Evolutionary Delivery

TomGilb's proposal from the early eighties for development of usable software sub-systems in very small steps, with user requirements being continually re-assessed after each delivery.

I had the pleasure to spend 2 days with Tom in early 2003 in London, and have been researching his and others' work for an IEEE Computer article on the history of iterative development. His proposal for evolutionary, iterative development comes from the 1970s, not the 80s. His 1976 "Software Metrics" book discusses, as do articles he wrote in trade press in 1978. Also, I came to see his focus is SystemsEngineering, not Software Engineering. He hasn't been involved in software technologies or programming for some time, is not into the details of object technologies, object design, modern testing methods or tools, etc. This is not a criticism; his focus is broader systems thinking.

Also, his method is not *about development of usable software sub-systems in small steps. Rather, it is about doing, in the next week, the highest value/cost ratio thing of measurable value to the customer. Several of his examples to me emphasized avoiding software solutions and finding quick operational change solutions. i.e., much more SystemsEngineering focussed.*

I can also say, after listening to him for several days, that a key message in his system is that someone must *define project objectives, with quantitative measures of relevance to management/customer, and each short iteration must measure the results, and communicate them to the customer. It is hard-nosed and value-driven; no wishful thinking that we are making progress.* -- CraigLarman

Gilb's ideas gradually became extremely influential through better publicized but often less well thought through movements such as RapidApplicationDevelopment (RAD) from JamesMartin and the DynamicSystemsDevelopmentMethod (DSDM) consortium in the UK.

ExtremeProgramming is the first soup-to-nuts proposal for the whole programming process that goes well beyond Gilb in its team development practices but it remains highly consistent with his goals.

Gilb is more extreme than XP in that he says to deliver the first functionality to the user in the first week, and to deliver new user-visible functionality each and every week thereafter until death do us part. He also says not to be afraid of "throwing away" work on temporary scaffolding to keep two systems running side by side. The reduction in risk is worth the money.

Agreed, Gilb is extremely evolutionary for good practical reasons and XP is extremely practical for good evolutionary reasons.

Whoever wrote this missed an opportunity in marketing!

Gilb's focus is on business benefit and feedback rather than the practices and disciplines within the software development team. TimMackinnon reported his disappointment at the blank look he received on asking Tom about the importance of UnitTests in 1999. On the other hand Tom's high-level consultancy experience helping prioritize small team EvolutionaryDelivery by groups of hundreds of software developers at Nokia and elsewhere could I believe be harnessed with real benefit by the XP community.

It might have made more sense to him if Tim had said "Automated RegressionTesting" instead of "UnitTests." In the non-XP world, "unit testing" is just the programmer running a few ad-hoc tests by hand.

One of the pains in trying to establish a connection between the valuable things Gilb has learnt from pioneering the evo stuff over 30+ years and XP is the terminology shift. It's an inevitable problem, because like other leaders of software tribes Martin and Kent "want to feel superior too" about XpPlanningTerminology. Here's a translation table, ED -> XP, for anyone to add to:

- JuicyBitsFirst -> BusinessValueFirst
- OpenEndedSystemArchitecture ~~> SystemMetaphor *(the XP term not just new but improved)*
- Automated RegressionTesting -> UnitTests and FunctionalTests

It's interesting to note that Gilb's ideas (and the SpiralMethod of BarryBoehm who wrote the foreword in 1988 for PrinciplesOfSoftwareEngineeringManagement) have been around for so long but were only tried by a few pioneers for about ten years. Until Microsoft re-discovered them that is.

What did Gilb leave out of real importance that prevented mass take up? Maybe it was just a timing thing. My hunch is that XP now includes most that's really necessary, apart from a cure for human laziness. And the XpCoach is an important corrective for that too. The following sentence from ExtremeProgrammingSystem is one of the most ED-compatible that I have found the XP wikerature:

> "XP also delegates to customers the responsibility to set priorities in such a way as to deliver maximum business value (BusinessValueFirst) within the resource constraints of the project, and to define tests that will determine when quality requirements are met."

-- RichardDrake

[Odd choice of word: "delegates". The responsibility for setting priorities has always belonged to the customers, so how can it be delegated to them?. On occasions, they may have been foolish enough to delegate this responsibility to project managers or software engineers, but rarely with great success, I'll warrant.]

What Gilb [and XP] recognize is just that you *will* evolve. The existence of the software artifact changes the requirements for it. Since that's a fact, a development process is better insofar as it accommodates that fact.

If the project is somehow defined to be large, or geographically distributed, or otherwise encumbered, it will still learn, it will still evolve. Just glacially. -- RonJeffries

Accommodating the fact of evolution is a key strength of Gilb & XP. The thing that will prove *mass* accommodation is drastic impact on which projects ever get started: "This project is defined to be large, geographically distributed, or otherwise encumbered; therefore we won't do it". I've consulted to companies doing complex systems like Reuters since 1985 and know it isn't always that easy, but it has to happen. -- RichardDrake

---

Are GlacialEvolutionaryDelivery and RapidEvolutionaryDelivery both valid implementations of the official heading "Evolutionary Delivery"? Or does ED, in its definition, include something about rapidity of the cycles? -- AlistairCockburn

Gilb was the main method influence on ObjectiveComputerSystems from 1986. For us, EvolutionaryDelivery has to be "rapid", that is involve the smallest possible cycles, down to a week's granularity or less, as described in ScribbledOnOnePage. After RAD came along we tried to improve the term with "ControlledRapidEvolutionaryDelivery", trying to capture both

- the SteeringSoftwareProjects idea - much better control through small steps
- productivity - much more rapid in the eyes of users through small steps

CRED was written up in an article called SevenPillarsOfCred in a special Java edition of AmericanProgrammer in January 1997, at the request of EdYourdon.

As Gilb says, the ultimate judge of software development productivity is the end user. As we thought more deeply about business value and properly modeling user perceptions we came across option pricing applied to large capital projects around 1990 and began to use this stuff to show just how much better the value is from ED from start to finish. See IsEarlierCancellationFailure and AnalyzingXpWithOptionsPricing for an XP take on some of the same issues. -- RichardDrake

---

EvolutionaryDelivery is about the delivery of quality or functionality in small steps. Small means that for instance 2% of the functionality must be delivered when 2% of the project resources (time or budget) are consumed. The rapidity of cycles is then a consequence of your choice in step and project sizes. The answer is theoretically ???YES???, but cycles are usually short: TomGilb advocates weekly delivery for a one-year project, and HewlettPackard uses 2-week cycles. HP has also used PlatformDevelopment, where products are developed with short cycles on top of a "platform" project with longer cycles. This possibility is included in EvolutionaryDelivery. -- MarcoScheurer

---

Now this confuses me (I'm easily confused): *EvolutionaryDelivery is about the delivery of quality or functionality in small steps.*

Isn't that IncrementalDelivery? It took me a long time to get clear in my head the difference between EvolutionaryDevelopment and IncrementalDelivery. Now here we have EvolutionaryDelivery!? What are we talking about, please? -- KeithBraithwaite

[EvolutionaryDelivery is not **just** about delivery in small steps, it is about learning from the experience. The "study" in PlanDoStudyAct is analogous to the forces of nature in natural selection: you compare the results of the delivery with the project targets to see how much of a contribution the delivery has made (contrasted with the intended contribution). The "act" is selection itself: you decide what to do next based on where you now find yourself ("The "Act" phase decides on what course of action should be taken based on the information supplied by the "Study" phase. It is to standardize the process at a new level or, to draw new conclusions about our original theory or, to determine and select new theories (to design or modify processes).", according to Gilb [PlanguageConceptGlossary definition for Act(PDSA)])]

EvolutionaryDelivery had a meaning prior to Wiki as well as now within Wiki. And for me Marco's words are consistent with ED but not sufficient to define it. I started this page early in 1999 to describe very briefly what TomGilb chose to call EvolutionaryDelivery from the early 1980s and say a bit about Objective's experience, having been heavily influenced by Tom since 1986. The reason being that I viewed (and still view) XP as operating broadly within an ED framework. Indeed, in moving from C3's fixed 3-week user deliveries to shorter ones, where possible, XP has, I would say, moved more firmly in this direction (see JohnDaniels' point above which is now out of date). Gilb is pretty much obsessed with making the time between deliveries to your customer as short as possible *and* benefiting and learning from the feedback they provide, altering all plans accordingly. He was saying this way ahead of other software management thinkers, as TomDeMarco and others have acknowledged (although many great programmers also knew to do things this way if they possibly could).

I've always defined IncrementalDelivery as that (normally very pointless) superset of EvolutionaryDelivery that delivers in small steps but doesn't learn and change plans according to user reactions. This is theoretically possible but I don't think I've ever seen real users let it happen this way, at least if they wanted the system in the slightest. Our main challenge as developers and planners therefore is to make delivery steps as small as possible. The people paying will take care of the rest.

Nowadays one should use XP practices as the best way to ensure consistent, even increasing, velocity for ED, with the odd very high velocity burst arising from unexpected requirements ScribbledOnOnePage by you and the customer. Often your customer won't even bother to mention these things unless you've won his/her confidence with consistency in more boring areas. -- RichardDrake

---

moved here from AnAcceptableWayOfFailing

Someone is going to have to explain to me how EvolutionaryDelivery is risky. I have my own (strong) opinions. Coming firstly from experience around the 'periphery' of a number of geoscience software developments that used WaterFall and went on to fail in "nobody ever used it so all that money was wasted" sense. And then, in managing/co-ordinating a 3.5 year bespoke development project, where ED provided a very effective, practical means to control the project's risk.

ED was used to control HowToGetWhatYouPayFor by linking small step development cycles (1-3 weeks) to a combination of end user requirements, technical development 'architecture' provided by Objective and the most critical factor of all, the almighty PROJECT BUDGET.

As Brits are fond of saying ..... BRILLIANT..... ED greatly assisted in controlling costs by avoiding gambling as dictated by WaterfallBudgeting, and it even helped to bring the technical aspects of the project to a clean, crisp end. When it came time to find the inevitable AnAcceptableWayOfFailing, ED left us with a known set of (paid-up) functions and therefore little time/money was required to complete the effort. It might sound strange, but initiating the project with ED went a long towards securing AnAcceptableWayOfFailing.

-- DaveSteffe

But you think like a business manager, not a software person Dave. You are right too! -- rd

Dave, the point is not that ED is risky. It is rather, that "doing something different from what we usually do" brings along a feeling of risk. It is, in fact risky because one has not yet done it. That is the risky bit. And therefore, keeping on doing the same thing, even if it is known to fail, is more comfortable (if unsuccessful). ergo, AnAcceptableWayOfFailing. This applies to more than just software staging techniques, it applies across life habits. -- Alistair (still wishing someone will add pointers to the correct book pages)

I agree that the ED type of process is "doing something different from what **unreconstructed software people** usually do" which is why it seems to be risky to them. For business people (at least those who have not yet been "instructed" by an unreconstructed software methodologist) it's exactly the kind of incremental decision making process that they're used to in other areas and they see it for what it is - much less risky from start to finish.

There has been such a ridiculous non meeting of minds in this area, between managers and developers, largely because we as developers have had a totally wrong mindset about our *discipline*. (I'm sure that it never has anything to do with unethical commercial motives of course. The software companies in those failed WaterFall projects never received payment for all that wasted work right Dave?) -- RichardDrake

---

IMHO, I think that BusinessValue and BusinessNeeds must drive and define the TimeBoxing parameters for any iteration. This puts clear constrains on how much functionality (how many FunctionPoints?) must be included on any given iteration. In this context, rapidity, scope and quality make the difference: not any evolving methodology is equally evolutionary. -- Gast&#65533;nNusimovich

---

Related:

- AgileVsWaterfall

See ObjectionsToEvolutionaryDelivery, ConsumerSoftwareAndEvo, VerySmallSteps

---

CategoryApplicationDevelopment

Last edit February 26, 2014, See github about remodeling.