

Software Estimation in an Agile Environment

Robert Armstrong

Tassc Limited
www.tassc-solutions.com

First Published: October 2009

Copyright 2010, Tassc Limited. All Rights Reserved.

Some say, there are no good software project managers – only lucky ones. We find that the more you plan, estimate and track – the luckier you get!

In an agile software development environment you still need to plan and schedule what to do and when to do it, calculate estimates of timescales, costs and resource requirements, and track progress when development gets underway – this paper shows you how.

Let's All Be Agile

*everyone's
doing it*

You're considering adopting an agile approach to developing software; maybe because you need to reduce costs, speed up development time, improve quality, minimize risk, or for any of a dozen or so other perfectly good reasons. Moreover, you're encouraged and reassured to discover that agile processes are used by many of the world's leading companies, from Cisco Systems and Google to Lockheed Martin, Microsoft and Yahoo.

*'lightweight' or
'heavyweight',
you still need a
process*

You already understand that agile is a 'lightweight' generic approach with many variations (Scrum, Dynamic Systems Development Method, Lean Development, Extreme Programming, Feature Driven Development, among others [1]), all of which share the Principles expressed in the Manifesto for Agile Software Development [2]. You also recognise that there are almost as many similarities as there are differences between agile and traditional 'heavyweight' approaches to software development. After all, you still need to plan what you're going to do, analyse the problem, design and build a solution, test, integrate, review and deploy it – performing these activities either in one pass or repeating some of them many times.

*iterative and
incremental is
the key to
flexibility*

But it's the differences that are most interesting: agile favours 'iterative' development rather than 'waterfall', and 'incremental' rather than 'big bang' project delivery, and emphasizes short timeboxes, empowered teams, collaboration and flexibility. Perhaps it's the flexibility and responsiveness to change that underpins the pragmatism of an agile approach and sets it apart.

Can An Agile Project Be Managed?

*if requirements
are allowed to
change freely
the rate of
change will
exceed the rate
of progress*

Much has been written about agile development, and in spite of the inevitable hype most of it is very good, yet there are still many misconceptions surrounding the approach; the most worrying being that an agile project is difficult to manage. How can you plan and estimate what work has to be done or schedule a delivery when the Agile Manifesto favours, yes favours, “responding to change over following a plan”? Opponents of agile methods, often those with a vested interest in structured and waterfall techniques, perceive this difficulty as a fatal flaw.

*evolving
requirements
don't need to
wreck the plan*

Uncontrolled change is of course disastrous on any project, waterfall and agile alike, but experienced project managers know that requirements do evolve as a project progresses. Rather than trying to resist or defer change for the sake of following the plan, changes are accepted, assessed and prioritised. Recognising that change is not only possible but also likely, the challenge for agile project managers is being responsive to change without the need to rework the whole project plan.

*change is the
only constant*

As if to make matters worse, change is actively encouraged in an agile environment through close and frequent collaboration between the project team and the business. As working code is delivered and users have early visibility of the software, there is a natural desire from users and developers alike to improve, enhance and extend application functionality.

Any Project Can Be Estimated Accurately (Once It's Completed)

*software
engineering is
like real
engineering*

The solution to the problem of controlling change on a software project becomes clear after we remove some of the historical rubble left over from the remains of analogies often drawn between software engineering and construction engineering. After all, projects in both disciplines follow a process. Processes comprise a number (possibly only one) of sequential phases; each phase has objectives and a milestone; workers are organised and assigned to tasks in order to achieve the objectives; tasks are the units of work and are scheduled in an inter-related network of dependencies.

*real engineering
can be
accurately
predicted*

This comparison has proven useful to software project managers, and many of the ideas and much of the terminology from civil engineering projects have been adopted and are now routine on software projects. This has led some stakeholders and project managers to the mistaken belief that software engineering and construction engineering are identical, to the extent that they try to plan and estimate software projects from start to finish in fine detail, creating detailed work breakdown structures at the person-day level months or years in advance.

*software's
unique flexibility
is difficult to
predict
accurately using
traditional
approaches*

However, the comparison only works up to a point, beyond which the initial usefulness of the analogy becomes a severe limitation. The failures of such a rigid approach to software development are well documented [3], and far outweigh the successes. Software is of course fundamentally different from any of the physical engineering projects that produce buildings, tunnels, dams or ships – real artifacts, whose form is set at the outset of the project and remains fixed (more or less) for the remainder of its operational life. Software isn't tangible in anything like the same sense, and its form can be changed, and changed again, throughout its operational life. This unique plasticity is a tremendous advantage to software architects and designers, as well as to the business, but it stretches traditional structured and waterfall approaches to project management beyond breaking point.

Agile approaches on the other hand aim to strike a balance and exploit the unique flexibility inherent in software construction and its ability to accommodate change, while maintaining enough rigor and control of change to prevent a project degenerating into what many would consider to be ‘cowboy coding’.

Agile Planning And Scheduling – Into The Unknown

*the project
timebox and
other constraints*

Software projects are often bound by pre-defined delivery dates (the project timebox), hard deadlines imposed by legislative, contractual or other good reasons, where the date of delivery is as important as the delivery itself. If the project delivers after the deadline, the delivery loses some of its value. Fixed budgets, established deployment infrastructure and limited resource availability impose further constraints on planning and scheduling. Agile project managers take the view that defining which tasks need to be performed and who will work on them doesn't need to be done in advance from start to finish in detailed work breakdown structures.

timeboxing

In order to enable delivery on time, and to an agreed level of quality, where development is driven by deadlines and other constraints, agile planning and scheduling techniques are based on Timeboxing – “a process by which defined objectives are reached at pre-determined and immovable dates through continuous prioritisation and flexing of requirements” [4].

*one strategic
plan*

Agile projects need a flexible, multi-level, planning approach – an overall strategic view, along with numerous detailed tactical views. The project plan is an outline of the project, coarse-grained and at the highest level, that provides a strategic view of the development process phases, milestone dates, deliverables, timescales, risks, resource profile, constraints and budget.

*many tactical
plans*

Iteration plans are tactical, one per iteration. A tactical plan is like a magnifying glass moving over the strategic plan exposing a detailed view of tasks and their inter-dependencies, worker allocation to tasks, milestones, deliverables and delivery dates. During a live agile project, each tactical plan keeps the detailed planning horizon short, just a few weeks in some cases. Continuous planning and tactical scheduling, based on actual progress measurement inherent in an agile environment, steadily increases the reliability of, and confidence in, the schedule and improves the accuracy of estimates.

Let The Business Drive The Schedule – Who'd Have Thought It?

*agile projects
deliver flexibility*

The Agile Principles state: “Agile processes harness change for the customer's competitive advantage”. Competitive advantages, typically described in a business case or vision statement, provide the rationale and justification for investment in systems development. Maintaining traceability from business objectives to individual projects through terms of reference, or some other initiation document, provides the context for prioritising requirements and assessing the impact of business change.

*fitness for
purpose of the
deliverable is top
priority*

New or changing requirements affect the scope and priority of what is to be developed, and in any timebox (iteration, phase or whole project) there is a limit to what can realistically be produced – a trade-off between resource availability and quality – where low priority items are simply dropped in favour of quality. So rather than having to rework the whole project plan, the only remaining decision is *when* a given requirement will be satisfied – a simple scheduling problem, solved by allowing business objectives to be the driver.

*prioritise using
MoSCoW*

The MoSCoW technique [5], common in agile development, prioritises requirements from top priority Must Have (essential), to Should Have, Could Have and lowest priority Won't Have this time. A timebox comprises a mix of requirements at different levels of priority (agreed jointly by developers and the business), so that under schedule pressure lower priority requirements can be dropped in favour of delivering higher priority requirements on time and to an acceptable level of quality.

Let Actual Progress Drive The Estimates – How Else?

an iteration is a mini project

Some, ourselves included, have discovered that managing an agile project is actually much easier than it first appears, much easier than trying to plan the entire software project at the outset. In agile, the project regularly and frequently produces actual reliable metric data at the end of each timebox – in methods like Scrum for example, data is available every 15 to 30 days.

smart managers use agile to hone their skills

Rather than resisting agile methods and sticking to traditional approaches to software development, smart project managers embrace agile as a unique opportunity to improve their skills in planning and estimating. Managers can use this early, valuable data to make planning and scheduling decisions about the next timebox and to estimate with increasing accuracy and confidence when the project will be complete. It is the early and continuous production of this actual metric data that is the key to success in agile development.

agile managers do it with metrics

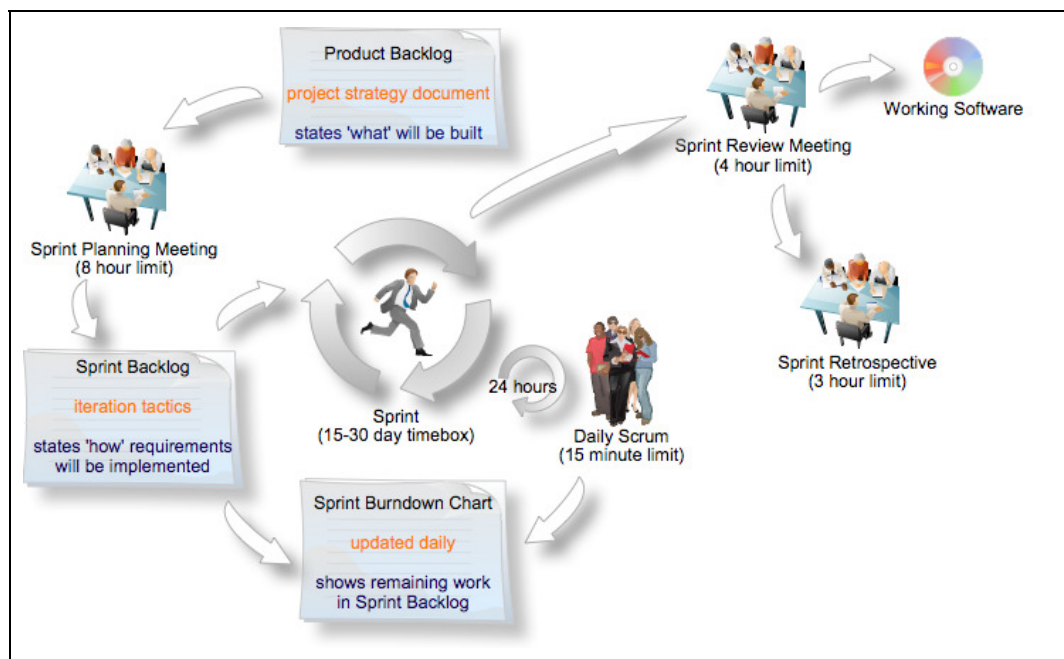
If you adopt an agile approach to development, and you don't capture and use this data, then you're not truly agile. Management success with agile depends as much on steering the plan based on measurement of actual progress as development depends on steering application functionality based on feedback from the business.

Scrum: An Agile Example

Scrum is popular and easy to use

The remainder of the paper provides an example of agile development from a planning and scheduling perspective. Agile development, being a generic approach with many variations, is best illustrated by example, and one of the most popular agile methods, Scrum (itself an iterative and incremental framework for managing complex work) is ideal. One of Scrum's biggest advantages (and the reason for its popularity) is that it's easy to learn and requires little effort to start using.

Scrum isn't an acronym, the word has been borrowed to describe a collective form of software development derived from the observation that rugby players 'scrum' when they surround the ball and push forward, in a concentrated burst of activity, together as a team [6].



Scrum : an agile framework for early and frequent delivery.

Scrum Project Initiation

a project initiation document

As with many other types of systems development methods, a Scrum project initiation typically starts with a high-level document describing the project, its scope, personnel, stakeholders, dependencies and risks. Hard deadlines and budgets are usually pre-set.

the Product Backlog is strategic

In Scrum, broad descriptions of all required features to be built are recorded in a Product Backlog document, which also contains rough estimates of both business value and development effort – this establishes the strategic plan. Estimates of effort are based on an initial set of metric values for software artifacts. Estimates of duration and cost are calculated using skill levels, availability and pay rates of team members. Planning and scheduling then become a negotiation on the balance between what functionality can be produced (to an acceptable level of quality, subject to known risk) within the constraints of the project timebox, budget and resources.

Ready, Steady, Sprint!

the Sprint Backlog is tactical

A Sprint Planning Meeting (limited to 8 hours) marks the beginning of the sprint cycle, every 15–30 days. The team and the business select what work is to be done and prepare the Sprint Backlog; a detailed document containing information about how the team is going to implement the requirements for the upcoming sprint – a tactical plan.

software architecture is a platform

Early iterations are usually concerned with the core architecture of the application, the structure that will provide a stable, yet flexible foundation. Subsequent sprints build on this base by incrementally adding features based on priority. How much needs to be developed in the early sprints varies between projects and depends on familiarity with the problem domain and complexity of the non-functional requirements (such as usability, availability, security, concurrency, distribution and performance). These technical aspects of the system may already exist as part of the deployment infrastructure.

tasks

Tasks in a sprint are broken down into hours, with no task being more than 16 hours. Tasks are signed up for by the team members rather than assigned, and the team sets the estimates (based on software metrics, team member skill level and risk). Each sprint includes requirements gathering and analysis, refinement of the design, coding, refactoring and testing.

short-term plans are more reliable than long-term plans

Producing an estimate for a small subset of the requirements over a short planning horizon of just a few weeks for a sprint (deferring estimates for future sprints) is likely to be more accurate than an estimate months or years into the future, particularly as we know at the outset that change is inevitable.

the daily Scrum

Sprint Burndown Chart

Each day during the sprint, a project status meeting occurs, a Scrum, timeboxed at 15 minutes. Attendees usually stand as it helps to keep meetings short, and each team member answers three questions: What have you done since yesterday? What are you planning to do by today? Do you have any problems preventing you from accomplishing your goal? A Sprint Burndown Chart, showing remaining work in the Sprint Backlog (updated daily) provides a view of the sprint progress.

an empowered team

During a sprint, managers must allow the team to focus on producing the product to quality and within the timebox (everything else is extraneous), and empower them to get on with the job of the sprint. No changes to requirements are allowed during the sprint, their impact will be assessed in the next Sprint Planning Meeting.

Don't Look Back In Anger

<i>Sprint Review Meeting</i>	A Sprint Review Meeting (limited to 4 hours) at the end of the sprint is where the team demonstrates working software to the stakeholders, and considers work that was completed and not completed during the sprint. Typically, each sprint delivers a production-quality deployable release, although in practice deployment is usually driven by the business cycle to minimize disruption.
<i>critical success factors</i>	Early and frequent demonstration of tangible progress in the form of working software that delivers measurable business benefit, is not only reassuring to the business, it also helps develop and build collaboration and trust – critical success factors in agile development.
<i>Sprint Retrospective</i>	A Sprint Retrospective (3 hour limit) allows team members to reflect on the sprint and assess what went well and what could be done to improve the next sprint.
<i>measuring actual progress is the key</i>	It's at this point that project managers can make their greatest impact – measure the actual effort that was required to build what was actually delivered by those who actually performed the work – and calibrate their metrics in preparation for calculating a more reliable estimate for the next sprint.
<i>a golden opportunity for project managers</i>	Calibration is a comparison of the estimate of effort required to develop the artifacts that were produced (based on initial metric values) and the actual effort it took. If estimates are consistently higher on average than the actuals then the manager can reduce the metrics, or if estimates are lower, increase the metrics. Likewise if estimates are specifically higher for a particular team member then the manager can increase their skill level, or if estimates are lower, decrease their skill level. This fine-tuning of essential metric data is a golden opportunity for project managers to learn, develop and refine core planning, estimating and scheduling skills.

Conclusion

<i>agile offers early and frequent delivery of working software driven by business change</i>	Agile is a 'lightweight' generic approach to software engineering characterised by iterative development, incremental delivery, collaboration, minimum ceremony and flexibility. It is, in many ways, a return to traditional practices common at the beginning of the software development industry; almost as a reaction to 'heavyweight' waterfall methods that had become established by the 1990s, now seen as micro-managed, bureaucratic, slow and inflexible. Early and frequent deliveries of working software and flexibility in response to business change, both of which are difficult to achieve using structured approaches, are at the heart of agile methods such as Scrum.
<i>short timeboxes deliver actual results and are easier to plan</i>	The short planning horizon of weeks in a typical agile project is much easier to manage than planning a project from start to finish in fine detail months or years in advance. Strategic planning and tactical scheduling of agile projects, using a series of fixed timeboxes, is driven by continuous re-prioritisation of new and changing business requirements. The success of an agile project depends as much on shaping tactical iteration plans based on actual progress measurement as it does on shaping application functionality based on user feedback.
<i>predictions based on past performance are the most reliable</i>	An agile project regularly produces actual reliable metric data at the end of each timebox; a golden opportunity for agile project managers to make better informed planning and scheduling decisions about the next timebox and to estimate with ever-increasing accuracy and confidence – practice makes perfect.

Web Links

- 1 Popular Agile Methods:
<http://www.scrumalliance.org/>
<http://www.dsdm.org/>
<http://www.lean.org/>
<http://www.extremeprogramming.org/>
<http://www.agilemodeling.com/essays/fdd.htm>
- 2 Agile, the Manifesto and Principles:
<http://www.agilealliance.org/>
<http://agilemanifesto.org/>
<http://agilemanifesto.org/principles.html>
- 3 Software Project Management Failures:
<http://net.educause.edu/ir/library/pdf/NCP08083B.pdf>
<http://www.stsc.hill.af.mil/crosstalk/2004/10/0410Jones.html>
- 4 Timeboxing:
<http://en.wikipedia.org/wiki/Timeboxing>
<http://www.dsdm.org/atern/techniques/timeboxing/timebox-planning/>
- 5 MoSCoW Requirements Prioritisation:
http://www.search.com/reference/MoSCoW_Method
<http://www.dsdm.org/atern/techniques/moscow-prioritisation/the-moscow-rules/>
- 6 The Scrum agile framework:
[http://www.search.com/reference/Scrum_\(development\)](http://www.search.com/reference/Scrum_(development))
<http://www.scrumalliance.org/>
<http://scrummethodology.com/>
<http://www.controlchaos.com/>