Agilex Methodology

PRAGSOFT
CORPORATION

Document: **Agilex Methodology**

Author: **Sharam Hekmat** (hekmat@pragsoft.com)

Revision: 1.2 (Aug 2009)

Contents

1 Introduction

Agilex is the result of eight years of practical experience in running successful commercial software projects according to agile principles. As such, it's a methodology deeply rooted in practice and common sense.

This document is a practitioner's guide, designed specifically for ease of understanding and rapid absorption. You'll notice that, compared to other methodologies, it's relatively short and succinct. This is mandatory for a genuine agile methodology. If it were any longer, any more comprehensive and complex, requiring you to spend days and weeks to become familiar with it, it couldn't claim to be truly agile!

The term Agilex is a shorthand for 'Agile Express', where 'express' is intended to convey two things: speed and communication.

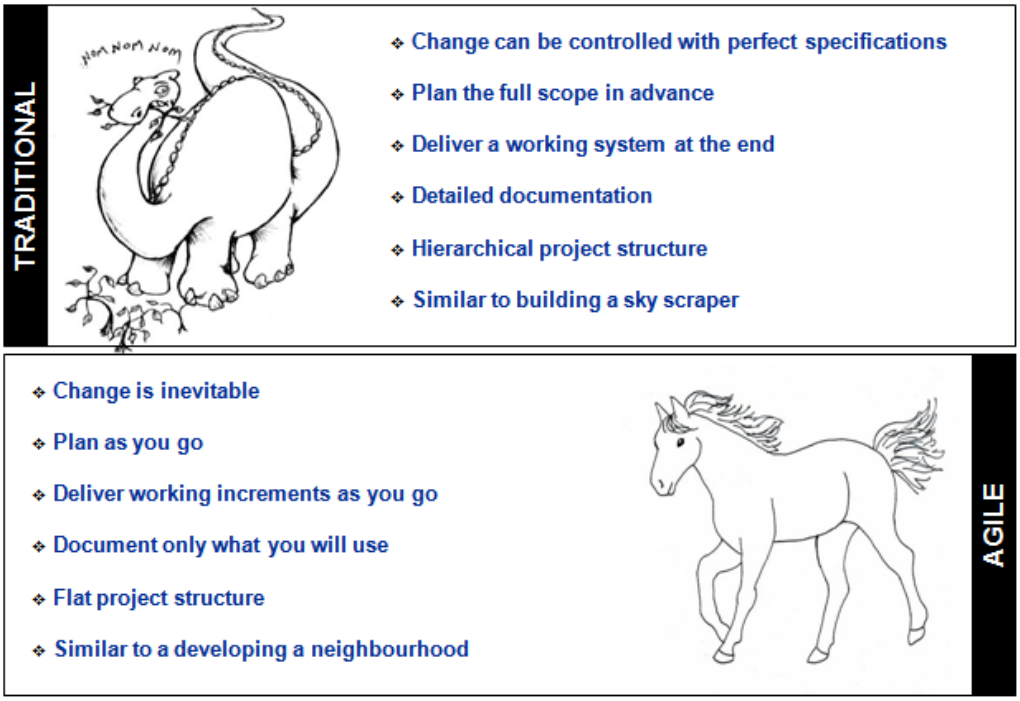
1.1 Why agile?

The shortcomings of traditional, waterfall methods have been known for at least three decades, namely:

- The requirements for a complex system can rarely be specified fully and accurately in advance.
- Despite tight quality control, the output of each phase will contain gaps or flaws that won't be discovered until a later phase.
- User requirements will evolve during the course of a long project, making the end-result inconsistent with the latest requirements.
- The cost of fixing a requirement or design defect discovered later in the project is substantial.
- Given that a working system is not available until late in the project, there is little opportunity for user participation and feedback; this increases the risk of the system not being accepted by the users.
- These challenges often cause schedule delays and budget blowouts.

The agile approach attempts to address these difficulties by promoting a more iterative lifecycle, where emphasis is on prototyping, user participation, having a working system all along, and less documentation. The approach is best summarized by the agile manifesto (*www.agilemanifesto.org*), which places more value on:

- **individuals and interactions** over processes and tools,
- **working software** over comprehensive documentation,
- **customer collaboration** over contract negotiation, and
- **responding to change** over following a plan.



Successful adoption of the agile approach requires the overcoming of cultural, process, and policy barriers.

1.2 Barriers to agile

There are a number of barriers that make agile adoption quite challenging:

- Most established businesses have considerable investment in traditional methods by way of established processes, trained staff, cultural values, and infrastructure. The inertia created by all this makes change difficult.
- Most experienced practitioners have accumulated their experience in traditional settings, so it's not surprising that they view agile methods with suspicion.
- Because agile methods are relatively new, there is a shortage of practitioners, especially seasoned ones who truly understand the principles and can help companies succeed in their transition.
- Most of the technologies available today haven't been designed to make agile development easier. This is, however, likely to change in the future.
- Culture is perhaps the biggest barrier. When an organisation decides to go the agile route, it needs to engage its staff and take them on a journey. This is easier said than done, because this is where behavioural issues kick in that can derail the plan.
- Agile is not regarded by many as a competitive advantage yet, despite its proven potential.

Any organization considering the adoption of agile should be mindful of these barriers and plan to put energy into overcoming them.

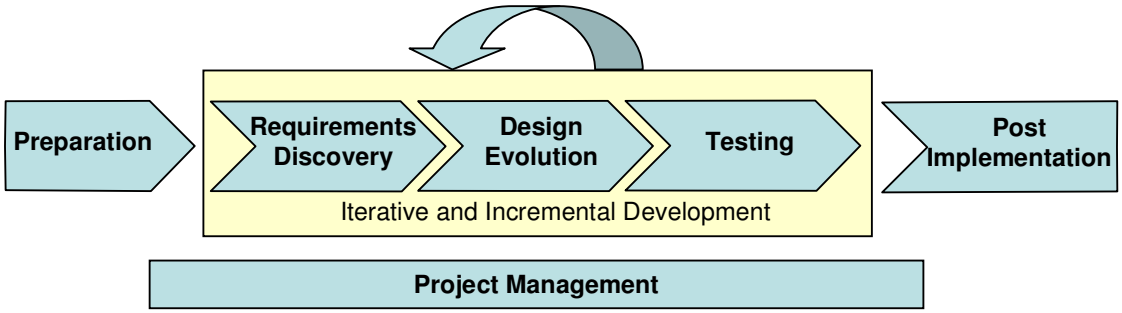
1.3 Agilex

Agilex advocates simplicity in all software development activities. This extends to the definition of the methodology itself. A methodology that over-specifies runs the risk of becoming a straightjacket and a threat to creativity. To avoid this, Agilex favors loose definitions that leave room for interpretation, enabling practitioners to adapt it to their local environment. In practice, no two organizations are the same, so an approach that works well in one organization might prove problematic in another. Practitioners are best placed to consider the dynamics of their own organizations and determine how to best adapt the methodology to their local requirements.

The most significant benefit of adopting a simple methodology is that it can be absorbed quickly. This is a vital point that's often overlooked by IT decision makers. Choosing a comprehensive and voluminous methodology can be quite tempting, but you should resist this temptation because such methodologies are rarely well understood (or remembered) by practitioners, leaving you exposed to deviations, inconsistencies, and non-conformance.

1.3.1 Agilex Software Lifecycle

The Agilex software lifecycle is illustrated below.



The **preparation** phase is concerned with laying the foundation for the project, encompassing activities such as: providing a business case, defining the initial business scope, assessing the suitability of the project for agile development, project prioritization, project team organization, project charter definition, and addressing training gaps.

The **requirements discovery** phase is mainly centered on JAD workshops where requirements are brainstormed. It also deals with scope prioritization and scope change control.

The **design evolution** phase uses prototyping and incremental development to flesh out the system according to an agreed architecture. Intensive coding and unit testing takes place in this phase, with frequent releases.

The **testing** phase provides quality control through activities such as integration, system, and user acceptance testing. Defect management is an integral part of this process, requiring the active participation of developers to rapidly turnaround defect fixes.

The **post implementation** phase tracks benefit realization through post implementation reviews and production support, where production fixes and enhancements are applied during the operation life of the system.

The phases and activities are regulated through a **project management** function responsible for planning, scope management, resource allocation, and so forth.

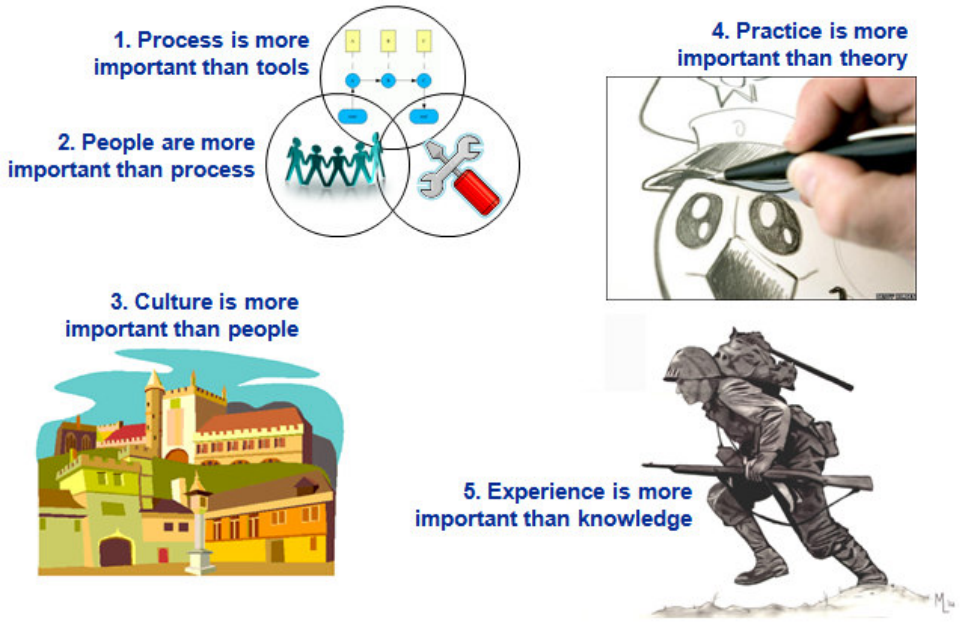
1.3.2 Agilex Principles

Agilex has five agility principles.

- **Strive for simplicity**. You can maximize your chances of success by keeping everything simple, including business requirements, design, environments, build process, deployments process, documentation, etc. Humans don't have an unlimited intellectual capacity; if you take on too much you will more likely fail. Simplicity will not only give you speed, it will also reduce the possibility of mistakes. Most IT practitioners have a tendency to aim for perfection and, as a result, over-engineer things. Don't fall into this trap. It's far better to deliver an imperfect solution that works than to deliver a perfect solution that doesn't work.
- **Don't over-analyze** or, put differently, don't expect to get it right the first time. Treat software development as a learning process – you won't know everything in advance regardless of how deeply you analyze things. Start building with the limited information available and plan to discover the rest along the way. Think of yourself as an explorer and pioneer. You won't know until you explore, and the best way to explore is to act than to just study. The risk, of course, is that you'll make mistakes, but it's through these mistakes that you'll learn and get closer to the truth.
- **Start with a problem not a solution**. Too often a project starts with a proposed solution. This is unproductive and will limit your thinking. Always strive to find out what the true business problem is. Your job is to build a solution to a problem, not to build a solution that someone has asked for. Unless you follow this approach, you run the risk of building an irrelevant solution. Sometimes you even discover that no solution is really needed because there is no real problem.
- **Involve the users**. Never forget that a solution is for the benefit of a group of users. However, most users are not good at adequately expressing their problem and what they expect to be done about it. This often remains to be discovered and, unless you involve them in the process, you won't discover the right solution. It's a bit like modern psychoanalysis – the therapist doesn't work in isolation to heal the patient; they work closely together.
- **Develop incrementally**. Never attempt to build a large or complex structure in one hit, as the most likely outcome will be disappointment and failure. Plan and build one increment at a time, and use the lessons learnt from that experience for the next increment. This will lessen the impact of mistakes and substantially reduce the risk of non-delivery. Each increment also gives you an opportunity to verify your assumptions with the users, so that you embark on the next increment with the confidence that you're on the right track.

1.3.3 Agilex Insights

In addition to the agility principles, Agilex provides five important insights that practitioners are encouraged to keep at the forefront of their mind. These insights help you maintain a balanced view of things.



- **Process is more important than tools**. To verify this, consider a good process for writing a book using pen and paper, versus a poor process involving the most advance word processing tool.
- **People are more important than process**. To verify this, think of a skilled chef following his intuition versus a novice following a recipe.
- **Culture is more important than people**. At the heart of every great organization or great nation, is a great culture. People come and go. They make contributions to the culture and their lives are enriched by it. Culture is all-encompassing and outlasts the people, not the other way round.
- **Practice is more important than theory**. Of course, theory has value because it can guide our actions, but for every grain of theory a mountain of practice is required to beneficially apply it in the real world. There is very little theory behind bricklaying, for example, but to become a skilled bricklayer takes years of practice.
- **Experience is more important than knowledge**. As an example, what kind of army would you prefer to have defending you against an enemy – one that has been formally educated in the art of warfare, or one that has experienced real wars and been battle-hardened?

1.4 Transition from Traditional to Agile

If your organization is using a traditional methodology and is new to agile then you should carefully consider the transition. This section outlines ten recommendations to help you manage the transition.

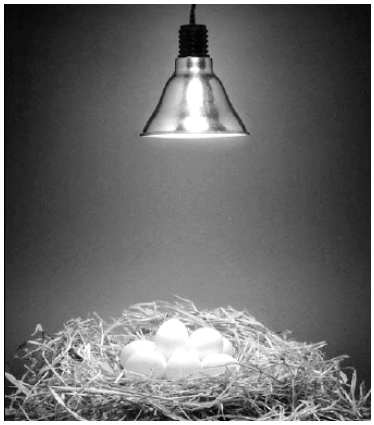
1.4.1 Prove it on a small scale first

If your IT department is of a significant size, it would be unrealistic to expect to rollout agile in one hit. It's unlikely that you'd be able to convince everyone that this is good move. So, instead, you should aim to prove the effectiveness of the methodology on a small scale first. This makes perfect risk management sense, because your mistakes will have a small impact.

This approach is standard practice in the bridge construction industry. A major bridge, once designed, is proven using a small scale model to ensure that it can sustain the anticipated load and environmental factors such as strong winds, earthquakes, and extreme temperature variations. It's much easier, safer, and cheaper to test and rectify design weaknesses with this approach, than constructing the actual bridge and addressing unforeseen issues afterwards.

A good approach would be to pick a motivated team and a manageable project and run it using agile. You may need to do this a few times to develop the capability and iron out any issues.

1.4.2 Select and incubate your 'A' team



The first agile team you put together should include some of your best people. These are individuals who are experienced, capable, open-minded, and motivated to make agile a success. To ensure that their efforts are not derailed, you should incubate this team by relocating them to an isolated area where they can get on with their task uninterrupted. You should also protect them from the normal company bureaucracy and any practices or processes that might hinder their initiative. The team should be given sufficient freedom in determining how to organize its work in accordance with agile principles.

The work of this team can be publicized and shared with the rest of the organization once enough success has been achieved to establish credibility.

1.4.3 Choose a suitable first project

The initial project you pick for agile development is a very important point of consideration. The right balance is vital. On the one hand you want the project to be small enough to be manageable and, on the other hand, you want it to be realistically challenging and have a high profile so that it can have a visible impact.

As an analogy, think of the pyramid entrance to the Louvre museum in Paris. Anyone who's been

to the Louvre would agree that it's a vast museum and, in the scheme of things, the glass pyramid structure that acts as the entrance is relatively insignificant. But, because it's the entrance and it sits in the middle of the Louvre grounds, its presence and profile is undeniable.

When you're thinking about selecting your first agile project, ask yourself, 'what would be the equivalent of the Louvre pyramid in my organisation?' By way of example, a large bespoke development would be a poor choice, whereas putting a web front-end to a legacy application could potentially be a good choice.

1.4.4 Deliver rather than talk

In order to sell your agile methodology to the rest of the organisation, you need to do two things well: you need to communicate and you need to build credibility. Experience indicates that the communication part is nowhere nearly as important as the credibility part. And nothing gets you more credibility than delivering quickly on the promises you've communicated.

Beware of cases where external consultants are brought in, months of talking has taken place, a lot of awareness has been created, but nothing tangible actually delivered. That kind of approach has a tendency to fall flat on its face and cause loss of credibility.

1.4.5 Secure executive support

At some point you need to secure the support of the whole executive team for what you're trying to achieve. This shouldn't be under-estimated because the organisational impact of adopting agile goes well beyond the IT function.

Virtually every department that has a customer-supplier relationship with IT will be impacted by the change. The trust and support that you require of these departments will not emerge overnight, but over the course of months and years.

1.4.6 Aim for a significant cultural change

Going agile isn't simply a change of approach, but a significant cultural change, because it challenges established thinking and requires individuals to behave differently. Cultural change is one of those things that you don't want to leave to chance. You want to make sure that it moves in the direction you want it to move.

To assist the cultural change, you should explicitly state your values and ensure that everyone understands them and their relevance to the agile approach. Agilex recommends the following ten values. Make sure that these values are not just treated as slogans – use them in daily activities to drive desirable behaviors. Every project member must be empowered to refer to these values in order to overcome obstacles. For example, if a staff member believes

that a proposed design is over-kill, s/he should be able to cite the sixth value (over-engineering is an unacceptable practice) and request a simplification.

1.4.7 Be prepared to face a lot of opposition

Humans are not entirely logical creatures – we're more often driven by emotions than logic. A consequence of this is that we tend to resist change, regardless of what the change is about. Change is typically seen as a threat, even when it's beneficial change.

When you attempt to rollout agile in your IT organization, it's highly likely that you'll encounter pockets of resistance. There will be individuals who are comfortable with what they've been doing for years and don't want to try a different approach. The point that you need to get across to these individuals is that even though they're comfortable with their existing practices, the rest of the organization isn't. Make their choices clear – either accept that change has taken place and they need to be a part of it, or simply move on.

1.4.8 Scale up in a piecemeal fashion

Agile methods promote a piecemeal approach to building IT systems. This principle is equally applicable to the rollout of the methodology itself. To mitigate risks, start small and ensure success using a small team and a manageable project. Once this baseline has been established, each subsequent project can be used as an opportunity to rollout the methodology further in the organization. The whole process may take years.

Time is the critical factor in most transitions. The conversion of analogue TV to digital, for example, has been in progress for many years now. This gives everyone time to adapt. It also avoids earlier investments from becoming worthless overnight.

1.4.9 Deal decisively with dissidents

One of the usual obstacles in a cultural change is the emergence of dissidents. These are individuals who do not believe in the change, do not want to see the change succeed, and would actively do anything to ensure its failure.

Don't assume that, over time, dissidents would be washed away by the change. This is a mistake, because it ignores one important fact about dissidents – that they're usually very capable individuals who can cause much harm. So you basically either want them on your side or out of the organization.

1.4.10 Continuous improvement and optimization

Because an IT process involves people, technologies, and various other environmental factors, all of which are subject to change, it shouldn't be viewed as having a static definition.

People learn, technologies evolve, and business conditions change continuously. You want to be in tune with these changes and tap into the opportunities they present.

The agile processes that you implement in your organization are likely to change over the years. As your staff become more skilled and your understanding deepens, you should take the opportunity to improve your processes. However, it's important to note that the basic principles don't change, only your interpretation of them.

2 Preparation

The aim of the Preparation Phase is to lay the foundation for the project by ensuring that all the required ingredients are in place to officially commence the project. Whilst a project is in this phase, no organizational commitment to it is guaranteed – the project remains tentative, until commitment is secured. Alternatively, the project may be put on hold, postponed, or even cancelled, due to organizational constraints or dynamics. For example, the Preparation Phase might provide strong evidence that the project's benefits are unrealistic or insufficient to justify the required expenditure.

2.1 Business Case Process

Projects that exceed a certain size (e.g., $50k) should be justified through a formal business case. The aim of the business case is to furnish sufficient information to enable decision makers to determine whether to approve the project and, if so, to determine its priority relative to competing projects.

A business case must articulate the following.

- The broad **scope** of the project (i.e., the business problem that it will tackle). A business case that proposes a solution without clearly identifying the business problem that it addresses is, by definition, flawed.
- The **alignment** of the project to the business strategy (i.e., how it will contribute to the achievement of the business strategy). For example, by delivering a new product to the market ahead of the competition.
- The expected **benefits**, expressed in quantified (monetary) and qualified (non-monetary) terms. For example, dollar value of additional sales generated, improvement in customer satisfaction score card, reduced downtime, improved operational efficiency, reduced error rates and compensation. Because quantified benefits tend to be ballparks, it's recommended that they are expressed in low/median/high format.
- The **estimated** resources required (i.e., staff, infrastructure, software licenses) and their respective costs. Again, these are best expressed in range format. Also, most organizations find it useful to identify the opportunity cost percentage of the overall cost.
- The **risks** associated with the project and how they will be mitigated.

Additionally, a business case must identify its sponsor, stakeholders, and approval list. Before distribution, the business case should be socialized with the relevant parties to solicit feedback and ensure clarity and completeness. The recommended length is around 5 pages.

A formal review meeting is recommended to provide an opportunity for collective discussion prior to submission for approval.

2.2 Suitability Assessment

The agile approach is not equally applicable to all projects. Therefore, it's worthwhile considering the suitability of the agile approach before making a decision to adopt it for a particular project. The following factors should be used as key indicators.

2.2.1 Requirements Volatility

This is by far the most important factor. The suitability of agile increases proportionally to requirements volatility. This is because agile is an effective process for requirements discovery, bringing potential issues to the front of the project rather than leaving them to be discovered later on. As a result, a project that attempts to break new grounds, deliver a new product or service to the market, or deliver an innovative process is ideally suited to the agile approach.

2.2.2 Shelf Life

The suitability of the agile approach is independent of the anticipated shelf-life of an application. However, an application with a relatively short shelf-life (a year or two) is far better served by an agile rather than a traditional approach due to superior development speed and lower cost of construction.

2.2.3 Mission Critical

Mission and life critical systems are generally not suited to the agile approach. Examples include: flight navigation systems, air traffic control systems, combat and warfare systems, space exploration systems, most medical devices, onboard car engine management systems, and the like. These systems typically have stable requirements and need to be developed to exact engineering standards with little or no margin for error. These constraints are better served by a traditional, waterfall approach.

2.2.4 Business Critical

Major business systems often have critical components that can expose the business to significant risks. The transactional components of a core banking system or a share trading system serve as good examples. Such components typically reside within application servers and handle significant transaction volumes. Due to their operational and risk constraints, these components are best developed using traditional methods.

2.2.5 Time To Market

Where time to market is critical, the incremental delivery style of agile provides the best chance of minimizing the development cycle. Releasing a new product quickly to the market (even though it might not be complete) allows you to get rapid customer feedback so that you can tune the product evolution path to market response. The result is often quicker customer acceptance and rapid evolution.

2.2.6 Scalability

Scalability is one of the most complex dimensions of application development. Consequently, it needs to be built into the design rather than left as an afterthought. For example, consider two web sites – one that handles a hundred transactions a day, and another that handles a million transactions a day. The server-side design of the latter would be considerably more complex than the former in order to handle the volume. Whilst the front-end of both sites would be ideally suited to agile, the performance critical server-side components of the second site would be best designed using traditional methods.

2.2.7 Target Environment

The target environment of a system ranges from embedded (e.g., a microprocessor embedded in a washing machine) to customer-facing (e.g., an Internet banking system). Those parts of the system that are closer to the user are generally better suited to agile methods. Specifically, new business applications with substantial user interfaces tend to require many iterations to evolve into something intuitive and usable, making them ideal agile candidates.

2.2.8 Technology

Most established businesses tend to have operational systems that span decades, ranging from aging mainframe systems to modern Internet applications. Regardless of their underlying technologies and original development paradigms, all these systems need to be maintained. The older systems (running on older technologies) tend to benefit less from an agile approach due to the technological limitations. The agile approach relies on rapid prototyping and rapid development cycles, neither of which fits well with older technologies. For example, some of these systems impose build cycles that take many hours, making agile impractical. However, those elements of the agile approach that are technology independent (e.g, JAD) can still be beneficially applied to these systems.

2.2.9 Integration

Integration has been a controversial area as far as agile practices are concerned. Some practitioners claim that agile is not suitable for large integration projects due to the technological complexities involved and the myriad of interdependencies. Others hold the view that agile is superior for such projects due to its power for untangling complexity. In practice, the suitability of agile for such undertakings is best analyzed on a case by case basis.

2.2.10 Transformation

A common theme within the business world is transformation, which often has a large IT component. Typically, the aim of an IT transformation is to modernise a set of older systems and replace them with a smaller number of better integrated systems that deliver greater operational efficiency, better customer service, and reduced maintenance costs. Where the ‘as is’ and ‘to be’ states are significantly different, transformation projects tend to have very complex requirements. This in turn makes them highly suited to the agile approach, enabling the requirements to be ‘debugged’ early in the project.

2.3 Project Prioritization

Good project prioritization ensures that an organization focuses its energy and capital on projects that deliver the best net benefit potential. There is often a direct link between the cultural maturity of an organization and its approach to project prioritization. In a dysfunctional organization, projects with sponsors that have the loudest voice, greater political connections, or simply more clout tend to get priority. In a mature organization, priority tends to be largely based on business case merit.

Assuming that the organization has an inclusive culture, project prioritization can benefit from the following measures.

- Having a Priority Forum that meets regularly (e.g., monthly) to review project priorities. The forum should be attended by senior business stakeholders and IT senior management

who have in-depth understanding of the business. To make discussions manageable, impose a maximum of around 10 members.

- Maintain a level of stability around project priorities for each calendar year. If the priorities change every month, staff will get confused and their energy will not be optimally used.
- Almost every business has more projects than they have budget for. Make it a rule that projects that fall below the budget threshold, are parked, deferred, or abandoned. In the long run, this will result in good financial discipline. The underlying cultural message should be: we won't implement every good idea, just those that we can afford *and* get the most reward from.

2.4 Project Team Organization

Agile project teams differ from traditional teams in a number of respects:

- Agilex favors **smaller teams** due to reduced communication overheads. Practice indicates that project teams of up to 10 developers are manageable. If you assume a similar number of testers, a few analysts, a project manager, and a few part-time business representatives, we arrive at a combined upper limit of about 30. However, average size projects typically involve no more than a dozen staff in total. By comparison, traditional projects may involve hundreds of people, but run the risk of underutilization, miscommunication, duplication of effort, and reduced productivity.
- Agilex recommends a relatively **flat project structure**. For examples, developers typically report to a lead developer who might also act as the technical design authority. Traditional projects, on the other hand, tend to have deeper structures and more formal reporting lines, which substantially increase the communication latency.
- Agilex assumes that the team is composed of **high caliber individuals** who are not only disposed to the agile approach but can also operate to its demanding dynamics. Put your best people on agile projects because this is where they can truly thrive and deliver outstanding performance.
- Agilex works best when the project team is **co-located**. By having everyone in close proximity, you further optimize the communication channels and reduce the need for meetings, workshops, etc. Bring the team together from inception, allowing informal relationships to develop quickly. These informal relationships will deliver more value to the project than formal ones.

Agile projects are ideal training ground for developing future agile practitioners, so use each project to train a couple of new recruits. Make sure that each new member with no prior agile experience has a mentor within the project team.

Wherever possible, remove any overheads that could slow down your agile project team. For example, traditional projects often impose a strict timesheet reporting process. This will only frustrate your agile team and reduce their productivity.

2.5 Project Charter

Your business project manager must provide a brief project charter shortly after commencement. The charter provides the terms of reference for the project by defining the following.

- The authority assigned to the project manager
- Business reasons for undertaking the project
- Project objectives and constraints
- Key stakeholders
- The adopted development methodology
- Paths of investigation to arrive at a solution
- Change control policy

Make the charter as brief as possible by summarizing the above and referring to documentation available elsewhere.

2.6 Training

For optimum productivity, you need to ensure that the project team members have adequate training to fulfill their roles in the project. Broad areas of training include:

- **Business knowledge**. Do your team members know enough about the business domain of the project? This is by far the most valuable commodity. No amount of technical knowledge can compensate for a lack of business knowledge.
- **Systems knowledge**. Do you team members have adequate understanding of existing systems that will be impacted by the project? Lack of systems knowledge will result in re-inventing the wheel and potentially introducing flaws and inconsistencies.
- **Technological knowledge**. Do your team members have a good grasp of the technologies to be used? This is typically the most widely available commodity and often over-valued. You need some, but it's not everything that you need.
- **Process knowledge**. Do your team members know the various processes they'll be engaged in (JAD workshops, build process, deployment process, test process, change request process, etc). Accessible and reliable documentation on these processes can substantially reduce the need for training.

The most effective form of training is 'on the job', but the agile capacity for this is limited. Therefore, you should aim for a combination of formal training from quality sources, followed by 'on the job' training.

Identify your training champions in your projects and use them effectively. Some individuals are naturally more approachable and have a knack for passing on their knowledge to others. Reinforce their behavior through appropriate reward and recognition.

3 Project Management

Make sure that every project has a **business project manager** and never combine this role with technical management. The technical manager (typically, the technical lead) will have too much technical work on his plate to be able to do the duties of a business project manager. Scope management, stakeholder management, planning, budgeting, project reporting, supplier management, handling change requests, and the like are the responsibilities of the business project manager and should not be imposed on the technical manager, or else the real duties of the latter will suffer.

Before appointing your project manager, make sure that s/he understands agile principles *and* believes in them.

3.1 Project Planning

Traditional methods put significant effort into up-front project planning and estimation. In Agilex, planning is more streamlined and focuses largely on addressing dependencies. Specific areas of focus include:

- Assembling the right team and ensuring that training needs are addressed.
- Arranging accommodation for co-locating the team.
- Allocating meeting rooms for conducting JAD sessions and ensuring that they're sufficiently equipped.
- Procuring the products and services that the project will depend on (e.g., computing resources, software licenses, third-party services).
- Setting up appropriate environments for development, build, system test, UAT, and later on, production.
- Organizing access to the relevant systems to be worked on by the project team.
- Organizing version control tools, environments, and processes.
- Identifying points of contact for external dependencies, communication channels, and escalation procedures.
- Assembling the project steering committee.
- Publishing an initial draft project plan. The project plan will be an evolving entity and never set in concrete. Avoid breaking down tasks to a low level of detail (e.g., expressed in hours). The dynamics of the project will make micro planning meaningless.
- Organizing project review meetings.

Good project management practices are as vital in agile as in traditional projects.

3.2 Estimation

Traditional methods require detailed estimation of project activities and tasks so that detailed plans and schedules can be produced. Agilex subscribes to a different view: detailed estimation of projects costs too much! The point being that most projects go through such

level of continuous change that it becomes prohibitively expensive to keep the micro plans up to date. By keeping the estimates at a high level, you'll be able to revise them much more quickly.

Because of the incremental development style of agile, you can focus your estimation on the current increment and simply produce ballpark estimates for subsequent increments. The rationale is that until the current increment is completed, there is not enough baseline data to meaningfully plan subsequent increments. As you progress through the increments, the accuracy of your estimates increase because they benefit from the experience gained from earlier increments.

To adhere to delivery timeframes, use **time boxing** – fix the length of each cycle and the team size but allow scope reduction in order to meet the timeframe.

3.3 Steering Committee

The role of the project steering committee is to 'steer' the project in a direction consistent with the business strategy. Limit the steering committee to the sponsor, key business stakeholders, the project manager, and the technical lead. Critical projects may also require CIO attendance. Keep the number of attendees below 10 to make discussions manageable. Aim for monthly meetings of one hour duration.

The project manager must do much of the preparatory work for the steering committee meetings. Typical agenda may be:

- Review of minutes from previous meeting
- Review of action items
- Project update
- Change requests requiring approval
- Any other business

As a rule, if the project manager intends to table a major issue or problem, s/he should come prepared with a list of options for addressing the problem. This works best when the project manager has taken the time to socialize the options before the steering committee.

The steering committee must not be used as a forum to devise solutions to problems, but to review available information and to approve or reject recommendations.

4 Requirements Discovery

Traditional methods use the term 'requirements specification', implying that business requirements can be reliable specified through analysis. Agilex uses the term 'requirements discovery', implying that exploration (not just analysis) is required in order to discover the true business requirements.

4.1 JAD Workshops

In Agilex, the primary medium for discovering business requirements is Joint Application Design (JAD) workshops. These workshops are more effective when they revolve around a working prototype or, at least, a mock up. The input to the very first JAD session is a brief statement of the business requirements, solicited from the stakeholders and/or intended users of the system. Aim for a couple of pages, but more is acceptable if the project is an enhancement to an existing system.

JAD sessions should be attended by:

- The technical lead
- One or two developers who undertake the prototyping effort
- A business analyst and/or subject matter expert
- A system tester
- A scribe whose role is to take notes

The very first JAD workshop is a brainstorming session, intended to generate enough ideas for the initial prototype to be built. Subsequent JAD sessions will use the prototype to conduct the discussion. JAD works best when the workshops are run informally and individuals get a chance to express their views without the fear of criticism or ridicule. After each workshop, the scribe will type the notes and distribute them to the attendees, highlighting areas where agreement has been reached.

The frequency of the workshops depends on the project size, complexity, and timeframe. During the Requirements Discovery Phase, weekly workshops are recommended. During the Design Evolution Phase, fortnightly or even monthly workshops may be appropriate.

Large projects often require multiple JAD streams, where each stream focuses on a key part of the system.

Newcomers to agile would benefit from JAD training so that they're fully aware of the format, dynamics, and expected behaviors.

4.2 Documentation

The approach to documentation is one of the key points of differentiation between agile and traditional methods. In Agilex, there is a significant shift away from comprehensive documentation and towards working software. The argument is that having access to working software early in the project and observing its evolution during the course of the project

significantly reduces the need for comprehensive documentation. In other words, working software is a more powerful medium of communication than written words.

Consequently, we have to be much more selective about what we document and what we don't. The following principles will help you in making wise choices.

4.2.1 The 'useful document' principle

Before you write a document, ask yourself: will anyone read this document and, if they do, will they find it of any value? If the answer to either question is no, don't waste your time.

4.2.2 The 'document shelf-life' principle

Think about the potential shelf-life of a document you're about to write. If the document's subject matter is highly volatile then its shelf life will be very limited. Unless you're prepared to put in the continuous effort needed to keep the document up to date, there is not much value in taking a one-off snapshot of a moving target.

4.2.3 The 'skip the obvious' principle

IT project documents often fall into the trap of striving for completeness. When writing documents, carefully consider the target audience and ask yourself: what are they likely to know and what are they likely to want to know? By focusing on the latter, you'll reduce the documentation effort and increase the reader's productivity.

4.2.4 The 'single point sensitivity' principle

Project and system components that are heavily dependent on the knowledge and skills of a single individual are prime candidates for documentation. Consider the impact of losing a key individual and ask yourself: what level of documentation will I need in order to maintain continuity?

4.2.5 The 'information duplication' principle

Too often the same information is captured in multiple documents without adding any further value. For example, a low-level design document that adds no further information that can be gleaned from the code, is a redundant document. In practice, information duplication leads to inconsistency (e.g., the code is modified but the design document isn't). This will mislead readers and ultimately result in loss of confidence about the document's accuracy.

4.2.6 The 'educate, not market' principle

IT project documents are mostly technical. The aim is to educate the reader about something important so that they can effectively do their role in the project. The aim should not be to sell or impress, so the style and the language should focus on education rather than marketing. For example, a paragraph that boasts about the superiority of a design without conveying any useful insights about the design would be better suited to a marketing brochure.

4.2.7 The 'Occam's Razor' principle

This principle implies that the simplest way of expressing something is the best. In practical terms, if you can say something in one sentence, don't say it in three. In writing, this

principle is used to eliminate phrases, sentences, or even entire paragraphs that don't add to the information content of the document.

4.2.8 The 'appearance is not content' principle

A reader's first impression of a document is formed by its appearance – how many pages, appeal of the cover, typesetting, etc. Once the reader starts reading the document, this is replaced by the impact of the actual information content. This is where the true value resides, so the bulk of the writing effort should go into making the content as valuable and as accessible as possible.

4.2.9 The 'less is more' principle

A document should always consider the likely constraints of the reader. Consider how much time the reader is likely to have to read the document and pitch it at that level. A busy executive can't find the time to read a 50-page business case, but will most likely read a one-page executive summary. Put differently, the true value of a document is the extent to which it's actually read and understood, as experienced by the target audience, not how it might be perceived by an ideal audience.

4.3 Scope Prioritization

Just as the proposed projects in an organization's portfolio need to prioritized, the scope of each project must be subject to careful prioritization. The 80/20 rule is the best guide for scope prioritization. Priority should be given to business requirements that:

- are core to the system,
- can be built relatively quickly, and/or
- influence the design of the rest of the system

For example, in a corporate reporting system, identify those reports that are frequently used and give them priority. Defer the less frequently used report to later increments and be prepared to de-scope them completely if you run out of time. The underlying rationale is that it's better to deliver an incomplete but working system than to strive for a complete system that doesn't quite work or takes too long to make production ready.

The project manager must maintain a register of project scope where each scope item and its relative priority is specified, and obtain sponsor and stakeholder approval for the register. Furthermore, the scope register must identify the planned target increment for each scope item. Each increment must be time-boxed. If you run out of time during an increment, don't extend the timeframe; simply de-scope the lower priority items so that you can deliver the increment on time. De-scoped items may be deferred to later increments or completed removed from the project scope.

4.4 Change Control

A disciplined approach to change control is vital to successful project management, regardless of the methodology used. Because of uncertainty and ambiguity in business requirements, change requests often need to be raised to adjust the scope. It's not uncommon to discover that a major requirement or design constraint has been overlooked, thus highlighting the need for a scope change.

During the Requirements Discovery Phase, an agile project readily allows for change, so you don't really need to tightly manage scope during this phase, but simply keep it within reasonable bounds. During the Design Evolution Phase, however, you must formalize your approach to change control. It's the responsibility of the project manager to establish and manage the change control process for the project.

To request a change, the requester must complete and submit a change request form, which identifies:

- The nature of the change
- The reason for the change
- Its relative priority
- Its impact (both in terms of doing or not doing the change)

Change requests must be submitted to the project manager who will socialize them and table them at the project steering committee, together with impact assessment and a recommendation.

Once a change has been approved, the project manager will have it estimated and planned for delivery in an increment.

5 Design Evolution

Design is performed at multiple levels. At the highest level, the architecture of the system is defined and, at the lowest level, the algorithms that the system will implement. Within this spectrum, only the architecture must be defined in advance. Other aspects of the design are addressed within the scope of each iteration.

5.1 Architecture

Traditional methods take a very rigid approach to architecture. Typically, this responsibility is assigned to a central architecture team that, after some analysis, will recommend an architecture that's consistent with the mandated corporate guidelines. This is then handed over to the project team who will be responsible for implementing it.

In practice, this approach is very problematic. Often the prescribed architecture is flawed and un-implementable. Because the architects have the design authority but lack the accountability for implementing the system, a major discontinuity emerges where developers have to pretend that they conform to the architecture and covertly find a way of twisting it to deliver a working solution.

The lacking element in this situation is architectural proof. Agilex requires that a proposed architecture must be proven before it can be accepted as a design blueprint. It favors a project organization where the design authority *and* accountability reside within the project team. A proposed architecture must be verified using a proof of concept. For example, if the proposed architecture involves the use of a third-party package, the vendor must be required to undertake a proof of concept within the project context to prove the viability of the proposed design.

The recommended organizational approach is not to have a centralized architecture team but to decentralize this function and, instead, use architecture forums to ensure consistency and continuity. Avoid employing architects who spend all their time putting diagrams together and are never actually required to implement them. The best architects are those who design, code, and test.

Organizations with a mature agile track record typically have proven architectural frameworks that new projects can simply pick and run with. These frameworks emerge through successful projects that deliver quality systems that have evolved through numerous iterations.

5.2 Prototyping

Prototyping is by far the most powerful design method available to IT practitioners. It recognizes the cognitive limitations of human beings in dealing with the vast complexity and level of detail found in modern information systems. It can be used to explore the validity and impact of design ideas, visualize difficult concepts, solicit meaningful user feedback, untangle dependencies, and learn by doing rather than reading and talking. Prototyping is powerful because it's based on practice rather than theory. It's a well established and indispensable practice in other engineering discipline (e.g., car design and manufacturing), yet its benefits have been largely overlooked in the software industry.

Even artists recognize the value of prototyping. Study the works of any classic painter and you'll find that behind each masterpiece, there is a stack of sketches that explore the possibilities of the final work. Humans are capable of producing amazing artifacts, but rarely in one go. Elaborate works are invariably developed in layers, in manageable chunks, where there is ample opportunity to recover from mistakes and apply the lessons to later stages.

Proponents of traditional methods treat prototyping with skepticism and even regard it as a form of hacking. This is unfortunate. There is nothing wrong with having multiple goes at something, with the intention of improving your understanding until you know how to do it properly.

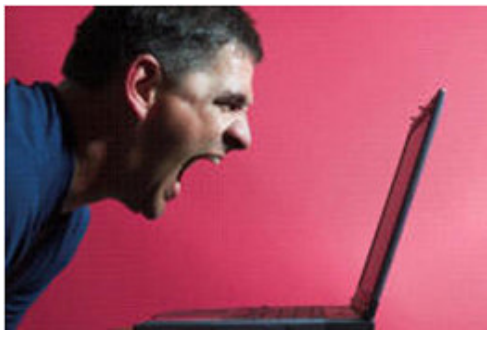
Agilex development relies heavily on prototyping. This is the vehicle for running JAD sessions, for discovery, and for fillings the knowledge gap. To do JAD well, you should opt for technologies that facilitate prototyping. Software development tools have improved considerably over the last 20 years – they've become more visual, more dynamic, and more supportive of the prototyping style. There is also an emerging trend in application programming languages away from the 'compiled' paradigm towards the 'interpreted' style. The latter are far better suited to prototyping, as they do away with time consuming build cycles and facilitate rapid iterations.

5.3 Principles of Good Design

Good design is important in every facet of life, not just information systems. It increases the value of the product by making it accessible and comprehendible, thus improving its chances of acceptance. Unfortunately, most IT practitioners don't have a clear view of what constitutes a good design. Their views are often marred by misinformation spread through advertising and marketing hype. There is also a natural tendency to confuse good design with making an impression through complexity.

Agilex recommends six principles for good design practice.

5.3.1 Technology serves human, not the other way round



Too often people blame themselves for the shortcomings of technology. For example, when browsing a poorly designed web site, the user might think, 'I must be stupid; I can't find what I'm looking for.' User frustration and disappointment are invariably due to poor design. The failure of the designer is to expect the user to conform to the way the system behaves, rather than understanding how the user is likely to behave and designing the system to that model.

Good design focuses first on the user, not the technology. The real hard work is in understanding the user's mental model. In this sense, poor designs are works of lazy or arrogant designers who can't be bothered to invest the time and energy to understand the user. Agilex encourage you to focus on this by involving the user early and validating your

assumptions through prototyping and experimentation. As a result, technology becomes just a tool, not the focal point.

5.3.2 Design is not art



Art is about personal expression – it's about the artist. Observer activity is not required, only appreciation. Design is about use – the designer needs someone to use (not just appreciate) what they create. Designs must help solve human problems, or else they fall into disuse. The highest credit we can give a design is that it's well-used, not that it's beautiful.

Because technology changes quickly, so must design. A design that worked well ten years ago might be totally useless today (e.g., floppy disks). By contrast, great art is timeless and always in style (e.g., Michelangelo's David). Sadly modern artists and designers often confuse these notions and thus abuse their profession.

5.3.3 The experience belongs to the user

Designers do not create experience, they create artifacts for users to experience. However, this doesn't rule out innovation (i.e., leaping beyond the expected). The experience of a design doesn't happen simply because the designer says so, it happens when a user actually reports it.

Designs that assume certain user experience without validation, risk failure. Therefore, prototypes, beta trials, and field tests are always worthwhile activities.

5.3.4 Good design is invisible


Great designs tend to be ones that we take for granted – they work so well that we forget the enormous creative effort that has gone into realizing them. Some great designs are so simple that they appear as obvious (e.g., spoon). Yet, at some point in history, they weren't. Other great designs are so complex and yet easy-to-use that we tend to ignore their long and complex history of evolution (e.g., automobile).

Bad designs are obvious because it hurts to use them. Examples are: a couch with poor back support, a programmable VCR that no-one can program, an electronic gadget that allows you to plug the power supply into the wrong socket.

5.3.5 Recognition, not recollection

A good design draws upon familiar metaphors so that its use is recognizable A bad design requires recollection – you need to memories how to use it. Examples include:

- Classic architectural styles follow recognizable patterns – when you enter a building, you recognize the floor-plan.
- Modern architecture ignores the value of recognizable patterns and creates confusion.
- Some companies, rather than making their web sites more recognizable, regularly redesign them and confuse returning customers.
- How confusing would a dictionary be if it arranged words other than alphabetical order?

The designer's temptation 'to be visually different' must be resisted. Go for familiar patterns that have proven themselves over millennia and focus on subtleties that could be misinterpreted. Make the end product as intuitive as possible. Users quickly get over the initial dazzle of a new product and ultimately form their view on the ease with which they become familiar with the product.

5.3.6 Simplicity is the ultimate sophistication

Simplicity is about balance: knowing what to keep and what to throw away. It's like magic when it works, because none of the complexity is transferred to users – only simplicity.

Simplicity is the toughest and the highest achievement of a designer. Without actual experience, users tend to perceive complexity as sophistication. With experience, the trend is permanently reversed.

5.4 Best Practice

You can improve the speed and productivity of your agile projects by systematically collecting, cataloguing, and publishing best practice recipes. Generic best practices (e.g., design patterns, reusable components, packaged services) can be sourced commercially or through open source forums. Instead of re-investing the wheel, invest the time to research these – they can save you tons of effort.

There will also be best practices specific to your organization and its way of doing business. You can't procure these, but you can invest the time and effort to build them over time. The

idea is to avoid redoing the same thing in each project you undertake. For example, if you've developed an effective way of sourcing data from your backend systems that deals with issues such as overlaps and inconsistencies, capture this knowledge for use in future projects. A great way of doing this is to setup a wiki and to encourage staff to capture their learnings. An even better way is to develop it into a reusable software component or service.

Newcomers to the project or your organization will truly appreciate the best practice material made available to them. This will reduce their learning curve, improve their productivity, and help them integrate into the project/organization smoothly.

5.5 Documentation

The documentation principles outlined under the Requirements Discovery Phase equally apply to the Design Phase. However, expect to write less documents in this phase and spend more time coding and testing.

5.6 Reviews

There is a lesser need for design reviews in agile than in traditional methods because, due to its iterative nature, agile validates designs on an ongoing basis. However, in some cases, holding design reviews would make sense. Examples include:

- Invisible parts or aspects of the system (e.g., server logic) that would have significant operational impact. For example, a commit that updates multiple corporate databases would need to reviewed to ensure that it's sound.
- Certain non-functional requirements (e.g., a batch job that's required to complete within a tight window).
- Interfaces to third-party systems that can't be readily tested (e.g., because the third-party system is under development and not operational yet).

Outside these, Agilex also encourages peer reviews, where one developer asks another developer to informally review his/her work. Another powerful form of reviews is for a developer to walk another developer through his/her work and try to explain how it works. The act of verbally articulating your work is likely to lead to the discovery of gaps, shortcomings, or inconsistencies that you would otherwise have overlooked. These flaws are more often discovered by the person doing the explaining than the observer!

5.7 Version Control

Version control is a mandatory requirement for any software project, regardless of the project size, team size, or methodology. Software systems are subject to numerous revisions and without version control it's almost impossible to keep track of the artifacts that comprise any given release. The need for disciplined version control increases proportionally to the team size.

It's recommended that you appoint one of your experienced developers as the version control administrator. Ideally, this person has a good knowledge of the version control tool, knows how to organize a logical structure for the project, can control branches, and is able to provide advice on complex merges. The administrator will also control user accounts and permissions for accessing the version control system.

The process of extracting source code from the version control system to produce a build should be script-driven and automated. This will allow builds to be generated rapidly – a fundamental requirement for agile development.

5.8 Release Management

You also need a release manager. This may be the same person as the version control administrator, but in larger projects this tends to be a full-time role. The release manager is responsible for the following tasks.

- Planning a release of the system, including the release scope, the artifacts that comprise the release and their correct versions, and the target environment for the release.
- Working with the infrastructure team to ensure the availability of environments and computing resources for forthcoming releases.
- Scheduling the release.
- Creating a run sheet for the release to clearly identify who does what, when, and in what sequence.
- Success criteria for the release in form of tests to be performed and what to expect.
- Procedure for backing out a release if it fails its success criteria.
- Compiling the release notes.
- Communicating the release and its supporting material ahead of its delivery, as well as communicating its outcome afterwards.
- Keeping a register of planned and completed releases and outcomes.
- Publishing release metrics on a quarterly or annual basis.
- Identifying and implementing improvement opportunities to enhance the quality of the releases.

6 Testing

Like design, testing is performed at multiple levels. As we go from lower level tests to higher level ones, the need for formality, documentation, and sign off increases. For any given project, system testing and UAT must have clear exit criteria (for example, no outstanding priority 1 or 2 defects) as well as formal sign off.

6.1 Unit Testing

Unit testing is the responsibility of developers and is performed during the Design Evolution Phase. Each developer is required to unit test the code they write. Unit testing does not require maintainable documentation. Minimum criteria for unit testing are:

- The code must compile or load (depending on whether the programming language is compiled or interpreted) without errors.
- The code must be executed using test cases that exercise typical execution paths.
- The code must be executed using test cases that exercise the boundary conditions.

6.2 Integration Testing

Integration testing is also the responsibility of developers and is performed during the Design Evolution Phase. Preference is for incremental integration rather than a big bang approach. As each new component becomes ready, it should be integrated with the parent application and tested. Like unit testing, there is no need for maintainable documentation of the testing process. Where fine grain incremental integration is used, it would make sense to combine unit and integration testing. However, this is not always possible (e.g., when part of the application is developed by a third-party). Minimum criteria for integration testing are:

- Execution of test cases that exercise the entire interface between the integrated components.
- Execution of test cases that invoke exception conditions in the interaction of the integrated components.
- Where integration crosses organizational boundaries, the test cases must be documented and test results recorded.

6.3 System Testing

System testing is the responsibility of the test team. This team must be separate from the developers and have defined practices and processes to perform thorough system tests. System tests require documentation that includes test strategy, test cases, as well as test results, and may cover the following types of tests.

- GUI testing
- Usability testing
- Performance testing
- Compatibility testing
- Load testing

- Volume testing
- Stress testing
- Security testing
- Scalability testing
- Capacity testing
- Regression testing
- Installation testing

It's highly recommended that a test case management tool is used to organize the test cases. This has the added benefit of making the test cases reusable for future increments.

6.4 Regression Testing

Regression testing is a type of system testing intended to ensure that the system has not regressed after introducing change. When changes are made to a system, it's not sufficient to just test the changes. The changes may have had inadvertent and undesirable effects on areas of the system that haven't been changed. A regression test suite is used to assess the overall health of the system. The scope of regression testing should be established by considering the potential impact of the change. To do this economically, a subset of the regression test suite should be used such that: (i) it fully exercises the area that has changed, and (ii) it broadly or randomly exercises the areas that haven't changed. If the tests reveal unanticipated issues in the areas that haven't been changed then additional testing may be required.

It is highly recommended that you automate regression testing as much as possible. For maximum productivity, test automation tools may be used to enable an entire regression test suite to be run without human intervention.

6.5 User Acceptance Testing

User acceptance testing is the responsibility of the business, the purpose of which is to validate that the solution meets the business requirements. Like system testing, UAT should follow a formal process and have its test cases and test results documented.

Close cooperation between the system test team and the UAT team will be beneficial. This will enable the UAT team to benefit from the established disciplines of the system test team, reuse some of their test cases, and utilize their testing tools to do their job productively.

6.6 Defect Management

Agilex requires the process for raising, allocating, fixing, and re-testing defects to be as streamlined as possible. Rapid turnaround of defects enhances the productivity of the test team, ensuring that they won't be idle, waiting for fixes to crucial defects. Use of a defect tracking tool is mandatory. A typical process involves these steps:

- A tester discovers a defect, records it in the defect tracking tool by providing information that includes: summary description of the defect, its severity, its priority, how to reproduce it, reference to the test case that led to its discovery, and any other supporting information (e.g., screen shots).

- The lead or a nominated developer regularly reviews the outstanding defect list and assign them to developers that have the most familiarity with that area of the system.
- The assigned developer opens the defect, studies it, and takes action to remedy it. The developer then assigns the defect back to the test team, indicating the build in which the fix is available for re-testing.
- Each build carries release notes that list the defects that the build rectifies.
- The assigned tester re-tests the defect and either accepts the fix or rejects it. A defect with an accepted fix is closed. A defect whose fix is rejected is re-assigned to the development team.

Make sure that defect severity and priority scores are not misused. Use simple and unambiguous scoring criteria. For example:

- Defect severity
  - Critical (1): renders the application unusable (e.g., it freezes or crashes)
  - High (2): incorrect behavior that prevents the execution of further test cases
  - Medium (3): incorrect behavior, but doesn't prevent further testing
  - Low (4): cosmetic issue (e.g., reporting format requires improvement)
- Defect priority
  - High: requires urgent attention
  - Medium: can be addressed later in this increment
  - Low: can be postponed to a future increment

The project manager should regularly generate reports out of the test case management and defect tracking tools to provide an overview of how the test and fix efforts are tracking. Simple trend graphs that show progress against the plan will suffice. Avoid generating very elaborate reports that could take too much time to study. Steering committee members should be able to look at these graphs for a few minutes and get a fairly accurate picture of the trend of the project.

7 Post Implementation

Once a system passes the UAT exit criteria, it's deployed into the production environment and made available to users. The objective of the Post Implementation Phase is to ensure that intended benefits are realized and that the system remains operationally useful, serving the needs of its users well.

Post implementation provides a limited scope (in form of production support) for fixing defects and applying minor enhancement. Major system enhancements or extensions must not be attempted in this phase, but rather run as a separate project in its own right.

7.1 Post Implementation Review

A Post Implementation Review (PIR) is a formal review that attempts to identify and capture learnings from a completed project. Focus should be on capturing what's worked well and what hasn't, and how the lessons can be applied to future projects. It should never be used as a channel to lay blame, review individual performance, or devalue achievements. Use it purely as a learning tool.

Organizing a PIR is the responsibility of the project manager. Anyone involved in the project can be invited, but try to keep the number small and manageable. Go for representation breadth rather than depth. During the meeting, the project manager will moderate the discussion and invite the participants to express their views. A separate scribe will take notes. After the meeting, the project manager will review the notes and publish them as a document or to the wiki.

7.2 Benefit Realization

The anticipated benefits articulated in the original business case for the project need to be tracked. The timeframe for the benefits often varies. Some benefits may be realized immediately (e.g., reduced transaction completion time), others may have longer timeframes (e.g., a reduced operational headcount target may take time to achieve due to the training needs of users to become productive with the new system).

The realization of benefits may be tracked by the project sponsor using data supplied by others. Upfront definition of a set of clear metrics will enable this process to be done systematically.

Most organizations are not good at tracking benefit realization. This is unfortunate and represents a lost opportunity to objectively assess the net benefit of a project to the organization. Organizations that do this well, however, become far better at business case evaluation and project prioritization, improving their chances of optimizing the use of capital.

7.3 Production Support

In Agilex, production support is largely the same as traditional production support, except that fixes are developed and applied using agile principles.

Make sure that you involve your production support team well before a system goes live. They need time to become familiar with the design of the system and find their way through its code base. An economic way of doing this is to rotate your developers through the production support team, so that they can transfer their knowledge.

7.4 Incident Management

It's unavoidable that production systems will suffer incidents. Make sure that you have a formally defined process for dealing with incidents. Unless an incident is handled systematically, chaos and confusion is likely to rein, causing even rational people to behave irrationally under stress and panic.

As a minimum, do the following.

- Define and publish your incident management process. Have a summary formatted as a poster and use pictorial language (e.g., flow charts) to illustrate the steps to be followed. Display these posters where staff are able to see them on a daily basis.
- Provide staff training so that everyone is clear about their roles and responsibilities in dealing with incidents.
- Have a roster that identifies a trained individual as the incident manager for the month.
- When an incident happens, the incident manager must be first informed.
- The incident manager will then assemble the incident management team and go over the facts. Decisions are made within the defined process framework as to what to do next.
- Never allow staff to act unilaterally. Without a coordinated approach, people will guess and resort to trial and error, which is likely to make matters worse.

Your process must require each incident to be recorded in form of an incident report, which describes the symptoms, impact, likely causes, course of investigation, and an expected resolution timeframe. Incidents must be actively tracked until closed.

8 Process Optimization

Once you've successfully implemented your agile process, you can start thinking about how to get maximum value by optimizing it.

8.1 Optimization Measures

Agilex recommends six optimization measures that can collectively have a positive impact on the overall length and cost of your agile projects.

Eliminate wastage

Streamline your development process

Use productive technology

Devise a fast build process

Use a simple and foolproof deployment process

Co-locate your project team

Most processes involve some level of waste. By simply observing the process, you can identify where wastage is happening. For example, you might discover that, for historical reasons, your company mandates the collection of detailed metrics to generate elaborate reports that no-one actually uses for any meaningful purpose. A simple review of these metrics could eliminate the ones that serve no purpose.

The streamlining of the development process can deliver significant gains. For example, using the same developer to develop a vertical slice of the system (i.e., user interface, business logic, and data access layer) will avoid costly communication between developers dedicated to each layer.

Use the most productive technology available to you. Some technologies are far better suited to agile development than others. For example, modern interpreted scripting languages facilitate a more rapid code-build-test cycle than traditional compiled languages.

For agile to succeed, you need a rapid turnaround of builds; otherwise changes can't be applied quickly. For example, some legacy business applications have such convoluted and lengthy build processes that they're built overnight. Agilex projects must use build processes that can run within minutes. Carefully review your build process and eliminate bottlenecks.

Similarly, because Agilex is iterative and incremental, you want your deployment process to be as simple as possible. Traditional projects often need to resort to technical experts to manage their deployment processes, because they tend to be complex and error prone. You should aim to simplify your deployment process to the extent that it can be performed by a junior staff member.

Finally, co-location of project teams will pay significant dividends. A situation where your analysts are in one location, your developers in another, and your testers in yet another will create unnecessary inefficiencies. Although co-location costs money in terms of staff movement, this cost is recovered many times over through the efficiencies gained.

8.2 Metrics

Organizations that collect and publish metrics, do it for one of two reasons:

- To show that everything is under control and running well.
- To identify opportunities for improvement.

If you plan to collect metrics, make sure that you do it for the second reason. Intent drives behavior, so when the purpose of metrics is to show that things are under control, they tend to be designed to do just that!

Metrics can be powerful management reporting tools, conveying valuable information in highly distilled format that facilitate timely decision making. More often, organizations collect metrics for financial reporting (e.g., actual versus target sales) and less often for process improvement (e.g., the percentage of planned releases successfully completed). However, one must not overlook the fact that process improvement will ultimately lead to financial performance.

An effective way of designing your metrics is to use the agile process:

- Spend some time to explore indicators that could give you an objective measure of your processes. Ideally, you should brainstorm this as a group in a workshop.
- List everything you can think of and categorize and rank them.
- From this list, choose indicators that are better understood, more objective, and easier to measure.
- Put processes in place to collect these metrics and turn them into reports. Aim for summary reports of a few sheets. Always report your metrics as graphs.
- Review these metrics on a monthly basis to identify trends and improvement opportunities.
- Revise your metrics every six months or, at least, annually. You'll discover that they can always be improved.
- Make the metrics visible to your staff. This will have a direct positive impact on their behavior.

9 Further Reading

Agile Development

- *The Art of Agile Development* by Shane Warden, James Shore, Shore James, and Warden Shane, O'Reilly Media (2007). Provides practical advice on agile planning, development, delivery, and management based on the authors' many years of experience with Extreme Programming (XP). Covers technical as well as non-technical aspects of agile development.
- *Clean Code: A Handbook of Agile Software Craftsmanship* by Robert C. Martin, Prentice Hall (2008). A technical treatment, mainly for agile programmers. It promotes the notion of cleaning code 'on the fly' in order to facilitate easier iterations and future code maintenance.
- *Agile Journal* (www.agilejournal.com) is the largest online community for agile development, providing members with information and resources for developing software according to agile principles.

Project Management

- *Agile Project Management with Scrum* by Ken Schwaber, Microsoft Press (2004). Applies the principles of Scrum (one of the most popular agile programming methods) to software project management.
- *Agile Project Management: Creating Innovative Products* by Jim Highsmith, Addison-Wesley Professional (2009). Aimed at project leaders, managers, and executives, it combines project management, product management, and software development practices into an overall framework designed to support agility.

Business Case

- *The Business Case Checklist: Everything You Need to Review a Business Case, Avoid Failed Projects, and Turn Technology into ROI* by Business Case Pro LLC (2009). Offers tools and advice for turning technology into return on investment (ROI). It will help you make good investments and, more importantly, avoid bad investments.

JAD

- *Joint Application Development* by Jane Wood and Denise Silver, Wiley (1995). A step-by-step guide for JAD practitioners, providing examples of room layouts, forms for the scribe, and templates for JAD documentation.
- *Requirements by Collaboration: Workshops for Defining Needs* by Ellen Gottesdiener, Addison-Wesley Professional (2002). A pragmatic book, describing the nuts and bolts of planning and leading requirements workshops.

Documentation

- *Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects* by Jeff Staples, Technical Communication Journal, Volume 51, Issue 4, August 2005. A short guide for writing agile documents.

Design

- *Software Architecture: Foundations, Theory, and Practice* by R. N. Taylor, N. Medvidovic, and E. M. Dashofy, Wiley (Hardcover - Jan 9, 2009). A useful and broad text book on software architecture.
- *Software Design (2nd Edition)* by David Budgen, Addison Wesley (2003). Provides a balanced view of the various software design methodologies most widely used by practitioners, helping you choose the method most appropriate for your project.
- *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides, Addison-Wesley Professional (1994). A modern classic in the literature of object-oriented development, offering timeless and elegant patterns for tackling common software design problems.

Rapid Prototyping

- *Rapid Prototyping: Principles and Applications (2nd Edition)* by Chua Chee Kai, Leong Kah Fai, and Lim Chu-Sing, World Scientific Publishing (2003). Traces the development of rapid prototyping methods and their practical applications.

Testing

- *Agile Testing: A Practical Guide for Testers and Agile Teams* by Lisa Crispin and Janet Gregory, Addison-Wesley Professional (2009). An excellent treatment of testing in an agile context, covering the full spectrum of testing phases.
- *Software Testing: Fundamental Principles and Essential Knowledge* by James D. McCaffrey, BookSurge Publishing (2009). A brief and authoritative summary of important software testing principles.

Metrics

- *Practical Software Metrics for Project Management and Process Improvement* by Robert B. Grady, Prentice Hall (1992). Provides over 70 charts and graphs from real projects, illustrating metrics aimed at process improvement.
- *Metrics and Models in Software Quality Engineering (2nd Edition)* by Stephen H. Kan, Addison-Wesley Professional (2002). A balanced treatment of product metrics (e.g., reliability) and broader metrics (e.g., customer satisfaction).