and other networking considerations
- Build documentation, help, and training materials
- Build and execute validation and testing scripts

Then ask yourself as the manager, do I:

- Have resources to build an application and all its reports, configuration options, monitoring, error handling and recovery?
- Have resources to maintain and support a custom-built application including all necessary changes that surface in the future?
- Have the time to assess all requirements and business processes to ensure that effective processes are being mapped into the new automated solution?
- Have time and resources to validate and continue to re-validate this custom-built application?
- Have expertise in evaluating the best approach to setting up the IT infrastructure for the new application solution across the organization?
- Have expertise and resources in setting up an internal helpdesk to support the custom-built application?
- Need flexibility to model workflows for both corporate and our local sites in one system based on company Best Practices?
- Have the resources for developing and implementing continuous improvements in the application functionality and technology?
- Want the product depth, and functional and external integration without developing it?

Advantages vs. Disadvantages of a Purchased Solution

Build vs. buy is a set of trade-offs beyond just that of cost. The costs vs. benefits are straightforward:

Disadvantages of Buying:

- Vendor retains rights to the code
- Product functionality determined by vendor
- Reliance on vendor's technical support to resolve issues
- Solution may not fit needed requirements completely or sufficiently

Advantages of Buying:

- Ready-made solution
- Many man-years of research and development
- Expert support and training
- Flexibility/adaptability
- Typically, function is more complete and robust than that of an in-house solution

Making the Decision

Enterprises often choose to build their own software because of the belief that it will be faster, cheaper, and better tailored to their specific needs. Yet the time, effort, and expense of building and maintaining such software can often far exceed cost and schedule estimates and miss the mark on expected benefits. Organizations, in their effort to become lean, look to invest their development resources into applications that are core to their business. Developing infrastructure that is readily addressed by off-the-shelf solutions is generally not an efficient of effective use of your intellectual or financial capital.

About Flux

Flux assists enterprises in provisioning, onboarding, scheduling, tracking, and reporting an enterprise's file orchestration processes. These orchestrations vary from simple file transfers to highly complex workflows involving extensive processing, many routes, varied alerts, and complex decisioning. Flux facilitates finance, healthcare, and software providers in effectively orchestrating files to create new revenue opportunities and facilitate expense reductions. First released in 2000, Flux has grown into a file orchestration platform that hundreds of enterprises rely on daily for their mission critical systems.

Contact Flux

sales@flux.ly
flux.ly