Reflections on applying iterative and incremental software development methodologies (Agile, RAD etc.) to aid and development work in developing countries.

Matt Haikin, *March 2013*
www.matthaikin.com

4,844 words

Recently I have been reading up on Agile project management methodologies (Extreme Programming, Scrum and a little on Rapid Application Development, EVO and Rational Unified Process). Despite this material being focused on traditional, commercial software development and management, it struck many, quite noisy chords regarding technology development in developing countries. In particular, the focus on starting small, not pre-planning everything from the start, and evolving software slowly through engagement with the 'customer', is strikingly similar to the practices recommended in various participatory approaches to development, and in socio-technical discussions around ICT4D projects.

With this in mind, I thought it would be interesting to explore these similarities and see what Agile software-development methodologies might have to offer the ICT4D community – not just in terms of developing software but in the wider development context too.

Note : This piece is not intended to be a robust analysis of the available evidence but more a think-piece that may provide some food for thought to investigate further at a later date. It's too long and structured to be a simple blog but not rigorous enough to be an academic article, but falls somewhere between the two media. So until I find a better term – please find below my first ***Blarticle****…* ☺

1. Introduction

I have in the past discussed some of the reasons for the "widespread failure of ICT4D projects" (Haikin, 2012). Without re-visiting this discussion, there appears to be a relatively clear consensus that some of the most significant reasons include a lack of engagement with beneficiaries and local communities, a tendency towards top-down delivery, techno-centrism and an over reliance on pre-planned engineering and blueprint approaches to delivery (Chapman & Slaymaker, 2002; Dodson, Sterling, & Bennett, 2012; Hamel, 2010; Heeks, 2010; Rozendal, 2003; Schech, 2002; Thompson, 2008; Walton & Heeks, 2011)

Many of these factors have clear parallels with the "widespread failure of large software projects" (Larman, 2004) seen in the mainstream commercial software sector. Agile and Iterative and Incremental Development (IID) approaches were a response to this failure and it seems instructive to explore whether or not the tools, methods and techniques adopted by those following these approaches might also have a role to play in improving the results and sustainability of ICT4D projects.

The following sections draw out some over-arching values and core practices from a range of Agile and IID approaches and explore how these may suit the circumstances surrounding developing software and other technology for aid/development goals, in a developing country context.

2. Iterative and incremental software development methodologies (IID)

Although they have relatively recently become an accepted part of mainstream commercial software development, IID approaches are not new phenomena. They date back at least to the 60s and 70s when EVO and RAD amongst other techniques first started to be used (Larman, 2004). Despite this, the 'Waterfall' model dominated until relatively recently. This is unfortunate, and ironic, considering that there is evidence that those credited with 'inventing' it actually never intended the rigid top-down and pre-planned model that it has become) (Larman, 2004).

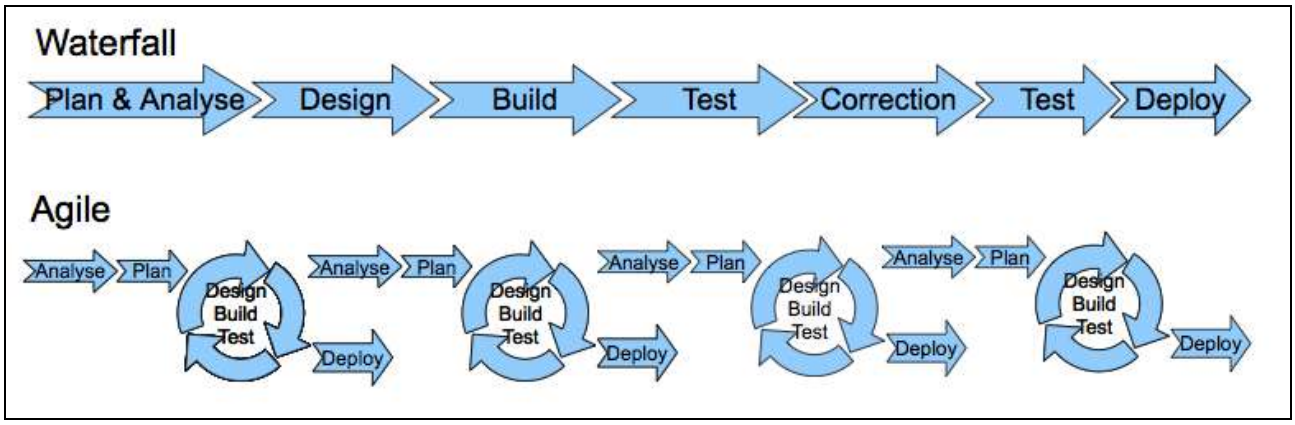

Figure 1. Waterfall & Agile Models *(image adapted from one by Amit Gupta, courtesy of article-stack.com)*

Agile as a specific Manifesto and defined set of principles outlined by the Agile Alliance emerged around the end of the 20th and beginning of the 21st Century, as did Extreme Programming, Scrum, the Unified Process, later followed by IBM's Rational Unified Process (Beck, 2000; Larman, 2004; Matuszek, 2008; Wells, 2009a, 2009b).

This range of IID approaches developed their popularity as a direct challenge to the top-down engineering approaches more common in software development then and now. They promote a more flexible way of working that *evolves* software of higher quality and that does a better job of meeting the *actual* requirements of the customer – software *development* for complex social / socio-technical problems, rather than software *engineering* for known, predictable problems.

Why Agile / IID..?

While there are, of course, distinctions between the different approaches and methodologies mentioned above, at their core, they all revolve around the idea of evolving a piece of software in short iterations, adjusting the requirements as the software is developed and these early versions are used to help the business and developers understand the true requirements better.

The Agile Manifesto

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

Agile project management principles and practices

Deliver what the client values
Cultivate committed stakeholders
Leadership-collaboration style
Build competent collaborative teams
Enable team decision-making
Use short time boxed iterations
Encourage adaptability
Champion technical excellence
Focus on delivery not process/compliance
Establish and continually reinforce a guiding vision
Facilitate collaboration
Establish and support team's rules and practices
Visible and open access to project information
Light-touch 'just enough' management to foster self-direction in teams

Extreme Programming (XP)

Values

Communication
Simplicity
Feedback
Courage

Principles

Rapid feedback
Assume simplicity
Incremental change
Embracing change
Quality work

Core Practices

The Planning Game
Small, frequent releases
System metaphors
Simple design
Testing
Frequent re-factoring
Pair programming
Collective code ownership
Continuous integration
Sustainable pace (no overtime)
Whole team and customer together
Coding standards

Agile core principles

Satisfy customer through early and continuous release of software
Welcome changes in requirements, even late in development
Deliver working software frequently
Business and development work together daily
Support and trust motivated individuals to get the job done
Face-to-face communication is most effective and efficient
Working software is the primary measure of progress
Development should be sustainable
It should be possible to maintain a constant pace indefinitely
Continuous attention to technical excellence and good design
Simplicity is essential
The best designs emerge from self-organising teams
Teams should reflect regularly and adjust to become more effective

Rapid Application Development (RAD) - Key Features

Iterative development
Incremental prototyping
Time-boxing
Use of time-saving development tools
Re-usable code/templates
Strong emphasis on user participation"

Scrum

Self-directed and self-organising team
No external additions of work mid-iteration
Daily stand-up meeting
Demo to external stakeholders at end of each iteration
Client-driven iteration planning

EVO

Short iterations
Evolutionary design vs. evolutionary delivery
Client-driven or value-driven planning
Quantifiable measurement of progress and value
Numeric definition of quality requirements

(Rational) Unified Process

Short time-boxed iterations
Develop high-risk and high-value elements early
Re-use existing components
Deliver value to customer
Accommodate change early
Work together as one team

A summary of the different values, practices etc. of a range of Agile and Iterative approaches can be seen in the tables on the previous page (various sources). For simplicity, some important common factors from this collection that cut across all Agile and IID approaches are outlined below:

- The importance of the people and the autonomy of the team over the process
- Incremental delivery of working software in relatively short iterations (usually 1-4 weeks, although for larger projects these could be a few months)
- Responsiveness to change – ranging from an aversion to defining requirements too early without adequate feedback, through to positively embracing change at any and every stage of the development process
- The 'customer' as a critical part of the delivery team – this is a vital counter-balance to the lack of detailed up-front requirements
- All software simple, yet technically excellent

The rationale for these approaches is based on a set of key findings about reasons behind software failure which are discussed below (Beck, 2000; Bell & Wood-Harper, 1998; Beynon-Davies, Carne, Mackay, & Tudhope, 1999; Govt. of Hong Kong, 2008; Heeks, 2002, 2008; Larman, 2004):

- **Large, complex projects are more likely to fail**

  The chances of failure of software projects increases with their size – most likely due to the increase in complexity and difficulty of predicting how complex systems will behave. By splitting large projects into smaller, manageable iterations (each one effectively a mini project of its own), Agile seeks to simplify this complexity in an attempt to reduce the chances of failure.

  It seems likely that this is one of the contributory factors behind ICT4D failures too, so adopting a more Agile approach could potentially help reduce some of these failures.

- **Requirements are usually unclear at the start of a project**

  Common-sense and experience tell us how difficult it can be to specify a complete set of requirements 12-18 months ahead of the time when they will be needed. This is borne out by the number of projects which, while they meet the *specification*, fail to meet the *real requirements*. By delivering the system and features in increments, Agile allows customers to develop their understanding of their true requirements over time, through experimentation and evolution.

  In aid/development work the requirements are rarely clear when work begins (and sometimes remain hazy throughout!), so any attempt to specify every aspect of a system up-front is clearly doomed – evolving the system as those involved increase their understanding of the situation, needs and potential solutions is likely to increase the chances of these systems achieving their social aims.

- **IKIWISI – "I'll know it when I see it"**

  Many people simply do not or cannot think in terms of requirements documentation but need to see, use and play with a real system in order to tease out what they really think or want. This is not possible in a waterfall model (unless expensive throwaway prototypes are made), but in an Agile development, customers/users get to use the system from a very early stage to help formulate their true requirements. Even customers who *do* think in this way may change their minds, *"It's just what we asked for, but it's not what we want"*. This may sound like a smart anecdote personal experience can attest - the problem is extremely common!

  In a development context, it is common that some of the key stakeholders are people with little familiarity with technology or software development. This means the chance of them being able to specify requirements up-front is even lower*, therefore the chances of them changing their minds is* even higher*, and so the need to see a real, working system early on to tease out further requirements becomes* even more critical*.*

- **Requirements change over time**

  Even in scenarios where it is practical to specify a system in advance, given the length of most software projects, it is likely that the situation, environment and requirements will have changed by the time the system is launched.

  This is especially true in development where the complex interplay of social forces, governance, NGOs, funders, grass-roots activities, politics etc. mean the environment could well change numerous times during a project lifespan. Agile embraces this change and can rapidly accommodate new/changed requirements in the next iteration (usually only a short time away). In a waterfall approach these changes are seen as high risk and it is more likely that the original specification will stay, and a fully working system will be delivered, but one which no longer reflects the needs of the beneficiaries or staff on the ground.

- **Most code written is never used**

  The above factors combine to ensure much of what is coded, while it may work technically, is never used – i.e. working features that are not required, or were never really relevant. The time and cost taken to build these non-required features is large (one study found that 45% of features built are never used!).

  While this is clearly important in every sector, in the finance-strapped world of aid and development, potentially wasting almost 50% of a budget is catastrophic. If an Agile approach can allow for this budget to be spent on more features that ARE needed (or spent in entirely different areas), this is a huge boost to the potential benefit per £ spent.

It seems clear that the values of Agile and IID are entirely compatible with and supportive of the types of problems aid and development work tackle, and that they have a resonance with the participatory and sustainable approaches to development that are becoming more popular, whereas waterfall approaches appear to have more in common with more paternalistic and top-down engineering views of development. The next section goes on to look at some specific features and practices of Agile methodologies to see whether they are as useful as the over-arching values would suggest.

3. How might Agile and Iterative practices apply to aid and development?

The various principles of Agile, RAD, Extreme Programming etc. described on page 4 share some core practices and techniques that can be summarised as follows:

- Short iterations producing working software (evolving requirements each iteration)
- Including the 'customer' in the development team
- Simplicity – especially simple designs and solutions
- Self-directing teams
- Favour face-to-face communication and feedback

Short Iterations producing working software (evolving requirements each iteration)

Short iterations producing working software are of particular use in scenarios where it is likely that people don't know their true requirements until they see and use an early version of the system – something that is highly likely in a development context, where problems are typically complex and socially driven; where some/many of the people aren't familiar with technology or the development-process; where people may need to see and use things repeatedly to understand fully how technology may help meet their needs.

This approach also provides a useful learning opportunity for those involved. By being involved from an early stage, and seeing a simple system evolve over time, peoples' level of technological awareness and understanding can be increased. This not only helps ensure local experts are in a better position to appreciate how technology can help them but enables more useful collaboration with external technology experts leading to a better, shared understanding of potential "latent needs".

In development there is often a difference between local experts who understand the needs on-the-ground, and technology experts who have seen what technology can do or has done in other similar situations. Bringing this expertise together is a powerful combination and one that iterative and incremental development seemingly is able to foster and support.

However, traditionally in Agile approaches there is a push to start coding as soon as possible (especially in Extreme Programming). Given the potential learning and experience-sharing that iterative development can foster, this may need to be re-thought as a longer exploratory and scoping phase (although still measured in days not weeks/months!) both at the start of a project/release, and at the start of each iteration, could maximise the value of the iterative process.

Include 'customer' in development team

Most Agile approaches recommend that a representative of the business/customer be part of the development team – in regular (ideally daily) contact. In fact, Extreme Programming goes as far as to suggest an on-site customer working in the same room as the developers. This is to ensure rapid response to questions and ensure the customer is on-hand to clarify requirements – vital in the absence of detailed up-front requirements.

Commercial development already recognises that in some cases there is not one simple 'customer', but multiple stakeholders, and suggests that a group of customers or advocates for stakeholders may work as an alternative. However, this still works on the assumption that all these customers/stakeholders share the same vision for what the software should do – even if they may not agree on every detail of every feature or requirement.

Software in the development sector does not necessarily fit this mould. It is entirely likely that the 'customer' is a large, disengaged donor, funder or government department, working in partnership with one or more NGOs, who have their own mission to balance with the needs of the specific community they are working within (perhaps the ultimate beneficiaries of the work), which itself comprises a range of different groups with different goals and potential conflicts. In this type of scenario, having one "on-site customer" is an impossibility – even in the unlikely event that the client (i.e. whoever is paying – the donor/funder) could be persuaded to make a staff member available daily, they are simply not able to represent the goals and requirements of all the different groups with a stake in the project.

In this scenario, the need for a more participatory approach emerges – and along with it the murky realm of power structures, group dynamics and the potential for abuse of seemingly fair participatory methods (Cooke & Kothari, 2001; Cornwall, 2003; Haikin, 2012; Kothari, 2001).

To start with, finding a way to ensure fair representation of groups with different goals and needs is vital - it may be possible to have a customer-*group* combining the donor, NGO and local champions for example, but even if this were possible, it is unlikely all of these representatives could be made available on a regular and frequent basis.

It is also entirely likely that their views on the project requirements will differ in not just the detail but in its fundamental vision – this clearly requires a level of sophistication, understanding and facilitation that is far beyond that required in a typical software development requirements workshop (although it is worth noting that in a traditional waterfall project, the most likely output is that the requirements of the donor will be the only ones included, so perhaps even this confused scenario is an improvement!).

It seems that a more engaged, collaborative and participatory journey is required, with a much higher focus on different needs and goals at the start, with technical and feature-level requirements perhaps being dealt with slightly later. It is an interesting challenge to see if an Agile approach can be adapted to this scenario and perhaps combined with elements of participatory design and participatory development to cater for customers that may be less engaged, have multiple and conflicting needs, and need to collaborate with each other and with the development team to make any progress. Likely impacts would be slightly longer iterations, a non-technical 'visioning' phase before any development occurs to try and reconcile conflicting needs, and more scoping/exploration at the start of each iteration to achieve consensus on specific features and requirements.

However, perhaps a caveat is also needed – in situations where, despite expert facilitation of a 'visioning' stage, there remain fundamental differences over the high-level needs and goals of a software project – perhaps the better option is *not* to develop it at all, rather than waste a lot of time and effort producing something nobody agrees on, that in all probability will just exacerbate existing tensions further. In these scenarios perhaps the most valuable thing to be gained from an Agile approach is a clearer understanding of when it is wise to just say No and to revisit the project at a later date if/when more fundamental conflicts have been resolved.

Resilience through simplicity and skill sharing

Extreme Programming recommends "Favour the simplest solution that does the job, design for today not tomorrow" (Beck, 2000). While this may go against the intuitive urge to build in flexibility early on, given what we know about how much code is unused or not needed – it may be a useful mantra.

In a developing-country context it has a significant added benefit. If one of the goals is for a development intervention (and therefore any software related to it) to become sustainable, then at some point it should be hoped or expected that the maintenance and continued development will be handed over to local developers, even if the initial development was undertaken by external experts.

In many situations, it is more likely that local developers will be from an area with poor education, and include those who are not skilled enough to find work elsewhere, and probably have high levels of staff turnover as people find higher paying work elsewhere[1]. In this context, simplicity has the benefit of making for a system that is much easier to learn, understand, maintain and evolve. In fact it could easily be argued that overly 'clever' coding is not only unnecessary but fundamentally a barrier to sustainability.

[1] *This is not to imply that developing countries do not have highly-trained programmers but to suggest that in this context of a typical aid/development project seeking a sustainable outcome, it is less likely that these highly skilled local developers will be available, as they are more likely to be working on more highly pad commercial work. Whether Agile is appropriate for a commercial software company in Bangalore, for example, is not the question (it is exactly as appropriate or not as for a software company in London or San Jose), but the question is whether Agile is appropriate for, for example, a small NGO in Uganda delivering technology/software as part of its work with the local communities, perhaps in partnership with a small local University – a very different question.*

Combining this simplicity with a suite of the other practices that work together gives an interesting insight. There are a number of practices from Agile and specifically from Extreme Programming that relate to the values around 'constant attention to quality and technical excellence' – test-driven development[2], continuous integration[3], shared coding standards and code ownership[4] and pair-programming[5]. These features combine to ensure the code is simple, functional, bug-free and easy to maintain; while also ensuring every member of the team is comfortable working on every aspect of the design. This also creates an 'agile' design that is responsive to change, giving teams the courage to constantly re-factor and re-design code as requirements change.

Looked at in the common context of an external/overseas developer initiating development, but a local resource being expected to make it sustainable, this has enormous additional benefits. Pair programming maximises the opportunity for skill-sharing (in both directions – the external 'expert' increasing his/her understanding of the local context, and the local 'expert' benefiting from the external coders commercial best-practice – which is reinforced through the sharing of coding standards and code ownership), as well as making the team more resilient to team members leaving and changing, as everyone is familiar with the code. Given the likelihood that the coders will be less experienced, the simplicity of the system is vital, and the test-driven approach and continuous integration ensure they have the confidence to make changes and evolve the system in the knowledge that mistakes can be found and fixed quickly and with little or no risk. This makes it far more feasible for local institutions and individuals to appropriate and extend the technology rather than simply viewing it as the legacy of yet another external intervention that they have no control over.

---

[2] *Write unit-tests first then build code to make the tests pass. And at a higher level write acceptance tests first then design/build functionality to ensure these tests work as expected.*

[3] *Instead of a long, complex and risky integration effort just prior to each release, integrate all code daily (or more often) so bugs are found and fixed immediately and the software is always ready for release.*

[4] *Nobody owns classes/areas of code, but everyone works on every aspect of the system and fixed problems as they see them – ensuring no bottlenecks on key classes. This relies on shared coding standards to be possible.*

[5] *All programming is undertaken with two coders sitting at one machine. One codes while the other watches, thinks, designs tests etc. Pairs swap often so both members take turns coding. Pairs also change often so everyone works on every element of the system.*

Self-directing teams

Agile (and in particular Scrum and XP) suggests that teams need to work closely together (in the same room ideally) and be self-directed and self-organised rather than be actively managed from above, and that to do so they need to be (a) motivated to take control and (b) trusted to do so.

This clearly is in line with the core values of participatory development and empowerment, but is not something that seems to be very prevalent or visible in ICT4D or even in development more widely. It would be interesting to extend this practice and combine it with other elements of participatory development (and with Paulo Freire's ideas on *Reflective Learning* too) to see if this can be a vehicle to build the kinds of skills, attitudes, processes and practices needed for local organisations or communities to take a much greater role in technology development for their own needs, increasing opportunities for local appropriation of technology and sustainability.

Favour face-to-face communication and feedback

Agile favours regular verbal communication (e.g. daily stand-up meetings from Scrum) over complex processes and detailed documentation. Given that some of the users / customers / beneficiaries in development projects may not be highly literate or educated, this immediately has added resonance. It also increases opportunities for weaving existing (often verbal) local decision–making processes and structures into the technology development process, binding it closer to the communities in which it will operate.

4. *Developing* new solutions to software *development* for *development* problems in *developing* countries..?

The explorations in the previous section give some interesting food for thought about how Agile and Iterative Incremental Development practices might prove useful in a development context – although as the title of this section shows, perhaps coming up with a less confusing terminology would be a starting point before delving into the detail!

So given the positive thoughts above, what types of development problem are most likely to benefit from an Agile approach? Clearly those where software development are a major part of the project – but this is probably not enough. The benefits of Agile are more compelling for (a) relatively small projects of 10-20 developers maximum, and (b) for the kind of 'wicked' social problems common in development.

Given that *most* development problems are soft/wicked, and much of the related software will be relatively small, this means that most of the development sector could potentially benefit – albeit possibly only in situations where the funder/donor can be persuaded to work in a more flexible manner and relinquish the requirement for rigid log frames defining the entire project and requirements in advance!

In terms of the *added* benefits that seem to emerge from an Agile approach in a development context, it seems that the biggest potential benefits are those scenarios where there is an external team creating the software initially (this could be in-country or overseas, but generally from outside the target area) but there is a goal for local institutions or individuals to take it over, appropriate the technology and sustain it over time. One would hope this describes a majority of ICT4D projects but it is far from clear that this is the case!

So effectively, the kinds of project/scenario where Agile has the most potential benefit, are the same kinds of complex social situations that Participatory Development also sets out to deal with.

Given ICT4D's dual roots in Development and IS/ICT (Dearden & Rizvi, 2008a; Haikin, 2012)- this means there are similar drivers from both the technical side (justifications for Agile) and the social side (around participation for empowerment and sustainability). The values and types of solution are similar from both of these perspectives, so perhaps it would be interesting to combine the practical, tested and technology-focused practices of Agile with the more social-political approaches of development techniques (such as PRA) and see what emerges…

However it is worth remembering that these are just techniques which might help, given the right motivations and under the right circumstances:

> *"Some problems are just hard, some people are just difficult, these methods are not salvation"* (Larman, 2004)

5. What next… further research?

The discussions and reflections above make a clear argument that the role of Agile and other Iterative and Incremental Development techniques is worth considering in a development and ICT4D context. But this is all based on theory, assumptions and common-sense – which may or may not be borne out in reality.

The next area of discussion would involve looking at people or organisations that already are using Agile-influenced approaches in the developing world to tackle the 'wicked' problems of development.

This could shed some light on how important some of the potential issues are in reality – for example does the problem of "there is no single customer, but multiple groups with conflicting goals" appear often, and is it a major difficulty or simply one of many obstacles to overcome with good facilitators. If these kinds of group and power dynamics are as important as they seem – can Agile be adapted or combined with other techniques to tackle them more effectively?

Looking at real-world examples might also offer valuable evidence on whether an Agile approach really *does* increase the opportunities for learning, local appropriation and sustainability as would be expected, or whether other factors come into play that prevent this from taking place.

More interesting still would be to see whether any of the practices of Agile could be adapted to help in a wider ICT4D / Development context – not just for software development but for participatory technology for development in general – telecentres, mHealth, GIS mapping and their related non-technical development programs .

There is some good academic work in these areas (notably Dearden & Rizvi, 2008b) and organisations such as Aptivate (www.aptivate.org) who adopt Agile approaches to developing software for International Development, so there is fertile ground for further investigation and reflections. I hope to explore some of these areas more in a follow-up ***blarticle*** soon – watch this space.

Reference List

Beck, K. (2000). *Extreme Programming Explained*. Pearson Education.

Bell, S., & Wood-Harper, T. (1998). *Rapid Information Systems Development: Systems Analysis & Systems Design in an Imperfect World (2nd edition)*. McGraw-Hill.

Beynon-Davies, P., Carne, C., Mackay, H., & Tudhope, D. (1999). Rapid application development (RAD): an empirical review. *European Journal of Information Systems*, *8*(3), 211–223.

Chapman, R., & Slaymaker, T. (2002). ICTs and Rural Development: Review of the Literature, Current Interventions and Opportunities for Action (Working Paper 192). London, UK: Overseas Development Institute.

Cooke, B., & Kothari, U. (2001). The case for participation as tyranny. In B. Cooke & U. Kothari (Eds.), *Participation: The new tyranny?* London, UK: Zed Books.

Cornwall, A. (2003). Whose voices? Whose choices? Reflections on gender and participatory development. *World Development*, *31*(8).

Dearden, A., & Rizvi, H. (2008a). Participatory design and participatory development: A comparative review. *PDC'08: Experiences and Challenges*. Bloomington, Indiana (October 1-4).

Dearden, A., & Rizvi, H. (2008b). Adapting participatory and agile software methods to participatory rural development. *Participatory Design Conference '08: Experiences and Challenges*.

Dodson, L. L., Sterling, S. R., & Bennett, J. K. (2012). Considering failure: Eight years of ITID research. *ICTD'12*. Atlanta, GA (March 12-15).

Govt. of Hong Kong. (2008). An introduction to Rapid Application Development. *Office of the Chief Government Officer, Government of the Hong Kong Special Administrative Region*.

Haikin, M. (2012). *Achieving empowerment in ICT for Development through community participation*. University of Manchester. Retrieved from http://matthaikin.files.wordpress.com/2012/10/matt-haikin-dissertation-final.pdf

Hamel, J.-Y. (2010). ICT4D and the Human Development and Capability Approach: The potentials of Information and Communication Technology. *UNDP Human Development Reports*, (37).

Heeks, R. (2002). Information Systems and Developing Countries: Failure, Success and Local Improvisations. *The Information Society*, *18*(2), 101–112.

Heeks, R. (2008). Success and Failure in eGovernment Projects. *eGovernment for Development*. Retrieved from http://www.egov4dev.org/success/

Heeks, R. (2010). Do Information and Communication Technologies (ICTs) contribute to development? *Journal of International Development*, (22).

Kothari, U. (2001). Power, Knowledge and Social Control in Participatory Development. In B. Cooke & U. Kothari (Eds.), *Participation: The new tyranny?* London, UK: Zed Books.

Larman, C. (2004). *Agile & Iterative Development*. Pearson Education.

Matuszek, D. (2008). Extreme Programming. *University of Pennsylvania*. Retrieved from http://www.cis.upenn.edu/~matuszek/cit591-2010/Lectures/00-extreme-programming.ppt

Rozendal, R. (2003). *Cultural and Political Factors in the Design of ICT Projects in Developing Countries*. The Hague, Netherlands: International Institute for Communication and Development.

Schech, S. (2002). Wired for change: the links between ICTs and development discourses. *Journal of International Development*, *14*(1).

Thompson, M. (2008). ICT and Development Studies: Towards Development 2.0. *Journal of International Development*, (20).

Walton, M., & Heeks, R. (2011). Can a process approach improve ICT4D success? (Development Informatics Working Paper 47). *University of Manchester*.

Wells, D. (2009a). Agile Software Development: A gentle introduction. *Agile Process*. Retrieved from www.agile-process.org

Wells, D. (2009b). Extreme Programming: A gentle introduction. *Agile Process*. Retrieved from www.extremeprogramming.org