What is Agile?

May 2011
HP Confidential

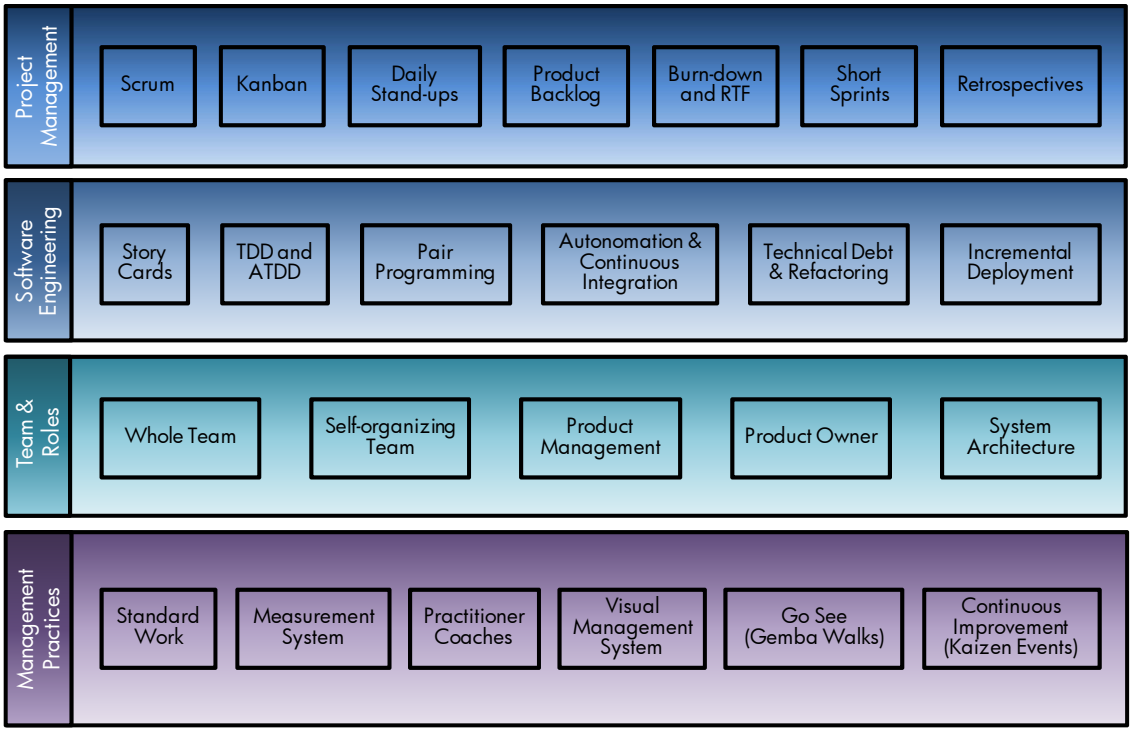
1 Introduction

The term "Agile Software Development" was coined at a gathering of software engineering thought-leaders in Snowbird, Utah in 2001, which also led to the creation of the Agile Manifesto[i]. It has since been used to cover a family of related software development methods such as Scrum, Extreme Programming, DSDM, FDD, Crystal, OpenUP and others.

Over time, Agile methods have evolved by absorbing more influences from thinking tools such as Lean and Systems Thinking, Lean Six Sigma, the Theory of Constraints, Queuing Theory and many other organizational and action tools.

This paper describes a synergistic framework of twenty-four habits that contribute to the effective use of Agile Methods at HP. Organizations that learn how to continuously refine their practice of these habits will manifest the meaning of Agile.

For ease of reference, the habits are grouped into four categories: Project Management, Software Engineering, Team & Roles, and Management Practices.



2 Project Management

In Agile methods, the Project Management habits are practiced by the entire Team rather than just one individual. These habits focus on the structure of the work process.

2.1 Scrum

Scrum is a simple framework aimed at achieving results of high value and high quality consistently and steadily. It is a way of structuring work activities such that attention is always devoted to achieving the highest value outcomes first.

The Scrum framework relies on four roles, three key artifacts and three ceremonies.

Four Roles

- Product Owner: responsible for the overall value of the product being created; prioritizes the Product Backlog and decides how to balance the needs of stakeholders—see the Product Owner habit.
- Scrum Master: keeps the Team healthy and happy; maintains a steady pace of work, removes impediments, defends the Team from unreasonable or absurd demands
- Team Member: generalizing specialist; self-organizes to get the work done, applying one's intellect to the most pressing task at hand.
- Agile Coach: accelerates the Team's continuous improvement.

Three Artifacts

— Product Backlog: list of desired product features or outcomes, prioritized in order of value to the key stakeholders.

— Sprint Backlog: subset of work from the Product Backlog that the team commits to complete in a sprint; only enough work to fill the Team's capacity for the Sprint, also known as the Team's velocity.

— Burn-down chart: at-a-glance perspective of the work remaining; may be shown on multiple levels: one for the Sprint and one for the current Release.

Three Ceremonies

— Sprint Planning: the Team estimates the top priority items in the Product Backlog, and in agreement with the Product Owner fill to capacity the Sprint backlog for the next Sprint.

— Daily Scrum: the Team meets each day to keep in tune with what's happened yesterday, what each Team member plans for today, and what impediments may have cropped up in the mean time.

— Sprint Review and Retrospective: for the Review, the Team demonstrates for the Product Owner the product increment completed during the Sprint. During the Retrospective, the Team recalls what went well, identifies what didn't go quite well, and identifies and prioritizes countermeasures to improve the work product and the work process.

The HP Agile work process is based on the Scrum framework, enriched with the other habits described here.

2.2 Kanban

The name "Kanban" is a Japanese term, literally meaning "visual record" or "card", and it refers to a card system used to visualize and manage the flow of work.

Kanban is a complementary approach to time-boxing; its objective is to achieve smooth one-piece flow by limiting the work in process in the various steps of the work process, based on the current skills and capacity of the Team members.

Kanban is often used to reduce cycle time and make the Team's work flow smoothly within the bounds of a Sprint, by making capacity bottlenecks visible. In situations where clients may benefit from completed work items immediately, Kanban may also be practiced without a time-boxing framework surrounding it.

2.3 Daily Stand-ups

Each cross-functional, co-located Team meets daily for 15 minutes or less, at the same time and in the same location.

It is customary for team members to be standing rather than sitting during this high-energy meeting, hence the name Daily Stand-up.

In this session, each Team member gives a brief summary of:

— What they accomplished since the last meeting;

— What they are committing to achieve before the next meeting;

— What impediments they have come across and overcame or are stuck with.

Daily Stand-ups serve to improve the Team's collective awareness of the true status of work, highlight key impediments, and identify promising collaboration opportunities.

Daily Stand-ups are not problem-solving meetings, although they may trigger problem-solving sessions to remove impediments and engage in mistake-proofing activities.

Scrum Masters ensure that the Daily Stand-ups run regularly and smoothly and coach their Teams to keep them short & sharp.

For Agile Product Development efforts that involve multiple teams, multiple Daily Stand-ups may occur in quick succession, with representatives from each team participating in higher-level Stand-ups until all teams have synchronized their views on the progress of work.

2.4 Product Backlog

The Product Backlog is the list of desired product outcomes or features, prioritized in order of business value. The items in the Product Backlog represent units of valuable functionality that serve specific needs of the product's user community.

The Product Backlog is dynamic—items may be deleted or added at any time during system development. It is prioritized by business value, and items with the highest priority are worked on and completed first. It is progressively refined, with higher priority items elaborated just-in-time, and lower priority items left intentionally coarse-

grained. This reduces the cycle time of feature realization and reduces the amount of work in process at all times.

2.5 Burn-down & RTF

A burn-down chart is a graphical representation of the progress achieved by tracking the number of completed backlog items at the current point in time. Backlog items can only be considered complete when they fit the "definition of done".

The "definition of done" is a set of acceptance criteria jointly agreed upon by the Product Owner and the Team, and it typically covers all the assets required to validate and successfully deploy and make useful a feature set, including executable test specifications, working code, user and technical documentation and so on.

Burn-down charts can be prepared at Release and Sprint levels. In essence, the burn-down chart is a run-chart for completed work items. The trend exposed by this chart is useful for predicting whether the rest of the work for the current Sprint or Release is likely to be completed in time.

RTF is an acronym for Running Tested Features:

— "Running" means that the features are work well within an integrated product increment.

— "Tested" indicates that features are passing all the agreed upon acceptance tests.

— "Features" refers to elements of system functionality that provide measurable value to end-users.

The RTF chart is very useful for ensuring that a steady flow of business value reaches the user community, and may serve to identify and diagnose impediments that may be hindering the healthy improvement of the product development work process.

2.6 Short Sprints

Agile Teams are long-standing, continuously learning how to collaborate effectively, refining their individual and collective skills over the course of many Sprints—see the Self-organizing Team habit.

A Sprint is an iteration of the product development work process, usually time-boxed between one and four weeks in duration. For any given product development effort and set of Teams, the Sprint duration tends to remain constant over time.

Shorter Sprint durations such as a week or two are beneficial for novice teams, giving them better opportunities to accelerate the improvement of their work process habits—see the Retrospectives habit.

Short Sprint durations reduce waste through lower amounts of work-in-process in terms of Product Backlog items elaborated for creation. The longer the Sprint durations, the more Product Backlog items are required to be ready for the Sprint Planning sessions. This is also likely to increases the cycle time for individual backlog items, increasing the time interval between the moment an item is ready and the moment it is done.

2.7 Retrospectives

Retrospective meetings are held at the conclusion of each Sprint and each Release. The purpose of the Retrospectives is to inspect the quality of the work process in terms of team habits, mastery of skills, relationships and use of tools.

This is an opportunity to reflect on the effectiveness of the countermeasures implemented in the previous Sprint and the nature of the most significant impediments that still affect the Team at this time. It is vital to establish a culture where it is safe and natural to surface problems, bringing them to the attention of the Team, harnessing the talents of the entire Team in order to devise the most adequate countermeasures.

The Team identifies problems and prioritizes them. Then, one by one, in order of priority, the Team explores the root causes that lead to the problem, and generates options for countermeasures that would effectively address the root causes.

Since the Retrospective session is limited in time, usually only less than a handful of the top key impediments have a chance to be explored in depth. That is very good, since the capacity of the Team to put countermeasures into action over the course of the next Sprint is also limited.

By focusing only on one or two countermeasures and thoroughly putting them into practice, the pace of improvement is sustained much better. By implementing countermeasures thoroughly, the Team can measure their effects on the quality of the work products and the work process. Otherwise, if the countermeasures are incomplete, their effects cannot be assessed.

3 Software Engineering

The Software Engineering habits help the Team to achieve and sustain excellent quality and fitness for purpose for features of high value to the user community. These habits focus on the content and quality of the work process.

3.1 Story Cards

In Agile methods, the discipline of requirements management shifts emphasis from documentation to conversation. Rather than creating elaborate requirements specifications as documents to be handed over, the Agile approach calls for frequent conversations focused on user stories, held between Team members and client stakeholders—see the Whole Team habit.

Story Cards (such as index cards or sticky notes) are used to record the key elements of a User Story, such as the role or persona served, the core objective of the story, the value expected and some of the acceptance criteria that indicate the story is complete.

The INVEST acronym is often used to evaluate the fitness of User Stories: **I**ndependent, **N**egotiable, **V**aluable, **E**stimable, **S**mall and **T**estable.

Stories capture elements of system functionality that will be valuable to users and are elaborated progressively through intense conversation between Team members, their diverse professional backgrounds illuminating all aspects that would need to be covered to complete the story, improving the accuracy of estimates.

In addition to using Story Cards as a focal point for conversations that illuminate all the relevant aspects of scope, the complementary practices of literate programming[ii] and domain driven design[iii], are vital to ensuring that code is understandable and easily maintainable by any Team, particularly in large-scale Agile programs—see also the Whole Team habit.

3.2 TDD and ATDD

Test Driven Development (TDD) is a design discipline involving the creation of executable specifications, in the form of automated unit tests, before any code is designed. Then, just enough code is designed and created to pass the tests, keeping the architecture as simple as possible. The cycle is then repeated every few minutes or hours at the most, rather than over a span of days.

By providing excellent automated test coverage, the practice of TDD ensures that code can be re-factored without fear of introducing regression defects, leading to a significant reduction in the time spent debugging or fixing defects while a simple, high quality system design is maintained.

Acceptance Test Driven Development (ATDD) is a variant of TDD relying on higher-level automated functional tests.

Typical ATDD practices rely on tools that make it possible to expressing executable acceptance tests in a form resembling natural language. In this way, analysts, testers and developers collaborate in creating ATDD tests as an executable and highly efficient expression of requirements specification.

3.3 Pair Programming

Pair Programming is a habit in which two Team members pair up and work together side by side, taking turns at the same keyboard, driving the same workstation. One person, the “driver”, creates unit and acceptance tests, code and documentation; and the other person, the “navigator”, reviews the assets in real time and provides improvement suggestions.

In this way, the “driver” is free to concentrate on the tactical challenges of completing the current task, while the “navigator” has the opportunity to also consider more strategic issues. Every thirty minutes or so, the two Team members switch roles, and every few days or so pairs may rotate Team members.

In addition, team learning is accelerated as knowledge and skills spread quickly among team members, while good working habits get reinforced through positive peer pressure.

3.4 Autonomation & Continuous Integration

Autonomation refers to “automation with a human touch”, it is a habit meant to prevent the production of defective products by stopping and fixing defects as soon as they are detected. This also focuses everyone’s attention on finding the root causes of problems; mistake-proofing countermeasures then prevent recurrences.

Continuous Integration (CI) is the habit of integrating new or changed code into a working software-intensive system as soon as possible (on the order of several times each day) testing to make sure that nothing has been broken by the latest addition, and if defects have been

introduced, then the offenders are notified immediately and the changes are rejected.

The goal is to always keep a working software system by making small changes, and grow the system organically. CI brings immediate feedback to developers on the system-wide quality impact of code they are writing, which provides excellent visibility, prevents "integration hell" and reduces time-to-market.

3.5 Technical Debt & Refactoring

Technical Debt refers to the adverse effects of hasty coding efforts on software design quality, leading to increased software entropy when insufficient attention is paid to maintaining a simple, high-quality system design—see the System Architecture habit.

Just as compound interest aggravates a deficit, so comparatively minor coding deficiencies left untreated will accumulate, and their combined effect will obfuscate the code base and lead to brittleness. Over time, code understandability and quality will degrade, requiring increasingly more effort to be spent in defect removal.

Refactoring is the habit of maintaining design quality by investing in transforming code structure in order to keep it simple and maintainable even as code capability increases. By refactoring judiciously and diligently over time, the code base can be kept in excellent condition so that maintenance and enhancements can be carried out efficiently and economically.

3.6 Incremental Deployment

The Incremental Deployment habit ensures that valuable business functionality can reach the user community in small increments delivered at relatively short intervals of a few weeks or months.

The effective practice of Incremental Deployment requires very reliable and streamlined deployment capability, usually involving substantial automation that provides mistake-proofing and reduces effort. In addition, short cycle times can only be achieved and sustained when the efforts required for system validation are minimized—see the TDD and ATDD, and Autonomation & Continuous Integration habits.

Incremental Deployment enables the user community to reap the benefits of improved functionality very early, also usually requiring a shallower learning curve as relatively few changes are introduced with every release—see the Product Management habit. In addition, it is often possible to use the profits derived from the use of early product increments to fund subsequent enhancement efforts.

4 Team & Roles

The Team & Roles habits help people to sustain collective learning, focusing on the organization patterns conducive to success in the software engineering game of collaboration and invention.

4.1 Whole Team

The Whole Team habit refers to the practice of creating small, long-lived, cross-functional teams that include a core of generalizing specialists with expertise in all the disciplines necessary to transform Product Backlog items from "ready" to "done"—see the Product Backlog habit.

Small team units—less than a dozen people—work better than large teams because people can develop closer relationships and self-organization is more effective. Long-lived teams are more efficient, because team learning enables them to significantly improve their collective work processes. Cross-functional teams are more effective, because they can reduce product development cycle time and diminish the amount of wasteful work-in-process at any moment.

In addition, the Team may co-opt from time to time various business or engineering domain experts that enhance the team capability and serve as mentors to the core Team members for specialist expertise otherwise unavailable in the core Team.

All Agile Teams are engaged in the art of creating only just enough documentation so that the product may be sustainably maintained in peak condition, balancing the investment in creating documentation with its use and value. Examples of minimalist documentation, in addition to literate code written with excellent style and sufficient comments, may include very cost-effective assets such as wikis, whiteboard snapshots or video recordings of domain or technical experts explaining the more intricate aspects of the system in order to improve knowledge retention.

The Whole Team habit also implies collective ownership of engineering assets—everyone shares responsibility for maintaining the assets in excellent condition. Everyone is empowered to make any necessary changes, engaging in Pair Programming with more experienced colleagues as needed—see the Pair Programming habit.

4.2 Self-organizing Team

The Self-organizing Team habit is manifested by teams in which colleagues agree through consensus how to plan and allocate their tasks and to monitor and manage their work process and progress. Peer pressure rather than an authoritarian figure ensures commitments are met.

Self-organization does not imply a lack of direction. Leaders set goals, define the boundaries and constraints, bring challenges and remove impediments. The Team's job is to create as much value for the system's user community as humanly possible, within the given boundaries and constraints, swarming to meet the challenges, learning continuously to improve the work product and the work process.

In a continuously improving culture, specific roles cannot be guaranteed, since the structure of the work process is subject to continuous experimentation and change. Not all changes will turn out to be improvements, although all improvements are changes. Organizations must show their respect for people by providing job security at the expense of role security.

If, as a result of continuous improvement, the same or better results can be achieved with fewer people, this cannot be used as an excuse for workforce reduction. Otherwise, the incentive for improvement is ruined, people will no longer trust the organization, and continuous improvement will grind to a halt. People must be able to feel safe and trust the organization to provide continued employment when certain roles are optimized away. In thriving organizations, there will always be opportunities to apply the skills of master thinkers and problem-solvers to new and diverse areas for continuous improvement.

4.3 Product Management

Projects are by definition temporary endeavors undertaken to create a unique output, with a definite start and finish. Products, on the other hand, may have an indefinite lifespan, bounded only by commercial relevance.

Product Management refers to the habit of managing software-intensive systems development as a product rather than a project.

In this way, product increments can be optimized to deliver the highest possible value to the user community at frequent intervals. Also, the product development activity can stop as soon as functionality of sufficient value has been deployed, and the product development capacity—the Teams—can be refocused on other products. Rather than disbanding highly effective Teams that have consistently learned how to continuously improve their collective performance, it is far more effective to assign new work to them rather than forming new Teams for new work—see the Whole Team habit.

In addition, by keeping the Teams intact, their engineering capacity can be assessed very accurately based on their track record. This enables product strategy and product roadmaps to be articulated and release content to be adjusted according to the available capacity, significantly reducing uncertainty and increasing the chances of achieving successful outcomes.

4.4 Product Owner

The habit of commissioning a Product Owner role is vital to the quality of the system under development and the health of the product development group.

The Product Owner embodies the spirit of the product, acting as both technological evangelist and customer advocate. Through intense study and collaboration, great Product Owners develop a deep understanding of both the problem domain and the product technology, becoming adept at foreseeing user and market needs and making superior trade-off choices.

Product Owners create and adjust a compelling vision for the product over time. This enables the Team to make good choices that are well aligned with the Product Vision. Inspired by the Product Vision, the Product Roadmap covers the intent of the next few Releases at a high level. The Product Owner manages the Product Roadmap and plans the current Release. In collaboration with the Team, the plan for the current Release receives more attention and gets fleshed out in more detail, to provide a sense of how feature sets may be sequenced within the Release. Over time, as Sprints create features, the Release Burn-down chart reflects the progress achieved towards the Release goals.

At the next level of detail, the Product Owner works with the Team to groom the Product Backlog, making sure enough high-priority items are ready in time for the Sprint Planning workshops.

Throughout the work process, the Product Owner collaborates with the various stakeholders, the Team, and the ScrumMaster to achieve excellent

transparency in the status of the product development effort. The Product Owner definitely attends the Sprint Planning and Sprint Review sessions, and may attend some or all of the daily Scrum meetings as needed.

For large-scale product development, when many teams must cooperate to create a single product, a Product Owner Team is called for. This is led by a Chief Product Owner and its members include Product Area Owners, responsible for the various business domain areas served by the system.

4.5 System Architecture

The System Architecture habit refers to the practice of devoting just enough attention to strategic concerns in both the business and technology realms, considering their mutual interdependencies.

By investing barely just enough effort to create an accurate lightweight representation of the system's architecture[iv], people can reason more effectively about the most promising and appropriate approaches for enhancing the system.

For instance, product roadmaps can be shaped to take advantage of specific technology assets or techniques, or avoid the adverse effects of certain technology constraints. Also, based on such insight, research or prototyping efforts can be focused on proving countermeasures to the most important business or technology impediments.

5 Management Practices

The habits in the Management Practices category help people to sustain collective learning, focusing on the organization patterns conducive to success in the software engineering game of collaboration and invention.

5.1 Standard Work

The Standard Work habit refers to the practice of creating and maintaining a lightweight representation of the Team's work process. It represents the best way that the Team knows how to conduct its work at the current point in time.

The fact that a Standard Work representation exists does not mean that it represents a perfect and immutable work process. Rather, it is a starting point to be used in reasoning about the most significant impediments currently hindering the Team's continuous improvement.

Although an organization-wide Standard Work representation may be used as a starting point for Teams, each Team is responsible for maintaining its own annotated Standard Work representation. Agile organizations ensure that over time, common improvements proven in the context of multiple Teams migrate into the organization-wide Standard Work representation, supporting organizational learning and providing an improved foundation for the induction of new Agile Teams.

Having such a representation of the work process available, each Team is able to focus its attention accurately to diagnose systemic problems, and establish a viable measurement system to objectively and reliably assess the effectiveness of countermeasures as they are put into practice—see the Measurement System habit.

The Standard Work representation also provides an indirect benefit, as its rate of change can be used as an indicator for the health of the Team's continuous improvement activities. If the Standard Work representation is stagnant and has not been changed recently, this suggests that the Team has become complacent, and no longer invests effort into improving the work process. If the Standard Work representation is frequently amended, this likely indicates that Team members continue to have an active interest in improving their work methods and skills.

5.2 Measurement System

The Measurement System habit focuses on creating a measurement framework and consistently harvesting measurements based on which the various qualities of the work product and work process can be assessed.

The Measurement System should be focused primarily on capturing systemic metrics, as system-wide optimization is always preferable. All changes worth implementing must lead to either a net improvement in the quality of the user experience and therefore value of the product, or a bottom-line improvement for the organization as a whole through waste or cost reduction when assessed on a strategic rather than merely tactical timeframe. The most effective and worthwhile improvement is the one applied to the aspects of the work process or work product that represent the top constraint for the system under development[v]. Improvement in other areas that do not ameliorate the top constraint may be wasteful until the top constraint is satisfactorily dealt with.

Systems developed using Agile product development methods should include a Measurement System segment that facilitates an

analysis of the user community's usage patterns. In this way, the Product Owner can learn which features or sequences are most frequently used, and which features are completely unused. Based on such insight, product enhancements can be devised to improve the efficiency of the most frequently used features or sequences, or remove unused features altogether, keeping the system as lean and simple as possible.

In addition, the Measurement System should also provide information on the health of the work product from an engineering perspective—by measuring various aspects of code and design quality—and keep track of the journey of the work process as well.

Good Measurement Systems capture information from the actual tools used in the work process or during the use of the product by its user community, with minimum burden to people.

The Measurement System encourages conversation and is a vital link in the cycle of inspection and adaptation that underpins all continuous improvement efforts. Naturally, the structure and operation of the Measurement System is also subject to continuous improvement.

5.3 Practitioner Coaches

Agile is a team sport, a game of collaboration and invention aimed at overcoming complexity in software-intensive systems. No-one would expect athletic teams to show up on the field against their opponents without training together under the watchful eye of a coach. Similarly, we should expect long-lived Agile Teams to train together and sharpen their performance through coaching—see also the Whole Team habit.

Newly formed Teams, perhaps containing individuals with limited exposure to Agile ways of thinking and acting, will require more time and attention from an experienced Agile Coach in order to master the core Agile habits rapidly. Over time, all practitioners will improve their skills, and some will develop sufficient mastery over a number of domains in order to be ready to serve as mentors or coaches.

Anyone with a sufficiently advanced degree of proficiency in a particular skill or technique may serve as a mentor for colleagues that are not yet well-versed in using that skill.

To serve as a coach, a person should be able to serve as a mentor in a number of disciplines, and must be trained and mentored by one or more accomplished Agile Coaches. Through such apprenticeship relationships we ensure that new coaches have the right balance of skills and temperament to be able to help their Teams to achieve amazing results. Emotional intelligence is just as important as engineering skill.

The speed of continuous improvement is constrained by the quality of the coaching available to the Team. The habit of using Practitioner Coaches ensures that Teams get help in focusing their attention consistently on the most valuable opportunities for improvement, and then follow through in implementing efficient and effective countermeasures.

For situations in which CMMI appraisals are required, SCAMPI coaches can also included in the team learning process.

5.4 Visual Management System

The Visual Management System habit refers to the practice of using highly visible displays to make information readily available at a glance. Information radiators may address various topics such as schedules, standard work, quality or the status of various assets.

By making such information public and instantly accessible, many opportunities for confusion and mistakes are removed, and error prevention, detection and resolution are improved.

Just as it is the case with all other habits as well, the Visual Management System is subject to continuous improvement. With simplicity in mind, Teams are free to experiment with variations that will provide them with an optimum balance between information availability versus the cost of its acquisition—see also the Measurement System.

5.5 Go See (Gemba Walks)

The Japanese term gemba literally means "the real place" and refers to the actual place where the value-adding work happens. In the context of software-intensive systems engineering, the gemba is the team room or virtual environment in which the Team works.

The idea behind the Go See habit—also often referred to as Gemba Walk—is that the most significant problems are most noticeable when visiting and directly observing activity at the gemba, which will also lead to the best improvement ideas. By asking judicious questions when visiting the gemba, coaches and leaders will often trigger insights for practitioners, in

addition to developing a very rich understanding of the actual constraints faced by the Teams.

In HP, this habit has been practiced since the 1970s, when it used to be referred to as "management by wandering around"[vi].

5.6 Continuous Improvement (Kaizen Events)

When a Retrospective session is insufficient to fully address the most significant impediment faced by the Team, a Kaizen Event may be called for—see the Retrospective habit.

Kaizen is a Japanese term meaning "evolution toward the better" or "improvement". The Kaizen Event habit refers to the practice of focusing the entire Team's capacity—usually in a facilitated workshop format—in order to design or improve a given process.

Similar to the Retrospective habit, Kaizen Events support continuous improvement by enabling Teams to achieve breakthroughs by identifying, analyzing and addressing the root causes of key impediments and improving the system through focused countermeasures.

6 Summary

Effective use of Agile methods requires a deep commitment to respect for people and continuous improvement. Respect for people is manifested well by providing expert practitioners with diverse opportunities to hone their technical and interpersonal skills and expand their professional repertoire across multiple domains.

Continuous improvement is best sustained through a culture in which teacher managers engage in Team learning and devote themselves to serving their teams, investing strategically in the professional and personal well-being of staff and the communities in which they work and live.

In this whitepaper we have introduced twenty-four habits that, when practiced with synergy and in the spirit of the Agile values and principles[vii], lead to the consistent achievement of superlative business results through optimum adaptability.

Business life is changing faster than ever, and rapid adaptability will continue to be a key characteristic required for thriving in the coming decades, just as it has shaped the course of history from time immemorial. Agile methods provide an excellent means of achieving splendid business results with an optimum balance between return on investment and total cost of ownership.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304
www.hp.com

[i] See http://agilemanifesto.org/
[ii] See http://www.literateprogramming.com/
[iii] See http://domaindrivendesign.org/
[iv] See http://agilemodeling.com/
[v] Learn more about the Theory of Constraints http://amzn.to/TheoryOfConstraints
[vi] See http://bit.ly/Wikipedia-MBWA
[vii] See http://agilemanifesto.org/principles.html and http://bit.ly/ReworkedAgileManifesto