systems@work Implementation

- Agile Implementation Method
- Agile Implementation and Estimation
- Skeleton Project Plan

Agile Implementation Method

Software development methods are hotly debated, and perhaps none more so than 'Agile' methods, which reform the classical, pedestrian phased approach consisting of a lengthy analysis phase, followed by a lengthy specification phase, and further sequentially organized and separate phases. This classical approach reliably delivers functionality that the project team *thought* was required several months or years earlier. Whilst this method can work well in a static environment where the requirements are vast, and hard to determine without exhaustive investigation, it is an inappropriate method in medium-sized organisations. At the outset of projects, when both requirements and software solutions are theoretical, it is almost pointless to obtain a customer's confirmation of a specification if he hasn't a clear idea of what the solution will 'look like' and how it will work 'in practice'.

The Agile software development method replaces the classical sequential approach with an iterative and collaborative process. Its principles include:

- Building frequent working versions
- Working closely with sponsors and users to determine a clear understanding of requirements and the suitability of solutions
- Collaborating closely with sponsors and users on workarounds, simplifications, and compromises
- Documenting requirements and specifications in tandem with iterative software versions (rather than always *prior*)

These Agile methods stemmed from frustration with an approach, which, though well defined, had a tendency to deliver something other than what was needed by the user (even when he had accepted the developer's specification). Agile methods are useful both for the development of specific ('bespoke') systems and for the development of software packages such as systems@work. Indeed, we use this method for the development of time@work and expense@work.

What do the Agile Development methods tell us about implementation methods?

To answer this question, we need to look at the way in which software packages are designed.

Software packages come in a variety of ‘styles’, two of the most identifiable styles being the *Adaptable* and the *Configurable*:

- In the case of the *Adaptable* style a solution is achieved through *programmed adaptation* of a system’s predefined functionality, usually through modification, and extension of the source code delivered with the package. One might call this ‘fast-track’ bespoke development, since the package provides something of a leap forward from nothing, even if it lands the customer only half way down the track. In this context implementation is part of the development process. Adaptable systems are not so easy to upgrade, but they can seem to deliver, eventually, ‘exactly’ what is understood to be required.

- In the case of the *Configurable* style a solution is achieved through the configuration of parameters *without underlying software modification*. Authors of *Configurable* packages don’t release the system’s source code, but provide a variety of interface techniques to overcome the issue of integration. This is also ‘fast-track’ development, but even if the customer lands only half way down the track, he continues in the same vehicle rather than an adapted one. In this context, implementation is a separate process from software development. These kinds of system are much easier to upgrade. systems@work’s time@work and expense@work fall into this category.

In general the iterative approach to software implementation (an approach one might call *Agile Implementation*), even though it was initially developed for the world of the Adaptable or bespoke system, is in reality **most** cost effective for *Configurable* systems, since they can more quickly be configured in a variety of ways (by a single consultant, rather than through the lengthier teamwork of consultant and developer) to demonstrate exactly how a solution can meet a requirement. Iterative system prototypes can rapidly be developed, and workshop sessions can rapidly be convened to try out alternative solutions. During the crucial system building phase, as long as customers and consultants are working closely together, requirements can be refined, solutions developed, consensus achieved and the system documented, to some extent ‘as we go along’. This may take ‘agile’ project management, but it is far more likely to create a workable solution than a method which involves detailed initial analysis, ‘theoretical’ confirmation, and then distant ‘laboratory’ development.

systems@work advocates this Agile Implementation method.

In practice this means breaking up a project into the following steps:

INITIATION

1. Team Development and Project Initiation

The objective is to assemble a team for the implementation, to define responsibilities and reporting obligations, to define project deliverables, priorities and an overall project plan.

The following must be identified:

Client	*Implementer*
Project Sponsors	Account Manager
Project Manager	Project Manager
Representative 'Clients' who define the requirements of the project. These will include representatives from: Management (who define key performance indicators, etc.) Finance (who define accounting issues) Project Managers (who resource, and monitor projects) Staff who define the commercial issues surrounding contracts	Business Analyst
System Administrators who will work on the configuration of the system and who will subsequently administer it	Implementation Consultant
IT Technical Specialists who will be responsible for hardware/software/server/database/ communications configuration	Technical Consultant

2. Initial Installation

The objective is to install time@work so that technical staff become familiar with installation issues and to enable training on the client's equipment.

3. Initial Training

The objective is to introduce the functionality of time@work and the way it is configured without detailed reference at this stage to the particular requirements of the client.

The training will be focused on System Administrators from the project team, though others may attend.

The training will take the form of a prototypical development of a working system from scratch.

Day 1	Timesheet and Expense Form Configuration Reporting
Day 2	Professional Services Workbench Customer Services Workbench Invoicing Accounting Issues and Interface
Day 3	Budgeting MS Project Interface Resource Scheduling Data Import Database Administration

ANALYSIS, DESIGN AND BUILD

4. Analysis Workshop

Typically a one or two day session led by a systems@work consultant, with sponsors and key users, which delivers an outline of requirements and scope, and enables the building of an illustrative prototype and an outline specification (requirements and outline solution). An outline project plan and change control procedures may also be developed during this phase.

The analysis phase will determine the configuration of and the processes surrounding the use of time@work.

All current documents (timesheets, expense forms, invoices, reports, etc.) should be available to the project team.

The analysis phase of the project will deliver an initial definition of requirements.

The project team and projects sponsors will sign-off this document.

5. System Design Specification

The system design phase will describe how time@work together with surrounding procedures will deliver the agreed requirements.

An initial document will describe and explain intended settings of time@work parameters.

The project team and project sponsors will sign off this document though it is expected that modifications to it will be made later.

6. System Building and Prototypes

Once the initial system design is complete, an initial prototype will be built, followed by workshop sessions with sponsors and users that:

- demonstrate the proposed solution (or alternatives) in order to clarify ‘how’ the system will provide a solution and, crucially, ‘what it will actually be like to use’
- identify areas where requirements cannot be met or cannot easily be met, so that compromises and solutions can be agreed and demonstrated

During this phase, which is iterative, and subject to careful change control (identifying risks, costs, etc.), requirements and design decisions may change. Documentation must be kept up to date with each prototype.

IN TERMS OF OVERALL SUCCESS, THIS PHASE IS THE MOST CRITICAL ONE, AND THE ONE WHERE THE ‘AGILE’ METHOD IS MOST IN EVIDENCE

Note that prototyping does not take the place of formal acceptance testing with key users.

Any modifications to requirements or design specification documents following formal revision should be signed off by the project team and project sponsors.

7. Procedures

The project team will develop a document describing all procedures and responsibilities surrounding the operation of time@work. This will cover at least the following:

Standard procedures for static data maintenance

New employees
Leaving employees
New clients
New projects and tasks
New fee rates
Closing tasks, projects and clients
Exchange rates

Project development procedures

Developing a project plan
Allocating employees to a project
Entering recurring invoices, if appropriate

Occasional procedures for static data maintenance

New calendars

Planning

Entering a forecast

Timesheets

Preparing timesheets
Monitoring timesheet completion
Authorising timesheets
Monitoring timesheet authorization
Approving timesheets
Modifying timesheet data
Confirming fee values

Expenses

Authorising expense forms
Reviewing expense forms
Monitoring expense form workflow
Entering invoiced expenses
Approving expenses (project view) in respect of recharging
Modifying expense data

Invoicing

Proposing invoices
Approving invoices
Reviewing invoices
Reversing invoices
Reprinting invoices
Ad-hoc credit notes

Reporting

Standard reporting packages

Database Administration

Purging data
Archiving data

8. Test Plan

The test plan defines a representative sample of data and specific tests for all input and output procedures, reports and data transfers.

The test plan must also cover any one-off transfer procedures used in transition.

9. Transition Plan

The transition plan defines how data from current systems will be transferred to time@work and exactly when this will occur. An exact sequence for suspending current systems and procedures and adopting new ones is required. Where large amounts of data are involved testing must prove that the proposed timescales are achievable.

Consideration must be given to fallback plans if transition fails for any reason, including alternative go-live dates.

Key data are:

Static Data:

Employees
Clients
Projects
Tasks
Analysis Values (Clients, Projects, Employees, Activities)
Value Tables (Fees, Costs, Charges, etc.)
Cross Rates

Transactions:

Work in Progress (Time and Expenses)
Current Allocations
Budgets and Forecasts
Historical Data

COMPLETION

10. Support Plan.

A full support plan must be developed to support users during the initial intensive period after transition to live use, and in subsequent periods.

11. Testing

Testing must be performed carefully in a test database and results signed off by those who carry out the tests.

When corrections or changes are applied the testing group must carefully assess the necessity to back track on completed tests.

12. User Acceptance Testing

time@work will already have been shown to representatives of each group of users during system building and prototyping. User acceptance testing is a final confirmation of the agreed configuration and procedures. Representative users must sign off on the system.

The situation whereby changes are requested and made during and/or after formal training, must be avoided.

13. Training

Training must be carried out on representative data (and using the procedure manual) just prior to live use of time@work. Early training is rarely remembered if the system is not used in the intervening period.

Training in support procedures must also be given.

14. Transition

Final data transition and go-live. Fallback plans for transition failure must be in place.

15. Review and Support

Review points during the early stages of live use and full project review some weeks after live use.

Agile Implementation and Estimation

Estimation of the time required of both consultants and clients during systems implementation projects is notoriously difficult. The Agile iterative approach would appear to exacerbate this situation since it suggests an undefined number of iterations and allows change at any stage of the project. The strictly sequential approach would appear to allow a more disciplined approach to estimation, even if estimates for a particular phase of a project may be precise only when preceding phases are completed.

But this is illusory. What happens in practice with the strictly sequential method is that either the wrong system is delivered or change control procedures are developed to allow extensions to the project when it is discovered (often belatedly) that change is required.

In fact, if changes are necessary in a project it will always be best to identify this at the earliest possible stage, and this is far more likely to occur earlier if an iterative approach is adopted involving frequent reference to key users and sponsors.

However, this does not solve the problem of estimation. In practice it is possible only to allow ample contingency in a project, especially during the analysis, design and build phases. It probably makes sense to estimate how long this would take if everything were right first time and then add 25%.

Project Plan

See MS Project Plan *time@work Implementation*