An Agile Approach to Instrument Control Software at the Australian Nuclear Science and Technology Organisation

P.V.Hathaway, T.Lam, N.Hauser, A.Gotz, F.Franceschini
Bragg Institute, ANSTO, Lucas Heights NSW Australia

"Build projects around motivated individuals.
Give them the environment and support they need,
and trust them to get the job done."
From *"Principles behind the Agile Manifesto"* [1]

1 Abstract

The Neutron Beam Instruments Project at ANSTO is undertaking the commissioning of up to eight instruments and associated ancillary equipment by mid-2006.

For the control system software, requirements were elicited from an international community, software architecture decisions are guided by international expertise and contributors are distributed across at least three continents. A successful outcome depends upon the collaborative skills of the development team and their ability to respond to emerging requirements and changing resources.

Agile Software Development emphasises continuous adaptation to the project goals, developer skills, and personal interaction, over the traditional documentation bound, definitive plan and process driven practices.

This paper provides an overview of Agile Software Development principles in the context of a science-driven organisation and relates the principles' relevance and impact on the Neutron Beam Instrument Project to date.

2 Software Development Processes in the Scientific Context

*" At regular intervals, the team reflects on how to become more effective,
then tunes and adjusts its behavior accordingly."* [1]

Contemporary software engineering provides a model that constrains the capacity of a development effort by, at least, the factors of time, scope, and resources, including budget. Often as the project progresses, an emerging change in one factor is the cause of significant shifts in the others. A failure to cope with shifting constraints results in project failure.

Research and educationally driven projects often exhibit tolerance, allowing the scope of one project to overlap with that of subsequent work, or redefining the project scope or resources to cater for shifting expectations. Cautious planners can base initial estimates on worst case experiences, allowing the project to either nominally succeed at a higher cost in time and resources than it should have received, or achieve more modest targets than was possible.

To date the success stories in scientific software developments have been primarily due to extraordinary individuals and dedicated teams that have built up years of experience. There is an increasingly large body of published knowledge about scientific software development as facilities redevelop their systems.

What is required in the next generation of projects is the capability to make use of that experience in conjunction with the energy of more recently qualified skilled developers and their knowledge in contemporary techniques and tools.

3 Manifesto for Agile Software Development

Several software development methodologies in the last decade have sought to build upon the strengths of developers. They recognised the individuals' contributions, the effects of close and frequent collaboration, and the benefits of short and frequent iterations of the only artifact that really mattered - the software itself.

The methodologies' inventors and practitioners met in 2001 to discuss common themes, resulting in this statement of their shared values: [0]

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

In addition, the group documented a dozen principles behind the Manifesto [Appendix A]. However, the key differences between methods based on these principles and the more established methods are:

- Agile methods emphasise adaptation over prediction. The design and construction plans embrace and manage emerging requirements and design ideas, adapting frequently over short iterations to lower risk.
- Agile methods are centred on people rather than process. The developer is recognised as the technical expert, expected to be skilled and motivated. The management and team are supported as being in the best position to plan the work. The customer is identified as the arbiter of hat the system should do and what works best.

A number of methodologies have been labeled as Agile, including eXtreme Programming, Feature Driven Development, Scrum, Crystal, and Adaptive Software Development. These methodologies vary in their required discipline and scalability, and so cannot be optimally applied to all projects.

Fowler [5] suggests that adaptive processes are best suited to projects with:

- Uncertain or volatile requirements
- Responsible and motivated developers
- Customers who understand and will get involved

In addition, the prevalence of small teams, collaborative environments and continually emerging developments, mark Agile methods as well suited to a significant proportion of scientific software projects.

4 A Case Study: The Neutron Beam Instrument Project

"Simplicity--the art of maximizing the amount of work not done--is essential."[1]

The Neutron Beam Instruments Project (NBIP) Computing team is tasked with providing a modern, stable, extensible control system for the planned and future instruments, whilst catering for a range of operator expertise. What differentiates this project is that the team is assembled and working to create these systems, in the absence of installed hardware - sometimes in the absence of identified components.

At present, procurement of components for each of the instruments is proceeding apace. The *opportunity* exists to create stable, generic systems from a clean slate, without significant legacy overheads. The *challenge* is to produce proven software systems in parallel with instrument procurement and installation tasks. The control systems must be ready to integrate with up to eight distinct hardware configurations and ancillary devices.

To meet these objectives, a number of major systems, applicable across all instruments, are required:

- *Control System*: a common core control application, extensible to cater for all instrument configurations
- *User Interface*: a customisable user interface built upon a common framework
- *Configuration Management System*: a repository for all instrument configuration information
- *Development Methodology, Practices and Tools*: to support the creation of these systems in a controlled and timely manner

The resulting selection is a combination established software components, open source development tools, traditional project management and agile development practices.

5 NBI Project Management and Software Development Methodology

"The best architectures, requirements, and designs emerge from self-organizing teams."[1]

Project management of the umbrella Neutron Beam Instrument Project adheres to mature best practices including rigorous scheduling, detailed budgeting, risk management, progress tracking and document control. Whilst meeting these requirements, the Computing team aspires to apply the best software development practices available.

Any software development process consists of several common components. [Palmer] lists them as:

- statement of purpose
- roles with characteristics and relationships defined to fulfil the purpose
- a team of people with skills and experience to carry out the roles
- a set of processes (formal or informal) used by the individuals to produce their software system
- a set of technologies - languages, tools, components and products used to develop the target system
- an architecture, a new or borrowed framework, that provides a basis on which to design, plan and build the target system

In the context of this case study, the purpose is known, the roles and teams selected. There follows an exploration of practices and tools, addressed within the experience of our project, and the principles of development to which we aspire. Day to day activities of the group follow an Agile principles philosophy, drawing upon eXtreme Programming (XP) practices and the concept of Planning by Feature.

5.1 eXtreme Practices

eXtreme Programming (XP) is a set of principles and specific practices that implement them. The inventors of XP, Kent Beck and Ward Cunningham, are signatories to the Manifesto for Agile Software Development.

A compact description and investigation of the eXtreme principles and practices is made in [Astels]. The practices are grouped here in Figure 5.1.

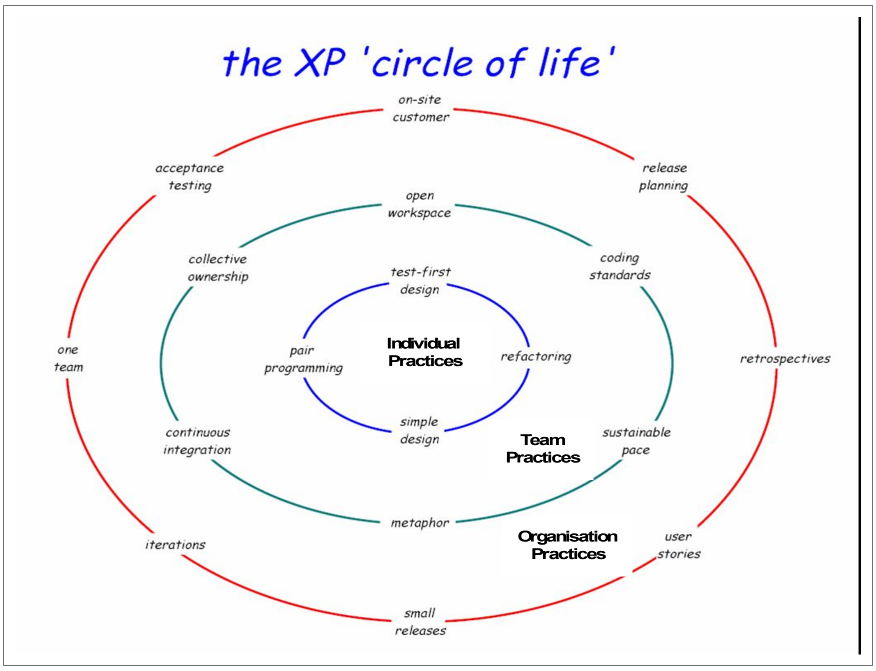

Figure 5.1 eXtreme Programming Practices

Just as the Agile principles are people focussed, XP practices are centred around the developer, the development team and the ways the organisation supports customer interaction with the team.

5.2 Developer Practices

Everyone on the NBIP Computing team takes responsibility for requirements gathering, modification to architecture and design, initial testing and integration. As various tools are added to the development environment, the team member who makes the addition is responsible for documenting the installation, configuration and any data recovery procedures.

The traditional sequential processes of software development are regarded here as peer processes, executed in parallel with frequent synchronisation points to stay on track.

To support collective understanding of the architectures and system designs, nominated coordinators for each major component are expected to deliver seminars and tutorials to the rest of the team as the component matures. The coordinator also provides a contact person for the component to their user and developer communities.

5.3 Team Practices

A main thrust of XP is to frequently adapt the activities of the development team to emerging requirements and understanding. To maintain awareness of the requirements, Agile principles mandate frequent interaction with customers - XP demands that support from a customer representative is always available.

With close collaboration, feedback and emerging requirements must be assimilated quickly. In order to respond, the team aims to sustain focussed effort over short development iterations.

Daily meetings of the NBIP Computing team are held to synchronise their activities. To keep meetings short and relevant "Stand-Up" meetings are held early every day, in which participants remain standing and are free to return to work after a specified period. Developers recount the problems and successes of the previous day, and propose how they will proceed. Everyone in the team remains informed as to what the team requires and is doing. Developers are free to offer assistance to overcome problems. Even if a developer has missed a meeting, they are brought back up to speed very quickly.

The work is carried out with developers, and sometimes customer representatives, in close proximity - using an open workspace, tackling problems and communicating solutions as they arise.

XP specifies a shared metaphor to assist team understanding and communication. In the case of instrumentation the physical instrument itself and its interfaces provides a clear picture for the team to share.

5.4 Feature Driven Planning

The basis of Feature Driven Development (FDD) was described in a single chapter of [Coad]. It documented a way of developing a complex software application with a moderately large number of developers of varying skill and experience.

The FDD methodology mandates five processes ([Palmer] Chap 4):

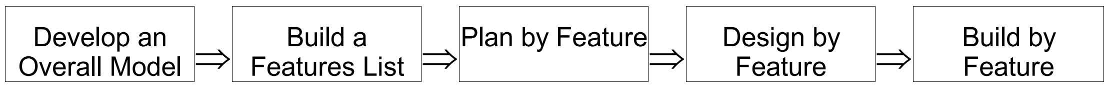

Figure 5.2 Processes of Feature Driven Development

The NBIP Computing team is not adopting FDD but borrows the concept of a feature, and the advantages behind planning and building by sets of features.

A feature is an artifact, resulting from the requirements gathering process.

> A feature is a short statement describing a small operation performed by a part of the software system to achieve a result relevant to the customer.

User stories are plain english description of operations to be carried out by the users of a software system. The development team records collections of stories from regular meetings with the customer representatives. From these user stories, sets of features are derived, careful to maintain the user perspective of functionality. Customer representatives assist in prioritising the features.

In this way, the development team has records of user requirements, independent of implementation and architecture.

The scope of each feature and effort required to implement it is called its granularity in deference to the milestone metaphor. FDD suggests that each feature take no longer than two weeks to implement. This granularity supports ease of task allocation, workload balancing, and a measure of progress from the feature lists implemented. Progress can be reported in units that the customer can both understand and readily verify from the next release update of the application.

In FDD the Design by Feature and Build by Feature processes are repeated until all feature sets have been implemented. Our feature planning commits only to the current iteration, indicating which feature lists will be implemented, and for which group of customers.

Sets of features are scheduled for implementation in sequence, depending upon their priority or dependencies. At the end of each iteration, feedback on the planning process is derived from the proportion of feature lists implemented. The feedback assists better planning of subsequent iterations.

6 Collaboration

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation." [1]

6.1 Control System

The NBIP Computing team has selected the SINQ Instrument Control System (SICS) as the foundation for its server-based control software. SICS is an operational, stable, tried-and-tested software platform in use at PSI. The criteria and decision process used to select this system are covered by [Hauser].

Early in 2004, the NBIP invited the primary author of the SICS system, Mark Koennecke, to attend ANSTO for development time and a series of seminars and tutorials. In addition to providing valuable developer insights into the design and use of SICS, issues in the ongoing development of SICS were identified and discussed.

The most direct impact of this collaboration is the addition of site libraries to the architecture of SICS, enabling specific sets of drivers and scripts to be developed for the site whilst the core system code is maintained independently. At the same time, access to the PSI-specific modules provides many templates and examples upon which the ANSTO site library is being developed.

It is specifically the SICS server application and device driver libraries that is incorporated into ANSTO's systems. Each instrument will run its own SICS server on a virtual LAN, connected to device controllers and data servers via TCP/IP over ethernet.

This connectivity paradigm decouples the SICS server machine from the need to share its internal bus with devices other than a network interface card. The server can then be run on a reliable, standard architecture PC mounted in a rack located in the vicinity of the instrument, and accessed via client machines in both the nearby instrument cabins or by any number of remotely located clients with the necessary access privileges.

The server has an object-oriented architecture and a constrained embedded Tcl interpreter, used to.

- parse command line instructions from the client
- execute initialisation scripts at server startup to define the instrument configuration
- interpret and execute server-side scripts that provide extended functionality
- interpret and execute user scripts written in Tcl

Device driver modules are developed for specific types of equipment and then using a script at initialisation defines the numbers, combinations and configuration parameters of the devices for that instrument.

By constraining instrument subsystems to a set of standard controllers for motion and sample environment, the development and testing effort is reduced, with increased system reliability. The architecture's capability to add new device modules as required ensures that the team maintains planning flexibility.

SICS components are synchronised with a code repository at the PSI.

6.2 Client Architecture

The NBIP Computing team settled quickly on using client-server architecture for the reasons outlined in [Hauser].

ESRF software engineer, Andy Gotz, was engaged during his sabbatical to provide guidance on selecting a configurable, platform tolerant framework for developing the graphical client user interface to our SICS-controlled instruments. To meet these requirements, investigations began into Java-based frameworks that would support the architectural decisions being made.

The advent of the Eclipse Rich Client Platform (RCP) with native integration of the Eclipse [2] development tools provides us with a ready-made customisable, modular, OS-independent tool kit for implementing our client application, named *Gumtree*. The architecture of the Gumtree platform is inherently flexible enough to support ANSTO-specific plug-ins (*GumNIX*) and may be further abstracted to cater for alternative control systems, such as EPICS or TANGO.

6.3 Further Opportunities

"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely." [1]

Wherever possible, the Computing team is leveraging off developed systems.

SICS is being further collaboratively developed to cater for site-specific extensions. The core of the SICS server code is maintained by Mark Koennecke of the Paul Scherrer Institut (PSI).

The impact of catering for inclusion of site-specific equipment and methods has been extensions to the core code base. Bringing the core code itself under the purview of a new team of motivated developers has sparked a stream of potential enhancements, challenging the system's extensibility.

An open source philosophy applies for tools and instrument control client development. That is, the team gives preference to open source development tools and utilities. Our client application platform, Gumtree, is registered on SourceForge as an open source project, open to other collaborators, and coordinated by our team. Developers from ESRF and PSI have registered their interest in our client platform.

Due to the collaborative philosophy adopted, virtually all other support packages and utilities selected have communities of users and developers maintaining and extending usable code modules applicable to our systems - including Eclipse, its plug-ins, and NeXus and TANGO.

7 Agile Development Toolkit

Integral to establishing the new team is the adoption of a development methodology and the selection of a core set of developers' tools. Both must be simple enough to learn quickly, effective enough to produce stable solutions over a range of devices, and flexible enough to respond to the evolving needs of instrument scientists.

Open Source Integrated Development Environment

Eclipse [2] is an open source integrated development environment and application platform. In addition to project support and the provision of Java and C/C++ editing tools, Eclipse plug-ins have given us graphical and menu-based interaction with our

- source code repository (cvs or vss)
- debugging tools (gdb)
- source code documenting tools (javadoc and doxygen)
- bug/issue database (bugzilla, MySQL)
- external editors (customised for Tcl, perl, Python etc)

Plug-ins also provide native functionality supporting

- refactoring tools for java and C/C++
- java unit testing
- database schema modeling with SQL output
- UML support (being developed)
- profiling of running processes (Hyades)

There are extensive libraries of freely available and commercial plug-ins for Eclipse, providing additional functionality or interfaces to external applications.

The entire development team is able to use the same integrated development environment whether developing in C/C++ or Java, debugging, testing or producing linked documentation. This enhances the team's capability to switch developers between tasks.

Other Community-Based Packages

- Apache / Bugzilla / Perl / MySQL for tracking bugs, issues and requested enhancements.
- MySQL also provides the backend DBMS for the configuration management system.
- The NeXus and HDF libraries are required by SICS. The communities supporting those products also provide APIs and data browsers important to our development.

- The Tango package and omniORB object broker.
- DejaGNU test framework

Configuration Management and Persistence

An important aid during development is the automatic generation of instrument configuration files from a central repository, and the visualisation of those instrument configurations for use as working documents.

In instrument operation, persistence of component parameter values supports recovery from fault conditions. Access to those parameters across a distributed system is advantageous for remote instrument state detection and debugging.

In order to fulfil both sets of features without duplication, we require a central, reliable, fast, and network accessible repository of instrument components, that captures the components' relationships in any one configuration, with a record of their current states.

The current database schema is customised for complete coverage of the necessary elements within any SICS initialisation script. Therefore, through appropriate queries and formatting, a SICS initialisation script can be generated based on the physical configuration specified. It is intended to use Java Database Connectivity (JDBC) functionality to code the query and format the response into a script, complete with comments.

The database schema itself is prepared using an Eclipse plug-in. The scripts to configure the database can be generated for multiple dialects of SQL. The database design is documented and under version control.

8 HIFAR MRPD: Adaptive Planning and Delivery of Code

"Working software is the primary measure of progress." [1]

In the absence of installed instrument hardware, the development team revived an earlier project idea to prototype our software systems on an existing instrument resident in HIFAR[1]. In return for some additional development load, a number of positive outcomes are expected:

- Gain confidence in the capability to adapt and extend SICS to a real, specified instrument
- Gain confidence through a practical test of the team's development process, task management and commitment to methodology.
- Increase the team's visibility amongst existing and NBIP instrument scientists by working on and getting feedback from real systems and people
- Provide a platform for continuously testing small enhancements
- Test the current implementation and feature sets of server (configurability) and client (control options, instrument status display, experiment status display, data display)
- Provide a test our testing and debugging processes
- Give all users a working client for relevant feedback on usability
- Get access to real data and prompt discussion on how to manage it
- Test our frontline data reduction and analysis tools

The HIFAR MRPD instrument was selected. It provided a modest target in terms of number of devices to be controlled and device classes that needed to be coded. The existing user interface provides a limited set of features, which can be duplicated and enhanced by the Gumtree client. The instrument is scheduled for reasonable usage, giving both time to develop and exposure to the user community.

The problem domain was examined to set objectives and constraints. Development then proceeded in two streams - for the client and server, with regular meetings to synchronise required functionality.

The team is now working daily with a domain expert - a current scientist, developer and maintainer of HIFAR instruments. He provides the team with the knowledge of how the diffraction instruments are being used, details on current and planned functionality, and provides immediate feedback on new operational code. In return, he is receiving in-depth access to the architecture and tools in use so that he can assist in further feature planning and design decisions.

At the completion of each device driver, bench testing is carried out to check the communications and functionality. Live testing of each device driver in-situ is carried out under controlled conditions.

[1] HIgh Flux Australian Reactor - ANSTO's current operational research reactor.

MRPD devices and features were integrated into the development plan as a subset of the current feature list. A complete set of device drivers have been coded and reviewed in short order. MRPD is now awaiting scan-specific coding and final integration tests before the instrument scientists and users begin to operate and provide feedback on the applications.

9 Methodological Aspirations

As a new team we did not establish a full development environment, providing all tools and supporting all practices before development began in earnest. The NBIP Computing development environment is evolving in parallel with the development project. This is less than ideal, but a very real constraint based on our resources. We have the tools to practise traditional lines of development, but choose to seek out tools and practices that better reflect our agile aspirations.

The adoption of feature sets aims to provide a mechanism to balance workload between and for developers. After planning a feature set, developers can focus on handling short lists of features over the iteration whilst the team manager has a simple progress metric. If any features prove problematic, or a developer becomes unavailable, resource allocation can be altered for the well-defined packages of work.

We have committed to announcing application milestones to our user community in three monthly intervals. We wish to maintain this interval for planning sets of features, but aspire to release updates at a higher frequency within the team. A prerequisite for this is full automation of our application packaging and deployment. Once automation is completed, each new feature can be made available to an application package within minutes. Extensive integration tests can be run overnight and acceptance tests run as quickly as our user representatives can be informed. In this way, the progress through feature lists can be objectively monitored and feature sets deployed simply at the end of the iteration period.

As per eXtreme Programming we wish to refine the granularity of development tasks. XP calls for coding tasks that should be completed by a pair of programmers within the day, including successful unit testing. Failure to do so indicates the task scope is too large and that it should be decomposed. Easy to propose, potentially difficult to achieve, nigh impossible without the support of the other practices.

Our testing regime still follows informal lines - constant testing by a single developer of their own code. Successful builds are mandatory and are automated, but the unit-testing framework is incomplete and not uniform. Integration testing remains a task performed by developers after a set of features has been impemented - we are trying to move this to more frequent intervals.

Pair programming is currently practised at the whim of each developer. It appears more of a psychological barrier than practical impediment to implement. In the authors' limited experience, the practice shows great potential for producing immediately tested, virtually bug-free code to an agreed upon coding standard that ensures readability. It will promote bringing all team members up to speed on all parts of the project. Introduction of this practice is highly dependent on the use of other practices.

10 Conclusions

Scientific software development progresses by the efforts of talented practitioners and dedicated teams. To advance further and faster, more effort should be placed into harnessing the existing body of knowledge and adapting contemporary software principles and practices.

Agile software development principles are people focussed. Methodologies subscribing to agility, such as XP and FDD, contain practices that offer an extremely attractive alternative to heavyweight documentation-driven development processes. The methodology originators stress the adoption of specific sets of practices over using a few practices in isolation. In the spirit of scientific and software exploration, however, this should not preclude experimentation with the practices or variations on their themes.

The Neutron Beam Instrument Project at ANSTO aspires to accelerate the production of quality applications by trialing and adopting many agile-based practices. Daily synchronisation of the team effort and frequent customer contact has had significant influence on the motivation and focus of the new team. Collaboration, communal development of software tools and end-user applications, flow of information and feedback are contributing to positive results.

A number of practices are difficult to introduce in isolation. Test-driven development, pair programming, adaptive planning cycles and continuous refactoring are initially counter-intuitive, but have proven successful in the hands of thoughtful developers.

Using these methodologies does not guarantee success. To date, we have found that the experience and skills of the developer are still the driving factors in achieving excellence. However, we are optimistic that with the right people, agile principle-based practices can deliver that excellence faster, whilst sustaining an enjoyable and stimulating team environment.

11 References

[Astels] Astels, D., G.Miller and M.Novak "*A Practical Guide to eXtreme Programming*", Upper Saddle River NJ: Prentice-Hall PTR, 2002. ISBN 0-13-067482-6

[Coad] Coad, P., et al. "*Java Modelling in Color with UML*", Upper Saddle River NJ: Prentice-Hall PTR, 1999.

[Fowler] Fowler, M. "*UML Distilled*" 3rd Ed., Addison-Wesley Publishing Co., 2003.
ISBN 0-321-19368-7

[Hauser] Hauser, N., et al, "Choosing an Instrument Control System", NOBUGS Conference 2004.

[Palmer] Palmer, S.R., and J.M.Felsing "*A Practical Guide to Feature-Driven Development*", Upper Saddle River NJ: Prentice-Hall PTR, 2002. ISBN 0-13-067615-2

12 URL References

[0] "Manifesto for Agile Software Development" <http://www.agilemanifesto.org/>

[1] "Principles behind the Agile Manifesto" <http://www/agilemanifesto.org/principles>

[2] "Eclipse" <http://www.eclipse.org>

[3] "eXtreme Programming" <http://www.extremeprogramming.org/>

[4] "Feature Driven Development" <http://www.featuredrivendevelopment.com/>

[5] "The New Methodology", Martin Fowler <http://martinfowler.com/articles/newMethodology.html>

[6] "The Agile Alliance" <http://www.agilealliance.org/>

13 Appendices

[A] "Principles behind the Agile Manifesto"

Appendix A:

Principles behind the Agile Manifesto [1]

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 behavior accordingly.