Diseconomies of scale

Allan Kelly, allan@allankelly.net

Software Development and Agile consultant

Without really thinking about it you are not only familiar with the idea of economies of scale: you expect economies of scale. Much of our market economy operates on the assumption that when you buy or spend more you get more per unit of spending. The assumption of economies of scale is not confined to free-market economies: the same assumption underlies much Communist-era planning.

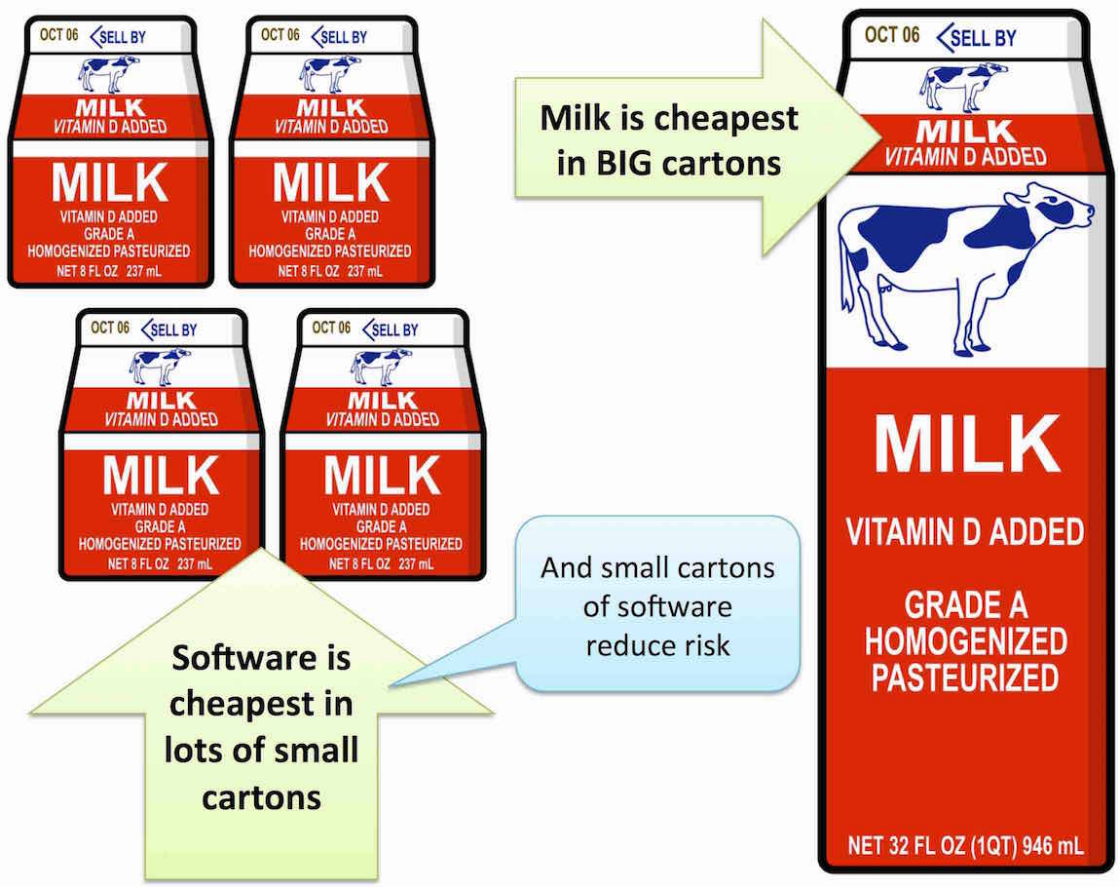
At some stage in our education – even if you never studied economics or operational research – you will have assimilated the idea that if Henry Ford builds a million identical black cars and sells a million cars, then each car will cost less than if Henry Ford manufactures one car, sells one car, builds another very similar car, sells that car, and continues in the same way another 999,998 times.

The net result is that Henry Ford produces cars more cheaply and sells more cars more cheaply so buyers benefit. This is *economies of scale.*

The idea and history of mass production and economies of scale are intertwined. I'm not discussing mass production here, I'm talking *economies of scale*, *diseconomies of scale* and software development.

Milk is cheaper in large cartons

That economies of scale exist is common sense: every day one experiences situations in which buying more of something is cheaper per unit than buying less. For example, you expect that in your local supermarket buying one large carton of milk – say four pints – will be cheaper than buying four one-pint cartons.



Small cartons of software are cheaper and less risky

So ingrained is this idea that it is newsworthy when shops charge more per unit for larger packs complaints are made. In April 2015 *The Guardian* newspaper in London ran this story headlined "UK supermarkets dupe shoppers out of hundreds of millions" about multi-buys which were more expensive per item than buying the items individually.

Economies of scale are often cited as the reason for corporate mergers. Buying more allows buyers to extract price concessions from suppliers. Manufacturing more allows the cost per unit to be reduced, and such savings can be passed on to buyers if they buy more. Purchasing departments expect economies of scale.

I am not for one minute arguing that economies of scale do not exist: in some industries economies of scale are very real. Milk production and retail are examples. It is reasonable to assume such economies exist in most mass-manufacturing domains, and they are clearly present in marketing and branding.

But – and this is a big 'but'...

Software development does not have economies of scale

In all sorts of ways, software development has diseconomies of scale. If software development was sold by the pint, then a four-pint carton of software would not just cost four times the price of a one-pint carton, it would cost *far more*.

Once software is built there are massive economies of scale in reselling (and reusing) the same software and services built on it. Producing the first piece of software has massive marginal costs; producing the second identical copy has a cost so close to zero it is unmeasurable – Ctrl-C, Ctrl-V.

Diseconomies abound in the world of software development. Once development is complete, once the marginal costs of one copy are paid, then economies of scale dominate, because marginal cost is as close to zero as to make no difference.

Diseconomies

Software development diseconomies of scale have been observed for some years. Cost estimation models like COCOMO actually include an adjustment for diseconomies of scale. But the implications of diseconomies are rarely factored into management thinking – rather, economies-of-scale thinking prevails.

Small development teams frequently outperform large teams: five people working as a tight team will be far more productive per person than a team of 50, or even 15. Academic studies have come to similar findings.

The more lines of code a piece of software has, the more difficult it is to add an enhancement or fix a bug. Putting a fix into a system with a million lines of code can easily be more than ten times harder than fixing a system with 100,000 lines.

Experience of Kanban style *work in progress* limits shows that doing less at any one time gets more done overall.

Studies show that projects that set out to be *big* have far higher costs and lower productivity per deliverable unit than small systems.

Testing is another area where diseconomies of scale play out. Testing a piece of software with two changes requires more tests, time and money than the sum of testing each change in isolation.

When two changes are tested together the combination of both changes needs to be tested as well. As more changes are added and more tests are needed, there is a combinatorial explosion in the number of test cases required, and thus a greater than proportional change in the time and money needed to undertake the tests. But testing departments regularly lump multiple changes together for testing in an effort to exploit economies of scale. In attempting to exploit non-existent economies of scale, testing departments increase costs, risks and time needed.

If a test should find a bug that needs to be fixed, finding the offending code in a system that has fewer changes is far easier than finding and fixing a bug when there are more changes to be considered.

Working on larger endeavors means waiting longer – and probably writing more code – before you ask for feedback or user validation when compared to smaller endeavors. As a result there is more that could be 'wrong', more that users don't like, more spent, more that needs changing and more to complicate the task of applying fixes.

Think diseconomies, think small

First of all you need to rewire your brain: almost everyone in the advanced world has been brought up with economies of scale since school. You need to start thinking *diseconomies of scale*.

Second, whenever faced with a problem where you feel the urge to 'go bigger', run in the opposite direction: go smaller.

Third, take each and every opportunity to go small.

Fourth, get good at working 'in the small': optimize your processes, tools and approaches to do lots of small things rather than a few big things.

Fifth – and this is the killer: know that most people don't get this at all. In fact, it's worse, they assume bigger is better.

This is an except from Allan Kelly's latest book Continuous Digital which is now available on Amazon and in good bookshop.

About Allan Kelly

Allan Kelly helps companies large and small enhance their agile processes and boost their digital products. Past clients include: Virgin Atlantic, Qualcomm, The Bank of England, Reed Elsevier and many small innovative companies you've never heard of. He invented Value Poker, Time-Value Profiles and Retrospective Dialogue Sheets. A popular keynote speaker he is the author of "Dear Customer, the truth about IT" and books including "Project Myopia", "Continuous Digital", "Xanpan" and "Business Patterns for Software Developers". His blog is at https://www.allankellyassociates.co.uk/blog/ and on twitter he is @allankellynet.