Seneschal Incorporated

Newsletter

July 2010 From Our President

Business and Technical Consulting You Can Trust

Choosing a Language

We've written before about the importance of choosing the right platform and tools for an IT project. This month, we'll take a closer look at some of the factors to consider when choosing the programming language to be used in writing source code for your software.

Languages are Not All the Same

If you are multilingual in spoken and written languages, which of these languages would you choose to write a love poem: French, German, Italian or Dutch? Why?

Similarly, when you think about what you want your software to do, some languages will be obvious misfits and others will be appealing possibilities. Then it's just a matter of deciding among the ones that are appealing.

Make a list of the answers to these questions:

- Will the software be doing predictable sequences of work, or unpredictable sequences?
- Must the software squeeze as much performance as it can out of the computer?
- Are specialized interfaces needed between the software and hardware devices (such as device drivers), or between the software and operating system (special hooks to operating system internals)?
- Is the software a small, medium, large or very large application? Is it complex?
- What hardware platforms must the software be able to run on?
- What operating systems must the software be able to run on?
- Is availability of expertise an issue?

There may be additional factors you want to include, but these are enough for most projects. Now it's time to see which languages are the best fit for your needs.

Strengths and Weaknesses

Make a column for each language under consideration. For each language, go down the list of factors and make a checkmark if the language is well suited for it. If you don't personally know the

Let's Keep in Touch!

Newsletter Subscription
Subscribe to our free email newsletter

Send An Inquiry or Comment
Ask a question, send us feedback, tell us what your business needs

Information to Help You Prosper

Information and Referrals to Bargains
Information products for sale, and also free reports and referrals to special discounts we negotiated for you with our favourite vendors

Get Paid! As Much As You Deserve, And On Time
Affordable updated ebook – how to get your customers to pay on time and in full, without needing collection agents or lawsuits (USA edition)

Get Paid! Paperback
Amazon.com listing for the above as a paperback

2003 Book
Amazon.com listing for the original book Make Sure You Get Paid (And Other Business Basics)

Contact Us

www.seneschal.biz
service@seneschal.biz

***3608 Audubon Place
Houston, TX 77006-4416
USA***

***+1 713 893 1254 phone
+1 832 201 7720 fax***

President: Bonnie D. Huval

strengths and weaknesses of each language, get help. It's important to fill in your spreadsheet based on factual information, not fads or personal feelings.

Here are some general examples of language differences that should jump out at you from the completed spreadsheet.

- Languages that must be compiled and linked to generate an executable program, such as C, C++ or COBOL, make much faster-running software than interpretive languages such as the original BASIC or Java, but must be recompiled and relinked for every target operating system.
- Interpretive languages execute relatively slowly, but can be taken from operating system to operating system with little or no effort.
- Procedural languages produce relatively fast-running, compact executable programs with low overhead for predictable sequences of work. They are cumbersome for unpredictable sequences of work such as responding to human users in a windowed user interface.
- Object oriented programs excel at handling unpredictable sequences of work, but require relatively high overhead, so they tend not to be as fast-running as procedural programs.
- High level languages such as COBOL, FORTRAN are more easily readable by people, are easier to maintain and debug, and are designed to reduce the likelihood of such errors as memory leaks. They are not as flexible as low level languages such as C, so they cannot easily manipulate bits within a byte or directly command hardware.
- Low level languages such as C (procedural) or C++ (object oriented) can work almost as "close to the machine" as a computer's native assembly language, so you can do nearly anything with them. It is easy for programmers to make such mistakes as memory leaks in these languages, and they are difficult to maintain because they are so cryptic.

Final Decision

When you finish filling in your spreadsheet, some languages should have a lot more checkmarks than the others. Those are your finalists. If you are lucky, the finalists might be similarly capable for your project. In that case, you can use the language among your finalists that is most comfortable for you.

Typical Phases in a Development Project

Each phase in an IT development project generates documentation. Sets of documents in an IT project are not just intended to record information. The act of writing each set is supposed to help the team move its thinking along a path that leads to successful completion.

Typical projects go through phases more or less like this, each generating one or more documents:

- Functional Requirements
- Functional Specifications
- High Level Design (architecture, heavily functional with some technical detail, a 50,000 foot view)
- Low Level Design (detailed design)

Very large projects sometimes add intermediate documentation sets, notably Technical Specifications just before High Level Design. Small projects sometimes compress a couple of sets into one set. We do a lot of work for manufacturers. They often do just one set of design documents, for example.

On a parallel track, the test team makes its own set of documents. Testers begin with the Requirements documentation and design tests to make sure each requirement is met—and to make sure every conceivable unexpected situation is handled gracefully by the software. Documents generated by the test team are typically:

- Test Plan (high level design of the testing)
- Test Design (test cases, test tool creation and setup, etc)
- Test Report (outcome of the testing)

We are simplifying a little here, talking mainly about Integration Testing which verifies the whole beast. In practice, developers must test their pieces as best they can before checking in their software, which is called Unit Testing.

The project manager makes and maintains a matrix that ties all of this together. Every Requirement is linked to one or more Specifications, which in turn is linked with elements of the Design and with Tests.

Each requirement should be assigned a number. The trace matrix links each requirement to documentation references about it in every phase of the development project, all the way through to the test results. There should be notations all over the requirements traceability matrix before the software is released.