Agile Software Development

KEMP LITTLE

KEMP LITTLE

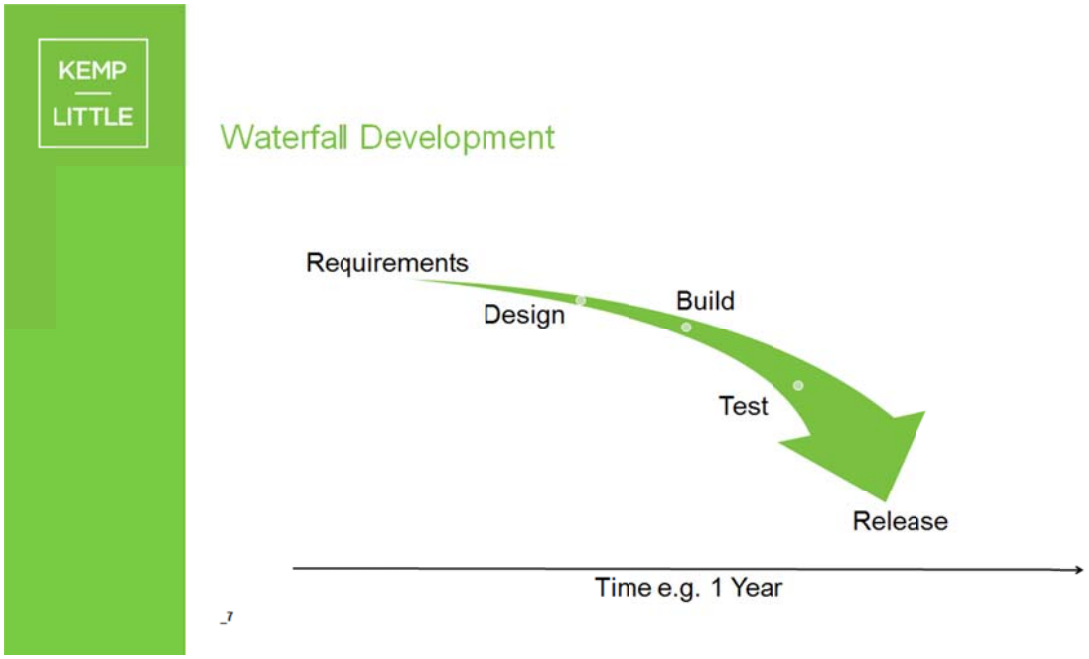
Contents

1 Introduction: What is Software Development?

The term “software development” is used to describe the set of steps or phases that a computer program goes through when it is being developed. Software development is not just about writing computer code, but also extends to the whole process from conception, design and creation through to deployment in a planned and structured process.

The processes around software development are not new concepts and in fact originate from similar long-standing principles employed in the manufacturing and construction industries. One of the longest standing and most popular forms of software development is known as “Waterfall Software Development”. The waterfall development model follows a single sequential design process, as shown in Figure 1 below. In the requirements phase, an exercise is completed whereby all the requirements and parameters of the software project are determined. Moving to the next phase, software is designed to meet these requirements in a way that is analogous to the creation of blueprints for a construction project. Following completion of the design, the software is then built to these designs and then tested to uncover and fix any defects in the software (known as bugs), before being released.

Figure 1: Waterfall Software Development



2 What are the Problems with the Waterfall Software Development Model?

While the waterfall model can be an effective method for developing software, there have, however, been a number of problems encountered, and criticisms subsequently levied against this model:

(i) **Artificial division of development stages** – waterfall operates on the assumption that each of the development stages can and should be completed before moving on to the next phase. This may not align to eventualities which are encountered during the development of the software. For example, during the detailed design or build phases new requirements of the software project may be determined which are not easy to accommodate under the siloed and progressing nature of this development model.

(ii) **Project hinges on complete and accurate requirement gathering phase** – the first stage of the waterfall model is the project requirement scoping phase. The whole software development project is then designed, built and tested to completion against these requirements. This can be a very effective method of developing software against detailed requirements where these requirements can be determined at the start of a project. Conversely, however, this model does not easily accommodate a situation where requirements are not fully understood or where requirements are subject to change over relatively short timescales. Additionally, requirements are sometimes not fully understood until a first version of software is deployed and shortcomings can be observed in practice.

(iii) **Changes "up the waterfall" can mean the development phases need to be repeated** – as highlighted above, a variety of circumstances may arise during the software development lifecycle which necessitate a change in requirements. For example, business focus may change or unforeseen circumstances (such as a change in law) may mean requirements have to be adapted to accommodate such changes. A change to the requirements, once these have been scoped and understood, then requires the design phase to be revisited, before the changed requirements can then be built and tested. The repeating of these phases will inevitably increase the cost of the project and delay the timescales before completion.

(iv) **Delivery of functional product is the final stage** – under the waterfall model, the requirements of the entire project must be scoped, then a complete software design produced against these requirements before the software begins to be built. This means that no software is actually being created until the mid-phase of the project, with a fully functional software not in existence until the latter phases. The net result of this is that significant time, resources and money can be spent on a waterfall project with seemingly no tangible or beneficial outputs until the final phase of the project.

(v) **Prolonged delivery timescale**s – linked to the previous point, as the functional software product is not delivered until all of the waterfall phases have been completed, significant time can elapse between the requirement scoping phase and the completion of testing. While this timescale will greatly depend on the size of the project, it is not uncommon in larger waterfall projects for the timescales between requirement scoping to testing completion to take months, if not years. This passage of time can make the requirements out-of-date, meaning the completed software is also out of date and/or no longer suitable for meeting its required purpose when it is finally completed.

3 What is Agile Software Development?

In response to problems that can be encountered with waterfall development, as highlighted in the preceding section, alternative methodologies began to gain popularity with developers throughout the 1990s. These new methodologies focused on "iterative development" with the overarching aim of the rapid creation of working software on a piecemeal basis. Development methodologies focusing on iterative development was not a new concept outside of the software development industry – for example, in the 1950s the US Air Force's project for the creation of the X-15 hypersonic jet used an iterative development methodology.[1]

In 2001 prominent software developers met to agree the main values of agile development. The Manifesto for Agile Software Development was created as follows:[2]

The Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions	over processes and tools
Working software	over comprehensive documentation
Customer collaboration	over contract negotiation
Responding to change	over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

[1] Craig Larman, *Agile and Iterative Development: A Manager's Guide* (Addison-Wesley 2003).
[2] http://www.agilealliance.org/the-alliance/the-agile-manifesto/.

The Manifesto for Agile Software Development was also supplemented by twelve principles[3]:

The Twelve Principles of Agile Software

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

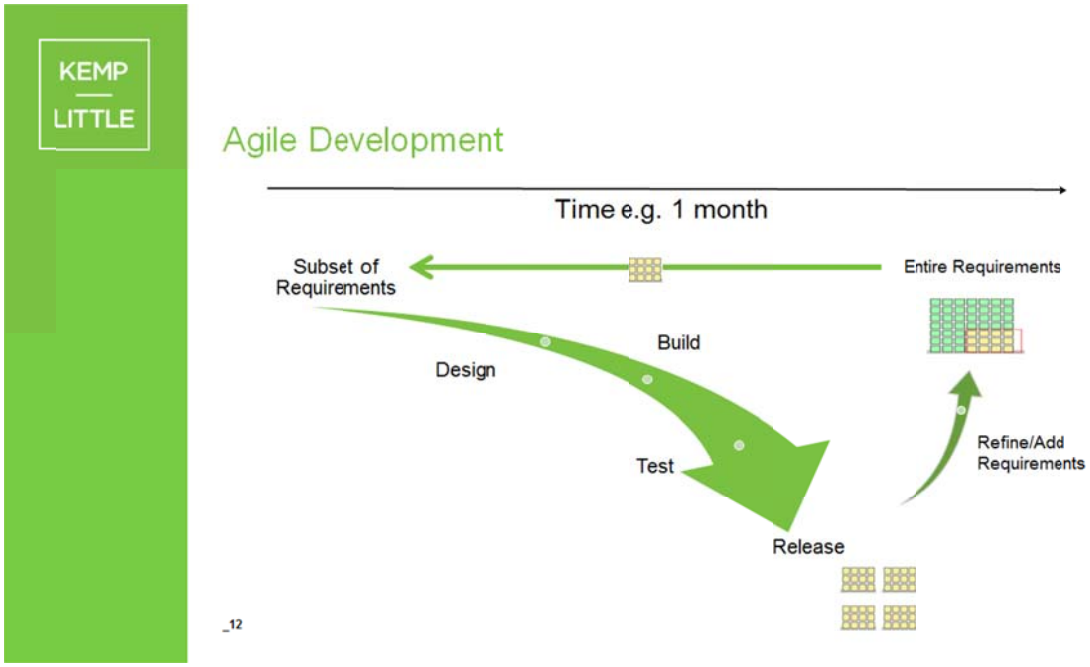
[3] http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/.

Agile software development therefore has a focus on:

- **Iterative development** – these are short bursts of design/build/test development activities, lasting no more than a few weeks, sometimes shorter;
- **Functional software** – working software is created on a "piece by piece" basis;
- **Accommodating Change** – requirements can be constantly changed, revisited, refined and updated as the project progresses; and
- **Customer involvement** – unlike the waterfall model where, following the requirement gathering stage, the supplier is left to "get on with it", agile development involves much more customer involvement in the day to day decision making process at a detailed level.

Figure 2 (below) sets out an overview of a typical agile software development cycle. Starting in the top right of this diagram, the development team will take a sub-set of the overall project requirements. This subset will go through a design/build/test process over the space of a number of days/weeks (top left of the diagram to bottom right). At the end of this period, working software will be produced which meets the subset of requirements. The overall project requirements will then be added to/refined (if needed) on the basis of the experience of that development iteration. The process will then keep repeating until either all requirements are completed, certain minimum functionality has been achieved or the project has completed a number of pre-agreed cycles of iterative development.

Figure 2: Agile Software Development



It is important to note that there is no single "agile software development methodology". "Agile" is an umbrella term that applies to iterative software development that broadly operates as per figure 2. Each agile development methodology has slight differences in both approach, terminology and project emphasis. The most common types of agile methodologies are:

- Scrum;
- Lean;
- Extreme Programming (XP);
- Crystal;
- Feature Driven Development Method (FDD); and
- Dynamic Systems Development Method (DSDM).

4 What is Scrum?

Scrum is one of the most popular agile development methodologies. Scrum is a lightweight framework designed to help small, close-knit teams of people to create complex software products.[4] The key features of the Scrum methodology are as follows:

- **Scrum Team**: A team of people using this methodology are called a "Scrum". Scrums usually consist of 7 people (+/- 2 people). For larger projects, "Scrum of Scrums" are employed whereby each member of the primary scrum represents another whole scrum (e.g. 1 scrum of 7 representatives, representing 7 scrums = 49 team members).
- **Project Roles**: a Scrum project has the following roles:

— **ScrumMaster**:

- Is the "shepherd"/supervisor of the development team
- Is a Problem solver
- Not the boss of the team
- Is a Project Manager
- Common for this to be a **Supplier role**

— **Product Owner**:

- Is responsible for the overall aim and goal of the project
- Represents the interests of the business
- "Owns" the product backlog
- Prioritises the items on the product backlog for selection onto sprint backlogs
- Creates acceptance criteria for the user stories (see below)
- Common for this to be a Customer role
- Product Owner role is a key difference to the waterfall model – transfer of risk/expertise requirement from supplier to customer.

— **Scrum Team Members**:

- Have complete authority as to how the work gets done
- Have complete authority as to which tools and techniques to use

[4] Chris Sims and Hillary Louise Johnson, *Scrum: A Breathtakingly Brief and Agile Introduction* (Dymaxicom 2012).

- Have an area of specialism required for the project, but may be required to work outside this - “doing the job” not “doing my job”
- Self-organise to get the work finished by the relevant deadline
- Are responsible for completing user stories
- Common for this to be a Supplier role, but a customer may provide team members from its own in-house team
- **Sprints**: The scrum team works together in short bursts of development activity called “sprints”. Sprints are fixed periods of time to “design, build and test” a portion of the overall project. These usually last between 2-6 weeks, with the goal of delivering a “potentially shippable product” upon the completion of each sprint. A scrum project can have any number of sprints which are agreed between supplier and customer, but these typically tend to be between 6-10 in number.
- **Documentation**: The scrum development project is tied to three key documents:

— **Product Vision** – The overall high level description of what needs to be developed, focussing on business goals and targeted benefits, rather than technical solutions or specifications. It is typically a static document.

— **Product Backlog** – The Product Backlog is the detailed project “to do list” for all software features needed to achieve the Product Vision. This covers everything the project could hope to achieve. The Product Backlog is a fluid document, and it can and will change as the project develops. Each item of the Product Backlog is called a “User Story”. All user stories on the Product Backlog are ordered by importance – the ones at the top being the most important/well defined, the ones at the bottom being less important/not well defined. Following the completion of each Sprint, the number of User Stories left of the Product Backlog will reduce as the project progresses (see Figure 4 below). The entire Product Backlog for a Scrum project is not always completed in its entirety leaving the lower priority user stories either for a future project or not developed at all as ultimately not regarded as priority items.

— **Sprint Backlog** – The Sprint Backlog is the scrum’s “to do list” for an individual sprint. Once a User Story is moved from the Product Backlog to the Sprint Backlog, the scrum team is committing to deliver it over the course of the sprint.

- **User Stories**: Each item of the Product Backlog is called a “User Story”. User stories are short, “outcome based” descriptions of functionality (see Figure 3 below). They do not describe “how” such functionality is to be achieved technically. Each User story will also have an estimate of the amount of work required to deliver the story (called “points”). Each User

Story is comprised of many individual “Tasks”. The higher number of Tasks that a User Story has, the higher number of points it will have attributed to it. Each sprint will have a limited number of points that can be delivered during that sprint (linked to how much development time the scrum will have in a given sprint), which in turns limits the combination of user stories which can be selected for a particular sprint.

Figure 3: Examples of User Stories

Examples of User Stories

Each user story will typically follow the following format:

As a <type of user> I want to <do something> so that <some value is created>

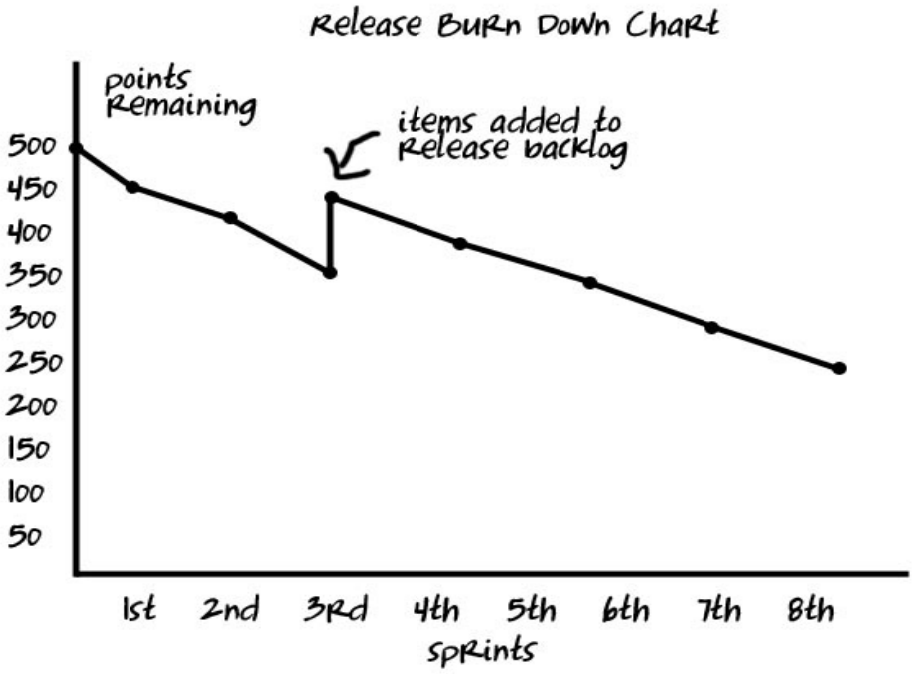
Example 1:

As a user closing the application, I want to be prompted to save so that I do not lose any changes to my data since my last save.

Example 2:

As an account owner, I want to be able to check my account balance on the website so that I can monitor my account without going in branch or to an ATM.

Figure 4: Release Burn Down Chart[5]



- **Testing:** the end of each sprint typically involves a User Acceptance Testing phase or UAT, whereby the customer determines whether the delivered software meets the user stories set out in the Sprint Backlog for the relevant sprint.

[5] Sims, note 4 above, p 21.

5 Do Traditional Waterfall Software Development Contracts Work for Agile Development Projects?

As will be evident from the above description of agile methodologies, the flexibility that is inherent in these forms of methodologies is radically different from the sequential phased approach of waterfall development in a number of ways including:

- **Development Lifecycle** – the waterfall model proceeds on the basis of a single progression through requirement scoping, design, build then test. The agile model differs from this in that these phases are repeated multiple times throughout the project.
- **Flexibility of Requirements** – once requirements have been scoped as part of a waterfall development project, these are typically contractually binding upon a supplier to deliver software that meets these requirements by a given date, with any subsequent changes to these requirements by a customer subject a change control procedure that will likely increase timelines and costs in order to accommodate such changes. Agile development projects on the other hand allow the requirements set out in the Product Backlog to change at any time during the project, with the supplier typically only being contractually committed to deliver certain requirements set out in the Product Backlog when these are committed to a Sprint Backlog.
- **Customer/Supplier Responsibility** – the waterfall model places all risk on requirement development, software design, build decisions and testing firmly with the supplier. The agile model requires far more participation by the customer with certain key decisions such as requirement development, requirement prioritisation and testing being undertaken by the customer, not the supplier.

In light of these key differences, approaching an agile development contract using contractual terms for a waterfall methodology will not be appropriate for all aspects of the contract. The following are a number of issues which should be considered as to why traditional waterfall clauses might not always be suitable for an agile project:

5.1 Commitment to Deliver Software Satisfying Customer Requirements

The Waterfall approach:

Under a waterfall development contract, the customer will typically require the supplier to contractually commit that the delivered contract will meet the Customer's requirements set out in the contract by a given date, e.g.:

3.1 The Supplier shall perform the Development Services with reasonable diligence and despatch, and with reasonable skill and expertise, to provide the Developed Software to meet the Customer Requirements Specification set out in Schedule 1 by the Completion Date.

Agile development issues with that approach:

As described above, a project using an agile methodology will not necessarily know what the totality of the customer requirements are at the commencement of the project, and these will change as the project develops. Additionally, the Product Backlog may set out a number of user stories which the customer ultimately declines to allocate to a Sprint Backlog for delivery during the available sprints during the project. Therefore a contractual obligation to (1) deliver requirements agreed at the date of contract; and (2) deliver the totality of all requirements agreed at the date of contract, will both be incorrect for the agile methodology.

Additionally, the concept of a "Completion Date" needs careful consideration in the context of an agile contract. As noted, a supplier will not be committing to deliver the entirety of a product backlog by a given date, therefore what are the parties agreeing must occur before this "Completion Date"?

Change required to accommodate the agile methodology:

Delivery commitments under an agile contract should be tied to each of the agreed Sprint Backlogs. As described above, user stories are allocated effort "points" and a supplier will allow the customer to allocate a certain number of user stories of a maximum total of points to any individual sprint. If a concept of a Completion Date is relevant, this should therefore be framed as a commitment by the supplier to successfully deliver user stories totalling an agreed number of points by a certain date.

5.2 Due Diligence

The Waterfall approach:

Customers may require a Supplier to provide some sort of due diligence commitment within a waterfall contract that it has analysed the Customer"s requirements set out in the contract and contractually agrees that it is able to provide software that meets these requirements for the agreed charges by the agreed delivery date, e.g.:

4.3 The Supplier warrants and represents that it has had an opportunity to carry out a thorough due diligence exercise in relation to the Customer Requirements Specification set out in Schedule 1 and has asked the Customer all the questions it considers to be relevant for the purpose of establishing whether it is able to provide the Development Services in accordance with the terms of this Agreement, including but not limited to the performance of such Development Services for the Charges set out in Schedule 2 by the Completion Date.

Agile development issues with that approach:

This warranty can't be made at the point of contract. As the Customer's requirements are either not known at the date of contract, or are subject to inevitable change, this due diligence concept is not correct.

Change required to accommodate the agile methodology:

Pre-contract due diligence of this type is not relevant for an agile project. Having said that, due diligence type commitments can be more targeted around the requirements set out in the sprint backlog.

5.3 Acceptance Criteria

The Waterfall approach:

It is common in waterfall contracts to contain contractually binding acceptance criteria (either included within the contract, or to follow following signature), that demonstrates that the developed software meets the customer's requirements, e.g.:

7.5 No later than 30 days from the date of signing this agreement, the Customer shall deliver to the Supplier proposed user acceptance criteria and test data for the Acceptance Tests for Developed Software. These criteria and data shall be such as are reasonably required to show that each module complies with the relevant parts of the Customer Requirements Specification.

Agile development issues with that approach:

In an agile project, acceptance criteria is agreed by the project team in respect of each individual user story. As acceptance criteria is set at this level of granularity, and also as the user stories are not fixed/in existence at the date of contract, it is not possible to either include acceptance criteria for the whole project either within the contract, or to follow within a certain number of days.

Change required to accommodate the agile methodology:

As explained above, in stark contrast to the waterfall model, the customer is typically responsible for selecting the specific functionality that the project team is to develop and what acceptance criteria needs to be set against the same. Acceptance criteria will therefore be created, and tested against, on an iterative basis for each sprint. This also raises the issue that while the output from each sprint may pass acceptance testing, the combined product may not work together as intended, or have issues on the macro level which are not evident from testing at the micro level. Customers are at risk therefore of bearing this component integration risk unless user stories and related acceptance criteria are being carefully drafted to check successful integration with previously delivered output. An alternative option is to provide in the contract for a further round of acceptance testing following the completion of the final sprint.

5.4 Legal Acceptance of Software

The Waterfall approach:

The point at which software is legally accepted by a customer is usually a prominent feature of a waterfall contract. The principle behind this is that a customer should be able to reject defective software and require a refund of charges where the development has not met its requirements set out in the contract. Once software is accepted, the right to reject is lost, leaving the customer to pursue a more lengthy/difficult damages claim for breach of contract in respect of problems arising post acceptance. A typical approach to legal acceptance in a waterfall contract is that acceptance will occur on the earlier of customer sign off or the customer commencing normal business use of the software, e.g.:

> 8.1 Acceptance of the Developed Software shall be deemed to have occurred on whichever is the earliest of:
>
> (a) the signing by the Customer of an Acceptance Certificate for the Supplier Software following successful completion of the testing under clause 7.5; or
>
> (b) the use of the Developed Software by the Customer in the normal course of its business.

Agile development issues with that approach:

As agile is focussed on producing functional software iteratively, there will not be a single delivery of a final product which is then subject to acceptance testing. This means that a contractual mechanism, such as the one highlighted above which assumes single delivery/testing of software, is not correct for this type of development methodology. Additionally, from the customer's perspective, the inclusion of a provision that developed software is accepted in its entirety upon using the same in

the normal course of business needs to take account of the fact that the output of each sprint may be used immediately, but using the same should not impact on the customer's rights or remedies in relation to the output from other sprints. Furthermore, from the supplier's perspective, it will also want acceptance provisions to take account of the concept of accepting separate output from separate sprints to avoid a scenario where failed acceptance from one sprint does not automatically allow the customer to reject the output from other successful sprints.

Change required to accommodate the agile methodology:

The concept of acceptance needs to be adapted to recognise that overall software will be delivered in segments over the duration of the development contract, and each segment will be tested and potentially put to live use separately. It is a matter of negotiation between the parties as to what, if any, impact failed acceptance of one sprint will have on the others.

5.5 Termination for Failed Acceptance Testing

The Waterfall approach:

Waterfall contracts will often contain a provision providing that in the event acceptance tests are failed a certain number of times or acceptance tests are not passed within a certain timescale, then the customer may terminate the development contract, e.g.:

> 7.6 If the Supplier
>
> (a) is unable to correct defects within a period of two months from the commencement of Acceptance Tests under clause 7.2; or
>
> (b) the Supplier fails Acceptance Tests on more than three (3) occasions,
>
> then the Customer may at its sole option reject the Developed Software as not being in conformity with the Agreement, in which event the Customer may also terminate this agreement for material breach pursuant to clause 24.4(d).

Agile development issues with that approach:

In a similar vein to the comments above on legal acceptance, these type of termination provisions are not geared towards a project that has multiple instances of testing/acceptance following each sprint. As further described above (see Section 5.4), each user story within a sprint is typically subject to its own acceptance testing process and in light of this, these type of waterfall provisions used in an agile development contract need to be carefully drafted to ensure they are not applying at the original user story level – the termination of an entire contract based on the failure of 3 user stories across any number of sprints will not be what either party intended.

Change required to accommodate the agile methodology:

This does however raise the question of what happens if acceptance testing is failed whether wholly or partially for a given sprint. From a methodology perspective, failed user stories in this scenario will be simply returned to the product backlog, for the customer's product owner to potentially reselect for a future sprint backlog. To avoid the development contract being the equivalent to a time and materials (T&M) developer contract, a customer will need to address how failed user stories are to be dealt with and where termination thresholds should arise in relation to these. A commitment by a supplier to successfully deliver a pre-agreed number of points in each sprint and across multiple sprints is often used as a threshold, the failure of which can trigger further sprint(s) being performed at no additional customer cost and/or termination rights.

5.6 Change Control Procedure

The Waterfall approach:

A change to the requirements of a waterfall project can have significant impact on charges and timescales as this type of change, particularly in the more advanced stages of the project, may require earlier phases "up the waterfall" to be repeated/reworked. It is common therefore for requirement changes to be subject to a formal change control procedure, e.g.:

> 16.9 The Change Control Procedure will apply to all changes requested by the Customer to the Customer Requirements Specification. Within seven working days … the Supplier shall … prepare for the Customer a written quote of any increase or decrease in the Charges, and of any effect that such change would have on the Completion Date. No change to the Customer Requirements Specification shall be effective until a Change Control Authorisation Note is signed by both parties.

Agile development issues with that approach:

A contract for agile development needs to make a distinction between those changes which need to be subject to a formal change control procedure, and those that do not. For example, the product backlog is not a static document and the user stories set out on the product backlog can change and evolve as a project develops.

Change required to accommodate the agile methodology:

Changes to the product backlog should nonetheless be subject to pre-agreed governance arrangements setting out how the product owner (i.e. the customer) and the scrum master (i.e. the supplier) will agree additions/removals/refinements to the product backlog (including how points will be assigned to each user story), however this is unlikely to be subject to a paper based formal change control procedure.

Certain aspects of the contract, such as number of sprints, charges etc., should be subject to a formal change control procedure.

5.7 Prohibition against the use of Open Source Licensed Software

The Waterfall approach:

Certain model form waterfall contracts contain prohibitions against the use of freely available open-source licensed software (OSS). The motivation of this can stem from a variety of legal and organisational reasons[6], but is often premised on the fact that where a customer is paying for bespoke development work, it does not expect OSS to be used unless it has expressly agreed otherwise. Such a restriction can take the form of:

> Supplier warrants and represents that there has not been included or used any Open-Source Software … in the development of the Developed Software nor does any Developed Software operate in such a way that it is compiled with or linked to Open-Source Software.

Agile development issues with that approach:

As highlighted in the above descriptions of the product vision and individual user stories on the product/sprint backlogs, agile methodologies focus on "outcomes based" technology neutral requirements and allow the development teams to determine the quickest and most efficient means of achieving these outcomes. Quite often OSS will play a key role in achieving certain user stories where pre-existing code can be leveraged to reduce the amount of work required to deliver the relevant user story. While suppliers will typically work to whatever limitations on OSS a customer may require, this may have a fundamental impact on the number of points that are assigned to user stories if OSS restrictions require more bespoke coding to deliver the same result. In other words, restricting OSS use can make agile development more expensive and slower.

Change required to accommodate the agile methodology:

Customers should consider carefully what, if any, restrictions it wishes to place on the use of OSS and the reasons for doing so. For example, if a customer is planning on distributing the completed software to third parties, there may be legitimate licensing concerns with certain restrictive OSS licences (such as the GPL family of licences[7]). Conversely, if the intended use case of the completed software does not raise significant OSS concerns, a customer should weigh up the cost impact of restricting OSS usage. Regardless of the approach, the use of OSS should be managed as part

[6] See Kemp Little white paper "Open Source Software: Freedoms, Responsibilities and Governance", June 2014.
[7] See Sections 4 & 5 of Kemp Little white paper "Open Source Software: Freedoms, Responsibilities and Governance", June 2014.

of an OSS governance framework that the development team should comply with as the project progresses. This may take the form of logging what OSS is used and where, what OSS licensing terms are applicable and, where required, either prohibit the use of OSS that is subject to certain OSS licences or require prior written approval from the customer before using such OSS.

6 Conclusion

In many development scenarios, following an agile methodology is likely to offer numerous advantages over following a waterfall methodology. It does however mean that there are other risks and issues which the parties should consider to make conscious allocations of risk. Using an agile methodology will mean that the contractual terms governing such an arrangement should differ from those traditionally used in waterfall development agreements – which form the basis for most precedents used. From a customer-side lawyer's perspective agile developments are often treated warily – perceived as being light on certainty and supplier commitment. However, with appropriate drafting these agreements can meet the twin goals of capturing the benefits of an agile development with securing the appropriate contractual protections for both parties.

Andrew Joint, Partner, Kemp Little LLP

Edwin Baker, Senior Associate, Kemp Little LLP