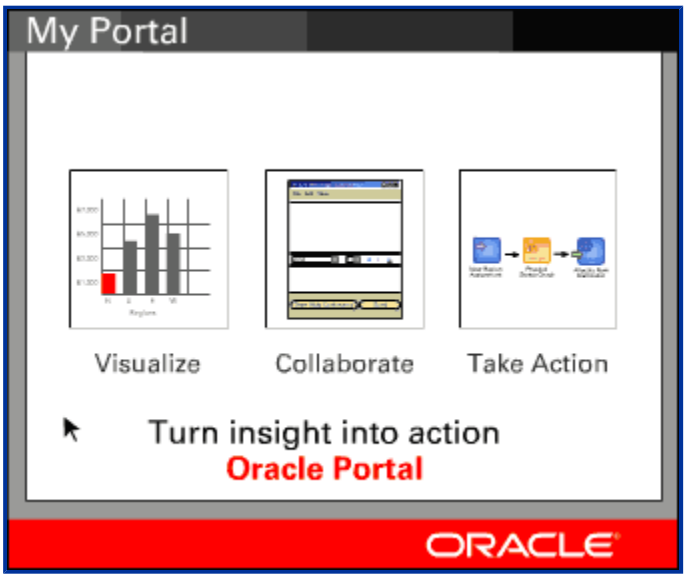
> Return to story

QuickStudy: System Development Life Cycle

Quickstudy by Russell Kay

MAY 14, 2002 (COMPUTERWORLD) - Once upon a time, software development consisted of a programmer writing code to solve a problem or automate a procedure. Nowadays, systems are so big and complex that teams of architects, analysts, programmers, testers and users must work together to create the millions of lines of custom-written code that drive our enterprises.



To manage this, a number of system development life cycle (SDLC) models have been created: waterfall, fountain, spiral, build and fix, rapid prototyping, incremental, and synchronize and stabilize.

The oldest of these, and the best known, is the waterfall: a sequence of stages in which the output of each stage becomes the input for the next. These stages can be characterized and divided up in different ways, including the following:

- **Project planning, feasibility study:** Establishes a high-level view of the intended project and determines its goals.
- **Systems analysis, requirements definition:** Refines project goals into defined functions and operation of the intended application. Analyzes end-

user information needs.

- **Systems design:** Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.

- **Implementation:** The real code is written here.

- **Integration and testing:** Brings all the pieces together into a special testing environment, then checks for errors, bugs and interoperability.

- **Acceptance, installation, deployment:** The final stage of initial development, where the software is put into production and runs actual business.

- **Maintenance:** What happens during the rest of the software's life: changes, correction, additions, moves to a different computing platform and more. This, the least glamorous and perhaps most important step of all, goes on seemingly forever.

But It Doesn't Work!

The waterfall model is well understood, but it's not as useful as it once was. In a 1991 Information Center Quarterly article, Larry Runge says that SDLC "works very well when we are automating the activities of clerks and accountants. It doesn't work nearly as well, if at all, when building systems for knowledge workers -- people at help desks, experts trying to solve problems, or executives trying to lead their company into the Fortune 100."

Another problem is that the waterfall model assumes that the only role for users is in specifying requirements, and that all requirements can be specified in advance. Unfortunately, requirements grow and change throughout the process and beyond, calling for considerable feedback and iterative consultation. Thus many other SDLC models have been developed.

The fountain model recognizes that although some activities can't start before others -- such as you need a design before you can start coding -- there's a considerable overlap of activities throughout the development cycle.

The spiral model emphasizes the need to go back and reiterate earlier stages a number of times as the project progresses. It's actually a series of short waterfall cycles, each producing an early prototype representing a part of the entire project. This approach helps demonstrate a proof of concept early in the cycle, and it more accurately reflects the disorderly, even chaotic evolution of technology.

Build and fix is the crudest of the methods. Write some code, then keep modifying it until the customer is happy. Without planning, this is very open-ended and can by risky.

In the rapid prototyping (sometimes called rapid application development) model, initial emphasis is on creating a prototype that looks and acts like the desired product in order to test its usefulness. The prototype is an essential part of the requirements determination phase, and may be created using tools different from those used for the final product. Once the prototype is approved, it is discarded and the "real" software is written.

The incremental model divides the product into builds, where sections of the project are created and tested separately. This approach will likely find errors in user requirements quickly, since user feedback is solicited for each stage and because code is tested sooner after it's written.

Big Time, Real Time

The synchronize and stabilize method combines the advantages of the spiral model with technology for overseeing and managing source code. This method allows many teams to work efficiently in parallel. This approach was defined by David Yoffie of Harvard University and Michael Cusumano of MIT. They studied how Microsoft Corp. developed Internet Explorer and Netscape Communications Corp. developed Communicator, finding common threads in the ways the two companies worked. For example, both companies did a nightly compilation (called a build) of the entire project, bringing together all the current components. They established release dates and expended considerable effort to stabilize the code before it was released. The companies did an alpha release for internal testing; one or more beta releases (usually feature-complete) for wider testing outside the company, and finally a release candidate leading to a gold master, which was released to manufacturing. At some point before each release, specifications would be frozen and the remaining time spent on fixing bugs.

Both Microsoft and Netscape managed millions of lines of code as specifications changed and evolved over time. Design reviews and strategy sessions were frequent, and everything was documented. Both companies built contingency time into their schedules, and when release deadlines got close, both chose to scale back product features rather than let milestone dates slip.