Decomposition of Projects:

How to Design Small Incremental Steps

Tom Gilb
Result Planning Limited
Tom.Gilb@INCOSE.org

Abstract: The basic premise of iterative, incremental and evolutionary project management (Larman 2003) is that a project is divided into early, frequent and short duration delivery steps. One basic premise of these methods is that each step will attempt to deliver some real value to stakeholders. While it is not difficult to envisage the steps of *construction* for a system; there is often difficulty in carrying out decomposition into steps when each step has to *deliver* something of *value* to *stakeholders*, in particular to end-users. This paper gives some guidelines, policies and principles for decomposition. It also gives a short example from practical experience. The evolutionary project management method (Evo) is used (Gilb 2005).

Introduction

See figure 1, which outlines the difference between the Waterfall method, incremental methods and evolutionary methods. This paper uses the Evolutionary Project Management (Evo) method as described by Gilb (2005).

There is evidence that evolutionary methods provide benefits. For example, Cotton (1996) states, "Benefits of Evo: The teams within Hewlett-Packard that have adopted Evolutionary Development as a project life cycle have done so with explicit benefits in mind. In addition to better meeting customer needs or hitting market windows, there have been a number of unexpected benefits, such as increased productivity and reduced risk, even the risks associated with changing the development process."

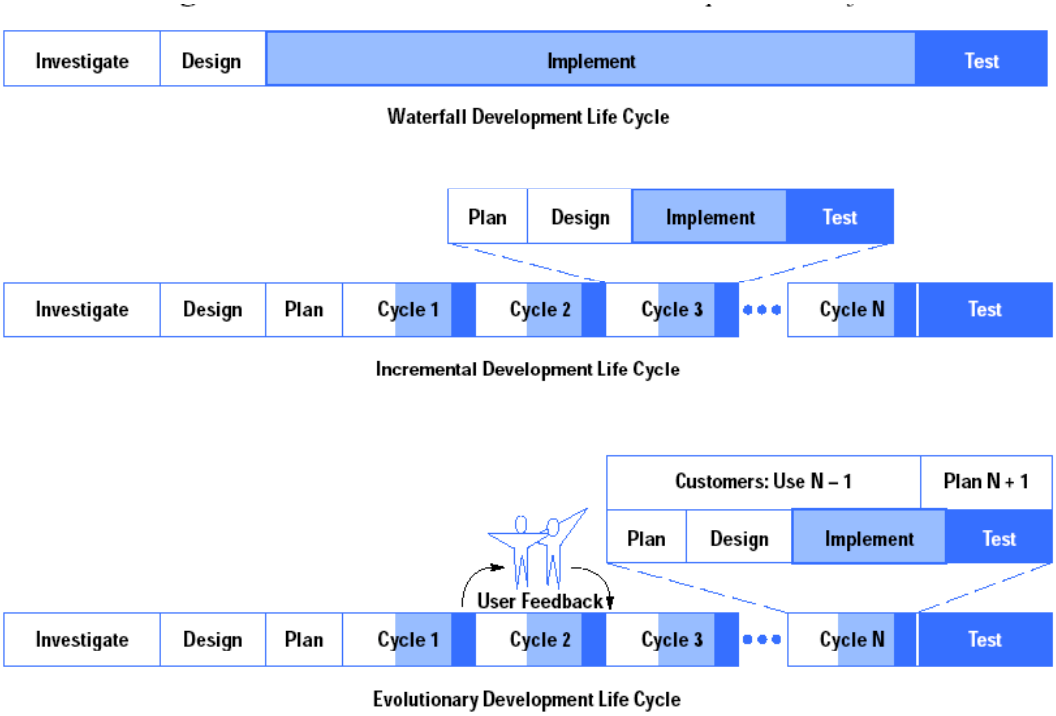

Figure 1. Life Cycle Concepts (Cotton 1996)

The experiences of a Norwegian company producing a market research tool, Conformit, also proved the benefit of using Evo. See Table 1.

Table 1: The end result of about 10 weekly result deliveries to one stakeholder, in 1 of 4 web survey product areas improved simultaneously. This was encapsulated in Product Release 8.5 of Conformit (Johansen 2004).

Description of requirement/work task	Past	Status
Usability.Productivity: Time for the system to generate a survey	7200 sec	15 sec
Usability.Productivity: Time to set up a typical specified Market Research report (MR)	65 min	20 min
Usability.Productivity: Time to grant a set of end-users access to a report set and distribute report login information	80 min	5 min
Usability.Intuitiveness: The time in minutes it takes a medium-experienced programmer to define a complete and correct data transfer definition with Confirmit Web Services without any user documentation or any other aid	15 min	5 min
Performance.Runtime.Concurrency: Maximum number of simultaneous respondents executing a survey with a click rate of 20 seconds and a response time <500 ms given a defined [Survey Complexity] and a defined [Server Configuration, Typical]	250 users	6000 users

The basic premise of iterative, incremental and evolutionary project management (Larman 2003, Larman and Basili 2003) is that a project is divided into early, frequent and short duration delivery steps. However, system designers often find it difficult to decompose a system into Evo steps. This paper will give some guidelines (in the form of policies and principles) for decomposition. It will also give an example from practical experience.

A Policy for Evo Planning

One way of guiding Evo planners is by means of a 'policy'. A general policy is given in figure 2 (you can modify the policy parameters to your local needs):

Evo Planning Policy
P1: Steps will be sequenced on the basis of their overall benefit-to-cost efficiency.
P2: No step may normally exceed 2% of total project financial budget.
P3: No step may normally exceed 2% of initial total project length.

Figure 2. Example of an Evo planning policy

Profitability Control: The *first* policy, 'P1: Steps will be sequenced on the basis of their overall benefit-to-cost efficiency' tries to ensure that we maximize benefits by delivering according to the official specified requirements values of the customer (or more generally, the stakeholder), while minimizing associated costs.

If this policy is not in place, technologists consistently choose steps, which are technologically *interesting*, and are not guided by the 'voice of the customer'. An Impact Estimation (IE) table can be used to calculate which potential steps are to be done before others (Gilb 1998). See the IE table example given in Table 2 from Johansen (2004).

Table 2: A 4-day step planned, and 4 days actually used. The step was estimated to improve 'User Productivity' by 20 minutes = 50% of the distance to the target goal of 25 minutes, from its current level of 65 minutes. User Productivity was originally 85 minutes, which minus 20 minutes (the current status improvement) gives the Past Goal of 65 minutes before this planned step. 65 minutes minus (Goal) 25 minutes = 40 minutes (100%) of improvement needed to achieve the target. Actual achievement on this one-week step was 38 minutes, or 95% of the remaining distance to Goal level (Johansen 2004).

Current Status	Improvements		Goals			Step 9			
						Design = 'Recoding'			
						Estimated impact		Actual impact	
Units	Units	%	Past	Tolerable	Goal	Units	%	Units	%
			Usability.Replaceability (feature count)						
1.00	1.0	50.0	2	1	0				
			Usability.Speed.New Features Impact (%)						
5.00	5.0	100.0	0	15	5				
10.00	10.0	200.0	0	15	5				
0.00	0.0	0.0	0	30	10				
			Usability.Intuitiveness (%)						
0.00	0.0	0.0	0	60	80				
			Usability.Productivity (minutes)						
20.00	45.0	112.5	65	35	25	20.00	50.00	38.00	95.00
			Development resources						
	101.0	91.8	0		110	4.00	3.64	4.00	3.64

Financial Control: The second policy, 'P2: No step may normally exceed 2% of total project financial budget', is designed to limit our exposure to the financial risk of losing more than 2% of the total project budget, if a step should fail totally. If this policy is not in place, technologists will insist on taking much larger financial risks, *with someone else's money*. They will wrongly, but sincerely, claim it *has* to be done, and there are 'no other alternatives'. This is usually due to lack of motivation, lack of strong management guidance, and lack of knowledge of how to divide steps into financially smaller expenditures. (We shall discuss how to handle the problem of decomposition later.)

This is really a 'gambling strategy'. How much can your project lose, and still survive, to complete the project as initially expected? If technologists cannot responsibly give value for money with small delivery steps, there is every historical reason to suspect they should not be trusted with larger budgets. The 2% number is based on observation of common practice, but it is not 'holy', and intelligent deviation is expected. It is dependent on how sure you are that a step may succeed or fail. The more conservative the step, the larger the financial risk you may be willing to consciously take.

"The Cataract approach [an Evo variation] to build planning may be likened to a rapid prototyping scenario in which the requirements for each build are frozen at the start of the build. This approach, however, is more than just grouping requirements in some logical sequence and charging ahead. Build plans must be optimized on the product, process, and organization

dimension, [you] implement the highest priority requirements in the earlier builds. Then, if budget cuts occur during the implementation phase, the lower priority portions are the ones that can [most] readily be eliminated because they were to be implemented last." (Kasser and Denzler 1995)

Deadline Control: The third policy, 'P3: No step may normally exceed 2% of initial total project length', is similar in principle to the second, except that the critical resource that is being controlled is calendar time. In this case, decompose the duration of each step by calendar time: as a rough guide, a week at a time is enough. Again, the 2% of time-to-deadline should be applied with discretion – intelligent deviation may be made. But, it is important to have a firm general guideline, or sincere people will always argue – wrongly - that they need more time before any delivery is possible. So, the 2% forces the issue. People have to argue their case well for deviation. And if they constantly deviate, they are to be suspected of incompetence. Give them help!

Cusumano and Selby (1995, page 200) state that within Microsoft, "Projects reserve about one-third of the total development period for unplanned contingencies or 'buffer time' allocated to each of the milestone subprojects." <<<Adding a buffer for some contingency is a good idea, you rarely know exactly what problems are going to be encountered, but notice that Evo will give you some reassurance that you are 'on track' and you should have delivered some early, highest-value parts of the system to your customer, which ought to help with customer relations (Of course, Microsoft have the business model of working towards major product releases).>>>

Some Examples of Evolutionary Step Size

The '2%' guideline is based on observation of common evolutionary increment sizes. For example, IBM Federal Systems Division reported (Mills 1980) that the 'LAMPS' project had 45 incremental deliveries in a 4-year period. About 2%. And, cycles were about monthly for 4 years.

This '2%' cycle is above all the 'deployment' component where step content is delivered to the **recipient** and the **host system**. It is the frequency with which the project interfaces with the 'stakeholder. It is the frequency with which we expect to get feedback on the correctness of both our own evaluations, and the users evaluation. If we allow much larger step increments, the essence of evolutionary delivery management is lost. Worst; irretrievable time can go by totally wasted, without us being aware of the loss, until it is too late to recover.

Note that we expect extensive user (and other stakeholder) interaction during development. This interaction refers not only to the feedback of requirements from one delivery to future

deliveries, but also to the intimate involvement of the users during the implementation life cycle of each delivery. Evo embraces the premise that the more the real users are involved, the more effectively the system will meet their needs. Thus, the Evo process includes a role for the users in virtually every step of the delivery life cycle and involves them, at a minimum, in the key decision processes leading to each delivery." (Spuck 1993, Section 2.3)

Microsoft (Cusumano and Selby 1995) stressed, in connection with their 24-hour evolutionary cycles (the 5 PM Daily Build of software) that it was vital to their interests to get feedback daily, so as not to lose a single day of progress without being aware of the problem.

"[A] key event [in Microsoft's organizational evolution] was a 1989 retreat where top managers and developers grappled with how to reduce defects and proposed … the idea of breaking a project into subprojects and milestones, which Publisher 1.0 did successfully in 1988. Another was to do daily builds of products, which several groups had done but without enforcing the goal of zero defects. These critical ideas would become the essence of the synch-and-stabilize process. Excel 3.0 (developed in 1989 and 1990) was the first Microsoft project that was large in size and a major revenue generator to use the new techniques, and it shipped only eleven days late" (Cusumano and Selby 1995).

While on the subject of Evo step duration, note that Microsoft has a variety of cycle lengths, up to two years for a variety of purposes. Note also that evolutionary delivery is a 'way of life' for this very successful company.

"Many companies also put pieces of their products together frequently (usually not daily, but often biweekly or monthly). This is useful to determine what works and what does not, without waiting until the end of the project – which may be several years in duration" (Cusumano and Selby 1995).

Hewlett Packard (Cotton 1996) noted that their average step size was two weeks: "There are two other variations to Tom Gilb's guidelines that we have found useful within Hewlett-Packard. First, the guideline that each cycle represent less than 5% of the overall implementation effort has translated into cycle lengths of one to four weeks, with two weeks being the most common. Second, ordering the content of the cycles is used within Hewlett-Packard as a key risk-management opportunity. Instead of implementing the most useful and easiest features first, many development teams choose to implement in an order that gives the earliest insight into key areas of risk for the project, such as performance, ease of use, or managing dependencies with other teams" (Cotton 1996).

Principles for Decomposing a System into Evo Steps

Almost everyone has various problems reducing their initial concepts of a system into Evo steps of a suitable '2%' size. It is, however, almost invariably possible to do so. Everyone can be taught how to do it. But many highly educated, intelligent, experienced 'engineering directors' cannot quite believe this, until they experience it themselves. This is a *cultural* problem, not a technological problem.

Some principles for decomposing a system into Evo steps are given in figure 3.

Principles for decomposing a system into Evo steps

1. *Believe* there is a way to do it, you just have not *found* it yet!

2. *Identify* obstacles, but don't use them as excuses: use your imagination to get *rid* of them!

3. Focus on *some usefulness* for the user or customer, however small.

4. Do not focus on the design ideas themselves, they are distracting, especially for small initial cycles. Sometimes you have to ignore them entirely in the short term!

5. Think; one customer, tomorrow, one interesting improvement.

6. Focus on the *results* (which you should have defined in your goals, moving toward target levels).

7. Don't be afraid to use temporary-scaffolding designs. Their cost must be seen in the light of the value of making some progress, and getting practical experience.

8. Don't be worried that your design is inelegant; it is results that count, not style.

9. Don't be afraid that the customer won't like it. *If* you are focusing on results *they want*, then by definition, *they* should like it. If you are not, then *do*!

10. Don't get so worried about 'what might happen afterwards' that you can make no practical progress.

11. You cannot foresee everything. Don't even *think* about it!

12. If you focus on helping your customer in practice, *now*, where they *really* need it, you will be forgiven a lot of 'sins'!

13. You can understand things much better, by getting *some* practical experience (and removing *some* of your fears).

14. Do *early* cycles, on willing local mature parts of your user community.

15. When some cycles, like a purchase-order cycle, take a long time, initiate them early, and do other useful cycles while you wait.

16. If something seems to need to wait for 'the big new system', ask if you cannot usefully do it within the 'awful old system', so as to pilot it realistically, and perhaps alleviate some 'pain' in the old system.

17. If something seems too costly to buy, for limited initial use, see if you can negotiate some kind of 'pay as you really use' contract. Most suppliers would like to do this to get your patronage, and to avoid competitors making the same deal.

18. If *you* can't think of some useful small cycles, then talk directly with the real 'customer' or end-user. They probably have dozens of suggestions.

19. Talk with end-users in *any* case: they have insights you need.

20. Don't be afraid to use the old system and the old 'culture' as a launch-platform for the radical new system. There is a lot of merit in this, and many people overlook it.

Figure 3. Principles for decomposing systems into Evo steps (Gilb 2005)

The list given in figure 3 was compiled by review of the author's decades-long practices and experiences. One or more of these hints should enable you to find a practical solution. Finally, when you have enough successful experience to realize that there always seems to be a way to decompose into small evolutionary steps, you will give up the mentality of saying *'impossible!'* and concentrate on the much more constructive work of finding a reasonable solution. The mentality that decomposition 'can't be done' is built on a number of misconceptions, so let us deal with some of them.

Misconceptions about Evo Step Decomposition

By decomposition, we do not mean that '2% of a car' is going to delivered. We will probably be dealing with 'whole cars'. But we may start with our old car and put better tires on it, to improve braking ability on ice.

The problem of getting 'critical mass', meaning 'enough system to do any useful job with real users', is usually dealt with by one of two methods:

I: Make use of existing systems (while ultimately evolving away from them)

II: Build the critical mass in the 'backroom', invisible to the user.

Then, as the large components (the components that take longer to 'ready' for delivery than your Evo step cycle length) become ready, consider delivering them in future regular delivery cycles.

The conceptual problem of 'dividing up the indivisible', which people wrongly perceive as a problem, is explained by pointing out that:

- We do *not* ask you to act like Solomon and demand to 'split the baby in half'
- We are *not* talking about system 'construction'
- We *are* talking about delivering 'results', not technical components, in small increments.

So, the systems are *organically whole*. No unreasonable chopping them into non-viable components is being asked. But we are still not going to overwhelm the user with all the capability of the system. For example, you probably learned PC software, like your word processor or email capability gradually. It might have been delivered to you as five million software instructions from Microsoft, with 20,000 features, most of which *you* do not need. But you could ignore most of them, and focus on getting your current job done.

You probably picked up additional tricks gradually. You found them by experimentation with the menus. A friend showed you some tricks. You might be one of those weirdoes who actually read a manual. Maybe you looked new capability up, using the Help menu. Maybe you got help from a help desk. You probably did not go on a two-week course, with an examination, to learn to be proficient at a high level. Nevertheless, the entire system was 'delivered' to you all at once. Then a new version became available, a million lines of code at once, but again you probably used it the same way as last time, picking up new features gradually.

Microsoft is not worried about building these features as each new user needs them. They build what they must for their market in their 'backroom', and we deliver to ourselves, in our 'frontroom' at an evolutionary learning pace, based on need and ability. The key idea is the incremental *use* of a system, which increases the *user (or at least internal stakeholder such as sales or help desk)* capability. Building and construction are semi-independent topics. You *can* build a system in the same step cycle time as it takes to deploy with a user, but you do not *have* to!

Some Additional Ideas for Decomposition

Here are some additional ideas for decomposition, which build on what we discussed earlier:

- Divide by deadline: do things early as needed by external stakeholders.

- Divide by stakeholder: there are typically about 30 internal and external stakeholders (for example, developers, suppliers, senior managers and end-users) in any project.
    - Deliver to one of these at a time (for example, to the salesperson who must demonstrate early to customers).
    - Divide by experience of stakeholder: deliver to the very experienced stakeholders first, and the novices later (or vice versa).
    - Divide by individual stakeholder: deliver to one particular local stakeholder that you have a good relationship with.
    - Deliver to an internal example of a stakeholder (that is, someone with the same function within your organisation), so that if something goes wrong you can control the bad news.
    - Deliver to a highly co-operative stakeholder that will give you adequate time (Johansen 2004)
- Divide by geography.
    - Deliver in your site's town or country before conquering the world.
    - Deliver where critical leading edge clients are to be found.
- Divide by value: deliver the highest value first, but remember that different stakeholders have quite different values, and value is not always financial.
    - Divide by tasks in a set of tasks: divide into sets of tasks, so that some useful tasks are operable early and 'luxury' tasks, which are infrequently used, come last.
    - Divide by risk level: sometimes the highest 'value' to internal stakeholders, is to get the high risk strategies and architectures tested to make sure that they really will work, before investing more on scaling up.

Summary

Most systems are capable of decomposition into Evo steps: an early (next week), continuous series of about one-week-to-delivery-to-stakeholder result cycles. The decomposition is by value delivery increments; it is not about 'construction' increments.

Evo decomposition can follow a policy that might prioritize profitability, or risk control, or effect on stakeholders, or any combination of interesting objectives.

Evo decomposition can be done as a process, using a set of known principles of

decomposition. The most fundamental of these principles is to partially deliver improvements in performance, especially quality levels, to stakeholders.

References

Cotton, T., "Evolutionary Fusion: A Customer-Oriented Incremental Life Cycle for Fusion", Hewlett-Packard Journal, August 1996, Vol. 47, No. 4, pages 25-38. http://www.hpl.hp.com/hpjournal/96aug/aug96a3.htm

Cusumano, M.A. and Selby, R.W., *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, The Free Press (Division of Simon and Schuster), 1995, ISBN 0-02-874048-3, 512 pp.

Gilb, T., *Competitive Engineering, A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Elsevier Butterworth-Heinemann, 2005. ISBN 0750665076, Sample chapters can be found at http://www.Gilb.com as follows:

Chapter 5: Scales of Measure:
http://www.gilb.com/community/tiki-download_file.php?fileId=26

Chapter 10: Evolutionary Project Management:
http://www.gilb.com/community/tiki-download_file.php?fileId=77

Gilb, T., "Adding Stakeholder Metrics to Agile Projects", *Cutter It Journal: The Journal of Information Technology Management*, July 2004, Vol. 17, No.7, pp31-35.

Gilb, T., "Evolutionary Systems Engineering Principles", 2004, Unpublished. Available on request from the author.

Gilb, T., "Impact Estimation Tables: Understanding Technology Quantitatively", *Crosstalk*, December 1998. http://www.stsc.hill.af.mil/CrossTalk/frames.asp?uri=1998/12/gilb.asp

Gilb, T., *Software Metrics*, Studentlitteratur (Sweden), 1976. Also Winthrop (USA), 1977.

Johansen, T., "From Waterfall to Evolutionary Development (Evo) or, How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market", *Proceedings of European Conference on Software Process Improvement (EuroSPI 2004)*, 2004. From which evolved, Gilb, T. and Johansen, T., "From Waterfall to Evolutionary Development (Evo): How we rapidly created faster, more user-friendly, and more productive software products for a competitive multi-national market", *Proceedings of the INCOSE Conference*, 2005. Available from http://www.gilb.com/community/tiki-download_file.php?fileId=32

Kasser, J. and Denzler, D.W.R., "Designing Budget-Tolerant Systems", *NOCOSE Proceedings*, 1995.

Craig Larman, C., *Agile and Iterative Development, A Managers Guide*, Addison Wesley, 2003. ISBN 0131111558 (pbk). 342 pages. Chapter 10 is on Gilb's Evo method.

Craig Larman, C. and Basili, V.R., "Iterative and Incremental Development: A Brief History", *IEEE Computer*, June 2003, pages 2-11.

May, E.L. and Zimmer, B.A., "The Evolutionary Development Model for Software", *Hewlett-Packard Journal*, August 1996, Vol. 47, No. 4, pages 39-45. http://www.hpl.hp.com/hpjournal/96aug/aug96a4.htm (not directly referenced in paper but relevant)

Mills, H.D., "The Management of Software Engineering. Part 1: Principles of Software Engineering", *IBM Systems Journal*, 1980, Volume 19, Number 4. Reprinted 1999 in *IBM Systems Journal*, Volume 38, Numbers 2 & 3. A copy is downloadable from http://www.research.ibm.com/journal/.

Spuck, W., "Rapid Delivery Method", *Internal Report (D-9679) of Jet Propulsion Labs, CIT, Pasadena CA*, 1993.

Biography

Tom has been an independent consultant, teacher and author, since 1960. He mainly works with multinational clients; helping improve their organizations, and their systems engineering methods.

Tom's latest book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (2005).

Other books are 'Software Inspection' (with Dorothy Graham, 1993), and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976, Out of Print) has been cited as the initial foundation of what is now CMMI Level 4.

Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, 'Planguage'. He is a member of INCOSE and is an active member of the Norwegian chapter NORSEC. He participates in the INCOSE Requirements Working Group, and the Risk Management Group.

Email: Tom@Gilb.com

URL: http://www.Gilb.com

Version April 5 2008 TG Edit Last before handover to INCOSE

Thanks to Lindsey Brodie, Middlesex University for editing advice and help!