8 December 1993
Wednesday
The Grain #1

>> is the key to reusable software. The claim is that a separate group
>> is in the best position to do this, one that can focus on the
>> architecture, and not on particular instances of it.

>In fact, it's hard to arrive at a viable architecture *without*
>focusing on particular instanceS (more than one) of it. The classic
>problem with such partitioning of effort is that the feedback loop
>breaks.

This is my problem with the Reuse Group pattern as well. It may make it harder for insight from actually trying to reuse something to find its way back into the system.
Consider the following alternative:
(This is something of a response/pattern hybrid.)

❖ ❖ ❖

DEPLOY PEOPLE ALONG THE GRAIN OF THE DOMAIN *
(ONE PERSON/MANY HATS)

Valuable architectural insight tends to emerge late in the lifecycle, as a result of having addressed requirements from concrete, successive problems drawn from a given domain. It is then that a system can be refactored to consolidate design insight and polish reusable artifacts. (More can be said about this, but that's another tale.)

Such insight can best be harvested if people are deployed along the grain of the domain, and a given individual has responsibility for a well-defined part of it. Organizational categories like Analyst, Designer, Coder, Maintainer, and Reuse Expert can cut across the grain, and greatly increase organizational communication overhead and inertia. However, when responsibility for all these functions for a given part of the system is vested in a single person, the communication overhead for redesign with that part of the system can be largely intracranial. This is a one person/many hats strategy. A single individual can cope far more quickly with on-going bi-directional tensions between top-down elegance and bottom up detail than can a functionally partitioned organization. Such a person can develop a more comprehensive sense of the possibilities that the design space allows, and exploit these to develop more genuinely durable artifacts.

A reusable API or object-oriented framework is, in many respects, a domain-specific language. Given this, Wirth's classic admonition that language design is better done by a single guiding intelligence, rather than by a committee, applies.

Small teams deployed along the grain should be able to glean similar benefits. Team members would be responsible for distinct parts of their team's domain. Metafunctions like pure management and documentation might be factored and assigned to additional individuals. Interpersonal communication would primarily be concerned with interface negotiation, and not become mired with approving changes to internals.

The key here is committing talented designers to a part of the system, and keeping them there until late in the lifecycle, when hindsight from addressing a range of design issues is available.

There is some commonality between this and Alexander's Architect/Builder notions as well. This sort of personnel deployment strategy is a defacto favorite in academic environments, and some small organizations, both of which often exhibit marked productivity advantages over traditional industrial organizations. If an organization really wants to get truly reusable software, it will have to be willing to budget time and talent in such a way as to exploit the insights that lead to it at the point in the lifecycle where they become available. But reuse isn't something that can itself easily be factored into its own department. The people who build and maintain something in the first place have the best, most intimate knowledge of how to generalize it.

I've left myself a certain amount of Alexandrian wiggle room with regard to the question of how one knows where the grain is, especially at the onset of a project. Often its something people
settle into.

❖ ❖ ❖

10 December 1993
Thursday
Design #1

From: Daniel LaLiberte <liberte@cs.uiuc.edu>
Subject: Re: Reuse Team

I liked your article and it resonates with my own understanding. Here's a couple related thoughts.

Since there is no clean distinction between design and implementation, we should not be surprised when there are problems with assigning corresponding job roles.

What about large problems that have a very large grain size such that no single person can fill one? We also have people who can't fill even a small grain size (maybe they are beginners), but still want to do useful work in cooperation with others. Perhaps the grain can be sliced finer, or perhaps we just have to learn how to put our heads together.

The pioneers in a new field can often go further, deeper, faster than followers not because the leaders are brilliant (which they may be) but because they realize they are creating the vision and all the pieces of the vision are wherever they put them. To be half as effective, the followers must somehow get into the pioneers heads to see with their vision.

Seeing how a whole thing fits together lets one understand more about it than the sum of parts. It is also more exciting.

A large whole thing that is too large for one person can be divide up in lots of ways, not just in discrete pieces but also across many overlapping dimensions. So design and implementation domains make some sense, and it is valuable to have people who are good at each. But we should also have individuals involved with design and implementation of one thing.

Pattern: Break away from strict hierarchies.

Organizations tend to fall into the traditional pattern of one leader at the top with many underlines, repeated recursively. Rarely do we find a person with more than one immediate boss, at least formally declared as such. But tasks that are multi-dimensional (as most are) might benefit from multiple leaders, one for each dimension.

Pattern: Collapse hierarchies when possible.

When a problem or subproblem is small enough to fit in one head, there is no need to break it apart into multiple individuals that must then cooperate usually under the leadership of a manager. Instead, one individual can be own boss regarding at least this one the task.

Combining these two patterns, an individual can be responsible for a domain that may intersect with many other domains, each attended to by other individuals. Imagine a tapestry of overlapping domains.

I guess this might be worth posting, cleaned up a bit.

dan

From: "Byard, Cory" <cbyard@wpdsmtp.daytonoh.NCR.COM>
Subject: Re[2]: Reuse Team

I agree with Brian's comments as applied to small systems development, but do not think that the model of "whole grained" developers scales well to large projects. Two problems arise:
1) for a large project (many function points, much complexity, large application domain), there is far too much expertise and work required for a small team, no less a single person. and
2) in most commercial organizations (all that I deal with), most developers do not have the experience, conceptual acuity, and inventiveness to deal with all the complexity.

However, I think there is (and must be) a middle ground. We must find a way to take advantage of our experience with total projects, while maximizing the capability of real developers. I think the architect role does this. An architect is a person by nature of assignment, experience, and ability, who has the whole grain responsibility for several applications in a system. The architect does not develop fine design details or write and debug the programs. It is this person (or a small team of architects for large systems) that can do the intercranial work to re-factor the system, identify reuse opportunities and communicate the conceptual framework back to the development team(s).

Cory Byard, NCR/AT&T CASM
cory.byard@daytonoh.ncr.com.

From: Doug Lea <dl@g.oswego.edu>
Subject: Re[2]: Reuse Team

You might be interested in reading what Christopher Alexander has written about preserving integrated Architect-as-Builder roles even in large projects. He thinks it essential to do so, leading him to consider alternatives to common contractual and organizational arrangements (as they occur in building architecture; not so different than in software). For a very brief account, see his `Perspectives:Manifesto 1991' in Progressive Architecture (magazine), July 1991.

-Doug

From: Doug Lea <dl@g.oswego.edu>
Subject: Re[2]: Reuse Team

To follow-up on my own posting...

Having teams of architect/builders is NOT at all in conflict with creating reuse teams, tool teams, or advanced-development teams. In fact such divisions are among the few that DO preserve most of the advantages of the concept. The clients of a reuse team are other developers. The same considerations of requirements sensitivity, client involvement, etc. apply. True, the resulting software is necessarily one step removed from ultimate user requirements, but this is balanced by the greater degree of abstraction that such teams can leverage in developing multiuse components.

-Doug