

Harvard Extension School - Spring 2007

Laurent Wiart

Iterative Methodologies to Manage Web-Based Software:

Focus on OpenUp/Basic

Jeffrey E. Francis

MGMT E-120

Project Management of Information Technology

Table of Contents

Introduction.....	4
Waterfall versus Iterative Methods: Quick overview.....	4
The OpenUP/Basic methodology.....	5
OpenUP/Basic principles.....	5
The OpenUP/Basic life cycle.....	6
Inception.....	6
Elaboration.....	6
Construction.....	6
Transition.....	6
OpenUP/Basic organization.....	6
Roles.....	7
Disciplines.....	8
Tasks.....	8
Artifacts.....	9
Process.....	9
Questions and Answers.....	10
How do the Agile methodologies improve the communication discipline?.....	10
How do the Agile methodologies improve quality?.....	10
What are the main values of OpenUP/Basic?.....	10
What are the common traps of OpenUP/Basic?.....	11
Why Agile method is better than upfront planning?.....	11
Why not plan the design upfront?.....	11
What is the impact of the OpenUP/Basic methodology when the Project Manager is also the developer?.....	12
What happens when the project is managed distinct from the development?.....	12
How is discipline of project management enforced by agile methods?.....	12
What about the scaling up of adaptive teams?.....	13
Conclusion.....	13
Appendix A - Overview of RUP and of the most common Agile methodologies.....	19
RUP: Rationale Unified Process.....	19
Agile Methodologies.....	19
Scrum.....	20
DSDM: Dynamic Systems Development Method.....	20
Crystal Methods.....	20
FDD: Feature-Driven Development.....	21
LD: Lean Development.....	21
XP: Extreme Programming.....	21
ASD: Adaptative Software Development.....	21
Appendix B - The OpenUP/Basic methodology published by the EPF	22
Appendix C - OpenUP/Basic Core Concept: Balance competing priorities to maximize stakeholder value.....	23
Appendix D - OpenUP/Basic Core Concept: Collaborate to align interests and share	

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

- understanding 26
- Appendix E - OpenUP/Basic Core Concept: Focus on articulating the architecture 29
- Appendix F - OpenUP/Basic Core Concept: Evolve to continuously obtain feedback and improve..... 31
 - Introduction 31
 - Practices 31
 - Develop your project in iterations 31
 - Focus iterations on meeting the next management milestone 32
 - Manage risks 32
 - Embrace and manage change 32
 - Measure progress objectively 32
 - Continuously re-evaluate what you do 33

Introduction

Nowadays, many software are web-based products. They tend to be more and more equivalent to their "desktop-installed" counterparts.

But they also eliminate a great deal of constraints from the delivery organization:

- The software/hardware environment is controlled and managed by the delivery organization,
- The testing phase is more predictable since the deployment environment is controlled by the delivery organization.
- New ideas, improvements and bug-fixes can be implemented and evaluated quickly.
- The integration and packaging phases are much easier.
- The usage patterns can be easily analyzed to improve the software.

This new paradigm has led to new opportunities but also to new constraints (network constraints, maximization of responsiveness, security and reliability of hosted user data) and new project management challenges. Indeed, heavy-weight development processes such as Waterfall are not fitted to the reactivity required by such products which rather require the adaptability of iterative software development processes.

OpenUP/Basic is a lightweight iterative software development process targeted to small and collocated teams. It is **minimal, complete, and extensible**.

The aim of this project is to assess the relevance of OpenUP/Basic for the development of web-based products.

This methodology will be described and a few questions will be answered: the risks of such methodology, impact of mixing roles, the potential communication issues, how discipline is enforced.

To conclude, we will enlarge the analysis by discussing the Agile philosophy and its tremendously positive impact on today's software development and project management disciplines.

Waterfall versus Iterative Methods: Quick overview

Perhaps the most striking difference between waterfall and the agile methods is the monolithic aspect of waterfall, as opposed to iterative/adaptive aspect of agile methods, as shown in Illustration 1 and Illustration 2 (Kruchten, 2000). The main flaw of the waterfall process is that the project manager is supposed to define and design early in the project, when little is known about the product and defects can be discovered late (potentially too late...) in the process. As a result, instead of diminishing with time, the risks increase and so do the costs to correct defects. This type of flaw is, by nature, addressed by iterative methods where each iteration is a full-featured

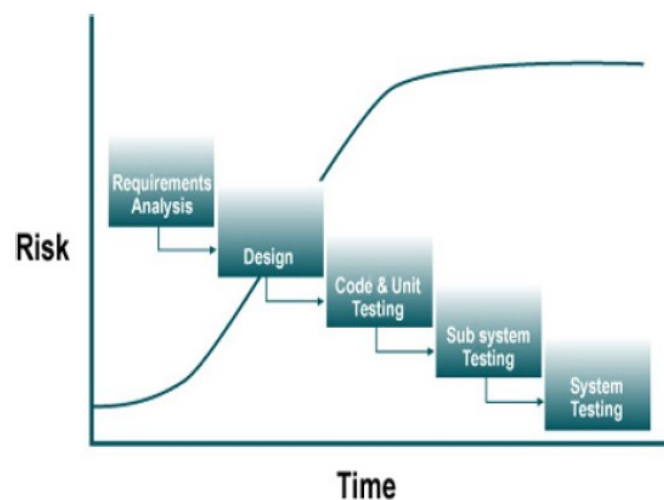
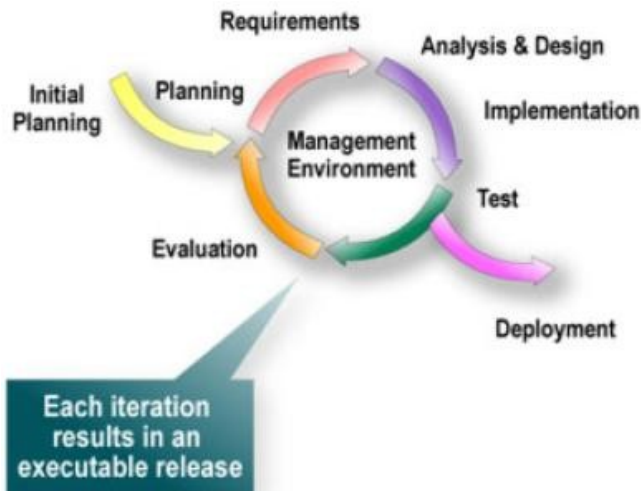


Illustration 1: The Waterfall Development Process

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

project management process with its own design-implement-test-close phases.



The iterative methodologies are particularly well suited to web-based products which require rapid adaptation to market needs, experimentation of new technologies, preservation of global coherency and reliability. Iterative methodologies accommodate change, mitigate risks early in the development process. To put it succinctly: **in a world of change and chaos like software development, the waterfall approach is too strict and rigid to be successful.**

Illustration 2: Iterative Approach to Development

The OpenUP/Basic methodology

OpenUP/Basic, previously known as Basic Unified Process (BUP), is the “open-source” version of RUP. It was donated by IBM to the open source community in 2005. Since then, it has been promoted by the Eclipse Project Framework (EPF, 2007). This software development process is targeted to small development teams (3 to 6 developers) for short project (3 to 6 months). It can be seen as the counterpart of RUP in terms of process: RUP contains “everything possible” (work products, roles, tasks...), whereas OpenUP/Basic is minimal yet complete, and can be extended (it can be built upon and tailored to fulfill any process requirement).

OpenUP/Basic principles

The 4 core principles of OpenUP/Basic are based on those of RUP listed in Illustration 3 (Maclsaac, 2006):

- **Balance** competing priorities to maximize stakeholder value - Team members and stakeholders must collaborate to develop a solution that maximizes stakeholders benefits and is compliant with the constraints of the product.
- **Collaborate** to align interest and share understanding - a healthy team environment must be fostered to assure success.
- **Evolve** to continuously obtain feedback and improve - divide the project into short, timeboxed iterations to demonstrate incremental value and get early and continuous feedback. As a



Illustration 3: Principles of the RUP methodology (Kroll, 2006)

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

result, the risks can be identified and mitigated earlier with changes made when needed.

- **Focus** on articulating the architecture - give an architectural foundation to the product, focused on reusability and quality.

See appendices C to F for details.

The OpenUP/Basic life cycle

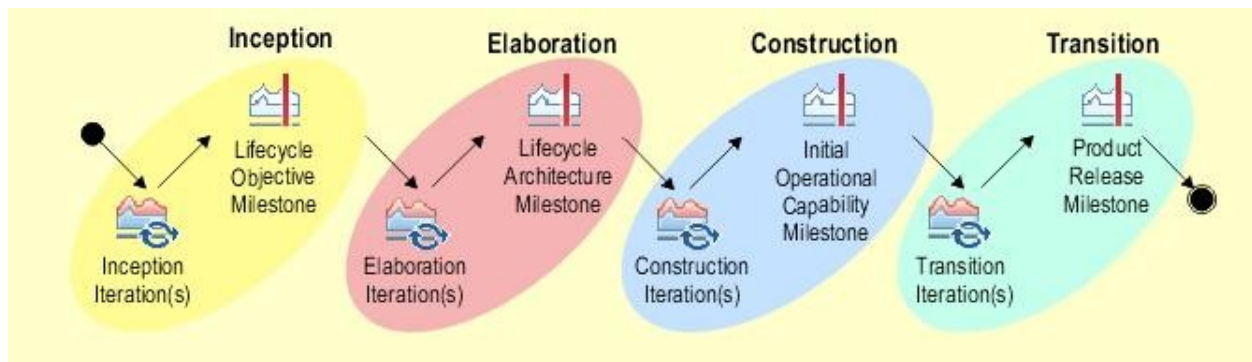


Illustration 4: OpenUP/Basic lifecycle (EPF, 2006)

As RUP, the product life cycle comprises an **inception**, an **elaboration**, a **construction** and a **transition** phase (Illustration 4).

Inception

Understand the project scope, the objectives and the stakeholder priorities.

Elaboration

Establish the baseline of the system architecture, focusing on the most architecturally delicate parts to mitigate the greatest risks first.

Construction

Design, implement and test functionalities have to be made in this phase. This phase relies on the underlying architecture created during elaboration phase.

Transition

Transfer the product to customers/users (training, feedback, last changes and documentation).

OpenUP/Basic organization

OpenUP content is organized into sub-processes around a core of communication and collaboration as seen in Illustration 5. Each content area of OpenUP/Basic supports a statement in the Agile Manifesto (Beck, 2001):

Content Area	Agile Manifesto	OpenUP/Basic principles
Management	Responding to change over following a plan	Demonstrate value iteratively and Adapt the process (steering, monitoring)
Intent	Customer collaboration over contract negotiation	Balance stakeholders priorities - to maximize benefits (design)
Solution	Working software over comprehensive documentation	Elevate the level of abstraction and Focus continuously on quality (tests, refactoring)
Communication & Collaboration	Individuals and interactions over process and tools	Collaborate across teams - to improve understanding and interest into the project

In addition to conforming to agile principles, OpenUP/Basic includes a range of agile concepts including **test-first design**, **continuous integration**, **time-boxed iterations**, and **refactoring**.

Roles

The essential skills needed by small and co-located teams are represented by OpenUP/Basic roles illustrated in Illustration 5, where each role is focused on particular content areas:

Stakeholder represents interest groups whose needs must be satisfied by the project. It is a role that may be played by anyone who is (or potentially will be) materially affected by the outcome of the project.

Project Manager leads project planning of the project, coordinates interactions with the stakeholders, and keeps the project team focused on meeting the project objectives.

Analyst represents customer and end-user concerns by gathering input from stakeholders to understand the problem to be solved and by capturing and setting priorities for requirements.

Architect is responsible for designing the software architecture, which includes making the key technical decisions that constrain the overall design and implementation of the project.

Developer is responsible for developing a part of the system, including designing it to fit into the architecture, and then implementing, unit-testing, and integrating the components that are part of the solution.

Tester is responsible for the core activities of the test effort. Those activities include identifying, defining, implementing, and conducting the necessary tests, as well as logging the outcomes of the testing and analyzing the results.

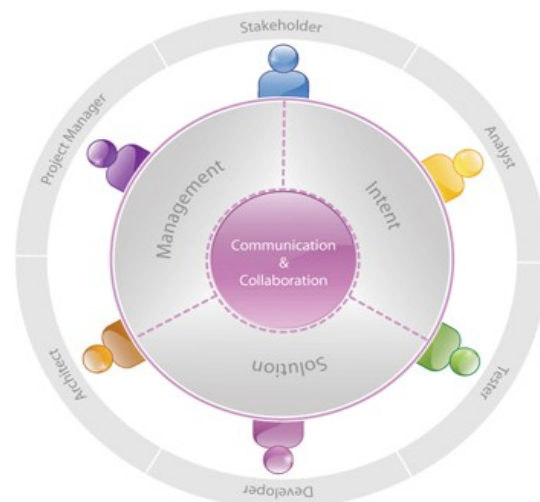


Illustration 5: The four major areas of content of OpenUP/Basic (EPF, 2006)

Disciplines

The OpenUP/Basic method is focused on the following disciplines: Requirements, Analysis and Design, Implementation, Test, Project Management, and Configuration & Change Management.

Other disciplines and areas of concern were omitted, such as Business Modeling, Environment, advanced Requirements Management and Configuration Management tools setup. These concerns are considered unnecessary for a small project or are handled by other areas of the organization, outside the project team.

The method content supporting the disciplines is spread across the content areas. Requirements and Configuration & Change Management content is in the Intent area, Analysis and Design and also Implementation content is in the Solution area, Project Management is in the Management area. The Test discipline is unique in that it has coverage in both the Intent area (with the specification of test cases) and the Solution area (with the creation of test scripts).

The roles which work together and the artifacts shared throughout the process all reside in the core Communication & Collaboration area.

Tasks

A task is unit of work a role may be asked to perform. In OpenUP/Basic, there are **18 tasks** that the roles perform with primary responsibility or as an additional performer (supporting and providing information used in the task execution). The collaborative nature of OpenUP leads to many interactions between the roles in order to perform a task.

The Illustration 6 and Illustration 7 show the tasks and work products for the Project Manager role and Developer role.

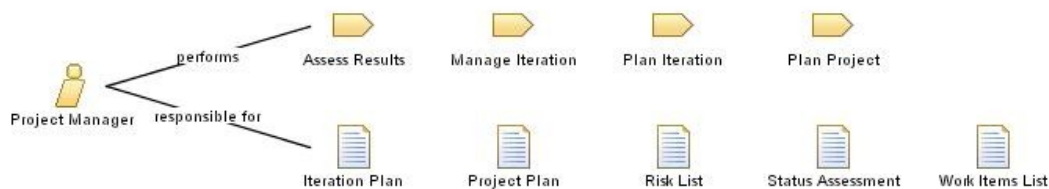


Illustration 6: Project Manager's tasks and work products (responsibilities)



Illustration 7: Developer's tasks and work products (responsibilities)

Artifacts

Each role is responsible for creating and updating the project management artifacts. They are subject to version control throughout the project life cycle.

The **20 artifacts** in OpenUP/Basic are considered the essential artifacts a project should use to capture product- and project-related information. There is no obligation in capturing information in formal artifacts. The information could be informally captured in white board (e.g., for design and architecture), meeting notes (e.g., for status assessments), etc. Templates though provide an out-of-the box, standard way to capture information. Projects can use the OpenUP/Basic artifacts or replace them with their own.

Process

Method content provides step-by-step explanations to achieve each development goal.

Processes take these method elements and relate them to semi-ordered sequences that are customized to specific types of projects. These patterns are made from organizing tasks into activities (from the method content) and grouping them into a sequence called a “capability pattern”.

All these elements rationalize each activity, the order of the tasks and the expected results.

The following table describes the relationship between each phase's objectives and the iteration's pattern.

Iteration template pattern	Phase objectives
Inception Phase Iteration Initiate Project Manage Iteration Manage Requirements Determine Architectural Feasibility	Understand what to build. Identify key system functionality. Determine at least one possible solution. Understand the cost, schedule and risks associated with the project.
Elaboration Phase Iteration Manage Iteration Manage Requirements Define the Architecture Develop Solution (for requirement) (within context) Validate Build Ongoing Tasks	Get a more detailed understanding of the requirements. Design, implement, validate, and baseline an Architecture. Mitigate essential risks, and produce accurate schedule and cost estimates.
Construction Phase Iteration Manage Iteration Manage Requirements Develop Solution (for requirement) (within context) Validate Build Ongoing Tasks	Iteratively develop a complete product that is ready to transition to its user community. Minimize development costs and achieve some degree of parallelism.
Transition Phase Iteration Manage Iteration Develop Solution (for requirement) (within context) Validated Build Ongoing Tasks	Beta test to validate that user expectations are met. Achieve stakeholder concurrence that deployment is complete.

Questions and Answers...

How do the Agile methodologies improve the communication discipline?

Scrum and XP promote the presence of the development team together in the same room. This practice has a direct benefit on the communication and interaction. This approach, coupled with XP's pair programming (2 developers programming together on the same machine) promote the common ownership of the code and reduce the defects by 65%. This discipline of communication is also enforced by the presence of white boards on the walls, and display of as many project information as possible on the walls (tasks, risk list, UML diagrams, vision, ...). This kind of practice can be viewed as pointless at first, but strongly promotes communication and collaboration. Simplicity of the tools (whiteboard, pencil, paper, camera to capture the mock ups, the designs) is also important (no powerpoint, nor fancy tools), because it simplifies communication and, as a result, team members are more willing to adopt these disciplines of communication and collaboration. (note from the author: **the easier it is to adopt, the more it will!**).

One of the simplest and most powerful practice is the **Scrum meeting**: a daily stand up meetings of no more than 15 minutes where each team member explain what he did since the last scrum meeting, what he is going to do until the next meeting and what are the road blocks that could prevent or delay attainment of his goals.

These 3 questions only (the Scrum Master must enforce these rules and prevent team members from diverting), in such short amount of time allow the entire team to be aware of the tasks of each others and to weight in (after the Scrum meeting !) when they think they could help in any way.

How do the Agile methodologies improve quality?

XP promotes test-driven development (unit tests): the practice of developing the tests first and then writing the code that will allow the tests to pass. This approach, coupled with use of coding standards (all developers must use the same coding style) lead to several advantages: the tests, frequently launched on a dedicated machine, will automatically detect when a functionality is broken.

As a result, any team member can refactorize at any time the code of a colleague if necessary, knowing that if he brakes something, the unit tests will allow (hopefully) detection of the defect. These disciplines are more likely to be followed if everybody use the same coding rules and won't fear to break a functionality thanks to the unit-tests.

What are the main values of OpenUP/Basic?

The values of OpenUP/Basic are numerous, here is a list of the most important:

- Attack the risk early and continuously, or they will attack you.
- Deliver value to customers, early and often.
- Stay focused on developing executable software in early iterations, not documentation or specifications.
- Accommodate change early. Encourage and manage it. To put it simply: the only thing that doesn't change is... change itself !

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

- Prefer component oriented architectures and reusability of existing components.
- Work together as a team.
- Quality is a way of life, not an afterthought.

Adapted and completed from Larman, 2004 (p.87).

What are the common traps of OpenUP/Basic?

The mistakes to avoid for agile methodologies and OpenUP/Basic in particular are:

- **Superimposing waterfall approach to the OpenUP phases:** inception assimilated to requirements, elaboration assimilated to requirements and design, construction assimilated to implementation and transition assimilated to testing.
- **Iterations are too long:** must be 2 to 6 weeks, to constantly deliver value-added to the product and stakeholders and to have a “production-ready” product at all time.
- **Iterations aren't timeboxed:** an iteration is a “contract” between the stakeholders and the development team on a list of features. The stakeholders agree to refrain from demanding a new feature in an open iteration and to wait for the next one (the fact that iterations are short helps to wait for the next one). In case of slippage, the duration of the iteration must not be prolonged, but features can be moved to the next iteration. This discipline allows to have a constant duration for the iterations and allow to have fixed landmarks where most of the elements are moving parts. To put it simply: **Once an iteration is started, what and when can't be changed.**
- **Iterations don't end in an integrated and tested baseline:** one of the main goal of timeboxed iterations is to provide a subset of the final product completely functional and tested.
- **Predictive planning:** one must not go too far in the planning because it will change frequently and unexpectedly.
- **Need many modeling and modeling tools:** OpenUP/Basic is a lightweight process. Its aim is to produce the necessary use cases and work products, not more.
- **Not conforming to the vocabulary:** one of the goal of OpenUP/Basic and the process methodologies in general is to establish a common vocabulary to facilitate communication across the team and the organization.

Adapted and completed from Larman, 2004, (pp.194-197).

Why Agile method is better than upfront planning?

Ken Delcol says: “People believe when they plan that they introduce certainty, which is far from the truth. What they introduce is something to gauge their performance by. Then, when the gauge does not reflect reality, they fail to replan.”. Agile methods force us to confront the reality of today's business: a highly volatile environment of product development.

Why not plan the design upfront?

The same reasoning as for the preceding question: the product is bound to change, and its evolution

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

can't be planned. As a result, it is almost impossible to have the right design upfront. Jim Highsmith puts it this way: “No matter how good it is the first time, it's going to change, so keep the cost of change low.” (Highsmith, 2004, p.181).

The XP disciplines reflect this reality: keep things simple (code for today and not for tomorrow) and refactorize and redesign when needed, to keep the product reactive to market change. This discipline, along with courage (don't hesitate to throw out code) and feedback (unit tests and acceptance tests) generate the best value out of the code at any time and facilitate rapid adaptation to change.

What is the impact of the OpenUP/Basic methodology when the Project Manager is also the developer?

The Project Manager and Developer roles have different tasks and work products in OpenUP/Basic methodology, as detailed in Illustration 6 (p.8) and Illustration 7 (p.8) but are not incompatible. Both roles can be played by the same person, depending on the constraints of the project: if the project management part is “light”, the time affected to this role will be less important and the developer role will be more preminent. If the project is more demanding in terms of project management artifacts (if the size of the project is bigger or its criticality more important). Note that Project Manager can also play the role of Architect.

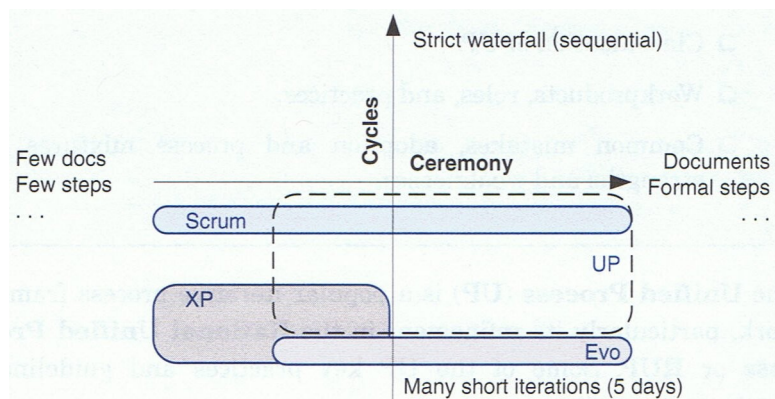


Illustration 8: OpenUp on the cycles and ceremony scales (Larman, 2004)

What happens when the project is managed distinct from the development?

In the Agile methodologies, the Project Manager is part of the team, one of the most preminent roles, to enforce the disciplines required by the method, to manage the team and the relations with stakeholders (see Illustration 6, p.8). The project manager must also be technically competent, able to understand the technical challenges of the product and at the same time able to communicate to the stakeholders. This role is best suited to a generalist rather than a specialist in only one of the technical area of the project.

How is discipline of project management enforced by agile methods?

Agility is a mindset, a way of thinking, not a set of practices or processes (Highsmith, 2004, p.237). People with the right skillset and attitude need to be steered rather than controlled. The agile methodologies tend to attain this goal. One of the most motivating approaches is to let team members understand what is expected from them, not in term of steps (micromanagement) but in terms of outcome. Marcus Buckingham and Curt Coffman recommend “Define the right outcomes and then let each person find his own route towards those outcomes”.

Jim Collins, in his book “Good to Great”, mentions “if we get the right people on the bus, the right people in the right seats, and the wrong people off the bus, then we’ll figure out how to take it someplace great” (p.41). As a result, the bureaucratic rules are mostly done to manage the wrong people on the bus, those whose lack of discipline prevents them from performing and from being accountable for their performance.

Adapted from Highsmith, (p.182: Coaching and Team Development).

What about the scaling up of adaptive teams?

If the size of the team increases, the values must remain, no matter the size. Agile teams balance flexibility and structure. The right small to medium sized teams (25 or less) utilizing an agile framework and practices can often accomplish more than a significantly larger team.

An organizational solution to the scaling of such teams is the Hub Organization Structure (Highsmith, 2004): it isn’t a hierarchical but a network structure where the power and decision making are distributed to the team. The project manager retains final authority, but his primary goal is steering, not controlling. The Hub organization is described in Illustration 9.

In such an organization, each team must retain accountability and engage collaboratively with the other teams.

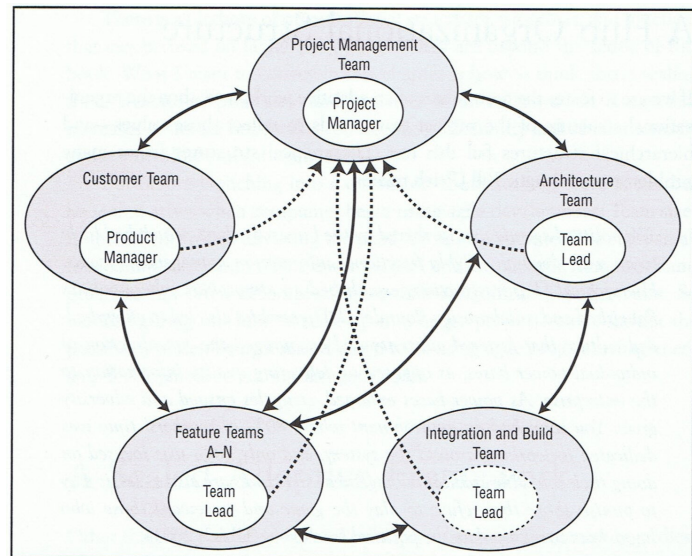


Illustration 9: Hub Organization Structure (Highsmith, 2004, p.240)

Conclusion

The beginning of this work was focused on studying OpenUP/Basic as a methodology applied to web-based software development. But through the study of some additional agile and iterative methodologies (Scrum, XP...), the author realized that Agile and Iterative methodologies are not just a set of practices. They are new ways of approaching software development and even project management (Jim Highsmith, in his book “Agile Project Management”, applies Agile philosophy to Project Management).

The Agile disciplines, whether taken from Scrum, XP, Crystal, OpenUP/Basic, ... are all parts of a puzzle. Taken separately, they don’t make a lot of sense. The real value comes from assembling some of the practices to create **Synergy**.

Consider the following practices taken from XP: pair programming + coding standards + unit tests + simplicity + refactorization of the code when needed. Used separately, you don’t get much out of them. But applied together, they lead to:

- the common ownership of the code by the entire development team
- lack of fear to refactorize the code to reach the optimal solution for the problem of today,

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

- improved quality of the code base,
- real capacity to face change in the evolution of stakeholder needs,
- optimal overall quality of the product.

This is one example among many and this type of “assembly” can be done with many practices in many different ways, as Craig Larman did in his book “Agile and Iterative Development: A Manager's Guide” (2004) where he assembles practices from XP, Scrum, UP, Evo together to maximize their core value.

The real synergy of these practices comes when team members and stakeholders get accustomed to the agile and iterative practices. But it is also important to mention that these practices can be adopted at different levels, depending on the needs of the product, the environment and the organization (Kroll, 2006).

As a result, the three remarks that come to my mind today are:

- **there is no one-size-fits-all in management of software project,**
- **many experienced thought leaders came up with solutions which are centered on human qualities more than on processes (Agile Manifesto, 2001),**
- **if we apply these ideas in an mindful manner to our project's environment and constraints, the benefits can be tremendous.**

Alphabetical Index

Adaptative Software Development.....	21
Agile Manifesto.....	7
Agile Methodologies.....	19
Analyst.....	7
Architect.....	7
artifacts.....	8p.
ASD.....	21
Basic Unified Process.....	5
BUP.....	5
construction.....	6, 19
Crystal.....	20
developer.....	7p., 12
Developer role.....	8
disciplines.....	8
DSDM.....	20
Dynamic Systems Development Method.....	20
Eclipse Process Framework Project.....	17
Elaboration.....	6, 19
epf.....	5, 17, 22
Extreme Programming.....	21
FDD.....	21
Feature-Driven Development.....	21
inception.....	6, 19
incremental process.....	20
Intent.....	8
iterative process.....	20
LD.....	21
Lean Development.....	21
life cycle.....	6
Management.....	8
Measurable Organizational Value.....	19
MOV.....	19
OpenUP/Basic.....	4p.
OpenUP/Basic methodology.....	5, 12
OpenUP/Basic organization.....	6
OpenUP/Basic principles.....	5, 7
PMBOK.....	19
process.....	4pp., 17, 19pp.
Project Manager.....	4, 7, 12
Project Manager role.....	8
RAD.....	20

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

Rapid Application Development.....	20
Rationale Unified Process.....	19
roles.....	5, 7, 8
RUP.....	19
Scrum.....	20
Scrum meeting.....	10
Solution.....	8
Stakeholder.....	7
tasks.....	8p.
Tester.....	7
Transition.....	6, 18p.
Waterfall.....	4
XP.....	21
.....	9

References

Beck, K. Beedle, M. Van Bennekum, A. Cockburn, A. Cunningham, W. Fowler, M. Grenning, J. Highsmith, J. Hunt, A. Jeffries, R. Kern, J. Marick, B. Martin, R. Mellor, S. Schwaber, K. Sutherland, J. Thomas, D. (2001). *Agile Manifesto*. Retrieved March 28th, 2007 from <http://www.agilemanifesto.org>

Balduino, R. & Lyons, B. (2006). *OpenUP/Basic - A Process for Small and Agile Projects*. Retrieved March 11th, 2007 from http://www.eclipse.org/epf/general/OpenUP_Basic.pdf

Collins, J. (2001). *Good to Great*. Harper Business. ISBN: 0066620996

EPF (Eclipse Process Framework Project). EPF Composer. Free/Open-source software. Retrieved March 18th, 2007 from <http://www.gtlib.gatech.edu/pub/eclipse/technology/epf/composer/release/epf-composer-1.0.2-win32.zip>

EPF (Eclipse Process Framework Project). *OpenUP/Basic published Web site*. Retrieved March 18th, 2007 from http://www.gtlib.gatech.edu/pub/eclipse/technology/epf/OpenUP/published/OpenUP_Basic_published-0.9-20061002.zip

Highsmith J. (2002). *Agile Software Development Ecosystems*. Addison-Wesley Professional. ISBN: 0201760436

Cockburn, A. (2002). *Agile Software Development*. Addison-Wesley Professional (first edition published December 15, 2001). ISBN: 0201699699

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison-Wesley Professional (first edition published October 19, 2004). ISBN: 0201699478

Cohn, M. (September 11-14, 2006). *Selecting a Development Process: Choosing Among the Leading Alternatives*. Retrieved March 18, 2007 from http://www.mountangoatsoftware.com/system/presentation/file/41/SDBP2006_SelectingADevelopmentProcess.pdf

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

Gilb, T. (1988). *Principles of Software Engineering Management*. Addison Wesley Professional. ISBN: 0201192462

Highsmith J. (2006). *Agile Project Management*. Addison-Wesley. (first edition published 2004). ISBN: 0321219775

Kroll, P. Maclsaac, B. (2006). *Agility And Discipline Made Easy: Practices from OpenUP and RUP*. Addison-Wesley. ISBN: 0321321308

Kruchten, P. (dec. 2000). *From Waterfall to Iterative Development - A Challenging Transition for Project managers*. Article.

Retrieved march 11, 2007 from

<http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/dec00/FromWaterfalltoIterativeDevelopmentDec00.pdf>

Larman, C. (2004). *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional (first edition published 2003). ISBN: 0131111558.

Appendix A - Overview of RUP and of the most common Agile methodologies

RUP: Rationale Unified Process

This iterative methodology is a refinement of UP (Unified Process). It was created in the mid-90's by a team from Rationale, led by Philippe Kruchten, Grady Booch, Mike Devlin, Rich Reitman and Walker Joice and was largely inspired by Ivar Jacobson's and Barry Boehm's work.

Its key practices are:

- Timeboxed iterations,
- Development of the most risky and most valuable parts first in the early iterations,
- Accommodation to change

This methodology is composed of 4 phases, which can be done in several iterations of 2 to 6 weeks.:

- Inception: the business case is established, the risks and success factors assessed. The MOV (Measurable Organizational Value) is also established,
- Elaboration: the most risky and architecturally significant blocks are programmed and tested. At the end of this phase, a plan and estimates can be pretty reliably created,
- Construction: programming and testing of the remaining parts, documentation, performance tweaking,
- Transition: deployment of the system (release candidate), review and feedback, and final deployment.

Typically, inception is the shortest phase and construction is the longest.

This methodology is broad and generic. It endorses many practices, identifies many work products but does not enforce the usage of any of them. It is a project management framework where the process can be precisely adapted to fit to the constraints of any type of project (like the PMBOK for Project Management), with a focus on best practices. One of the most common misconceptions about RUP is that it is too “heavyweight” (many work products, processes, ...). It can be adapted to each case, resulting in an heavyweight OR lightweight process, depending on the criticality of the project. The second most common misconception is to apply it in a waterfall state of mind with a loss of focus on the iterative concepts.

The most striking achievement of UP/RUP was the creation of the Canadian Automated Air Traffic Control System (Ada, C++), led by P. Kruchten, which involved 400 people for 10 years, after a failed waterfall project of 11 years and 2.6 billion USD...

Agile Methodologies

Agile methodologies is a subset of the iterative methods whose philosophy is centered on the people rather than on the process. The core values are summed up in the famous “Agile Manifesto” (Beck, 2001). Each iteration is a full featured process with requirement-analysis, design, implementation

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

and tests. At the end of an iteration, a stable and tested product is delivered, with a subset of the required features. The final iteration delivers the final product, feature-complete.

In fact, an **iterative process** can also be called an **incremental process** since each iteration brings new features, as part of the global improvement of the final product. One of the main constraints is that at the end of each iteration, the deliverable must be of “production-quality”. As Craig Larman puts it: “the software resulting from each iteration is not a prototype or proof of concept, but a subset of the final system.” (Larman, C. 2004, p.11).

A higher quality standard can be achieved by these methods (Kruchten, 2000) since:

- you can **discover and address risks** during the integration phase. As Tom Gilb (1988) puts it in an elegant manner: “Attack the risks actively or they will attack you”,
- you can **accommodate to change** throughout iterations: changes in requirements (the famous “feature creep” tendency), tactical changes (strategical adaptation to competitors and market needs), and technological changes (new technologies),
- you **learn as you go**, resulting in a constant adaptation of the developers to the best practices,
- you increase the **opportunity for reuse** since many design phases are conducted and common problems can be solved with common approaches,
- you achieve a **better overall quality** since the system is continuously tested and presented to users (during each iteration).

Scrum

This method was originally developed by Ken Schwaber and Jeff Sutherland. Each iteration is 30 days long and is called a sprint. The main process is a daily 15 minutes meeting during which team members explain what they did since the last meeting, what they are going to do until the next one and what are the potential problems they think they are going to face.

DSDM: Dynamic Systems Development Method

This is an extension of the RAD (Rapid Application Development) practices. This method has 9 principles. The most important of which are: active user involvement, frequent delivery, team decision making, integrated testing throughout the project life cycle and reversible changes in development.

Crystal Methods

These methods, developed by Alistair Cockburn primarily focus on people and communication aspects: good citizenship, cooperation and collaboration. Crystal is a complete family of methodologies. The most well known version is Crystal Clear (Highsmith, 2002. chapter 19). Each version contains an increasing “weight” in terms of process (project deliverables, documentation, processes) depending on the criticality of the project. Alistair Cockburn has created a very elegant scale to match the method to the criticality of the project according to the size of the team and to the core stake: life, essential money, discretionary money, comfort (Cockburn, 2002).

FDD: Feature-Driven Development

This method is a five step process focused on building a “shape” object model, building a feature list, then planning by feature. Then the iterative process is design-by-feature/build-by-feature steps. This method, created by Jeff De Luca and Peter Coad has been successfully applied to projects of more than 50 people (Highsmith, 2002. chapter 20).

LD: Lean Development

Bob Charette's Lean Development is derived from the lean production principles that developed in the Japanese car industry in the 1980's. Charette extends the classical view of project management to a view of “risk entrepreneurship” to produce new opportunities (Highsmith, 2002. Chapters 16, 21).

XP: Extreme Programming

This method, flagship of the Agile methodologies, has been developed by Kent Beck, Ward Cunningham, and Ron Jeffries. The main values are: **communication** (between the developers and with stakeholders), **simplicity** (of the code and the solution: “code for today and not tomorrow”), **feedback** (from the system with the unit tests, from the customers with the acceptance tests and from the team to the stakeholders), **courage** (to start simple and then refactorize the code, or even throw out the deprecated code), and **respect** (other team developers by not committing changes that would break the code baseline, respect of the work by always searching for the best solution, and simply respect of the other team members). The technical excellence of this methodology is based upon “unit-test first” developments, simplicity and constant refactoring (Highsmith, 2002. Chapters 4,10,12,22).

ASD: Adaptive Software Development

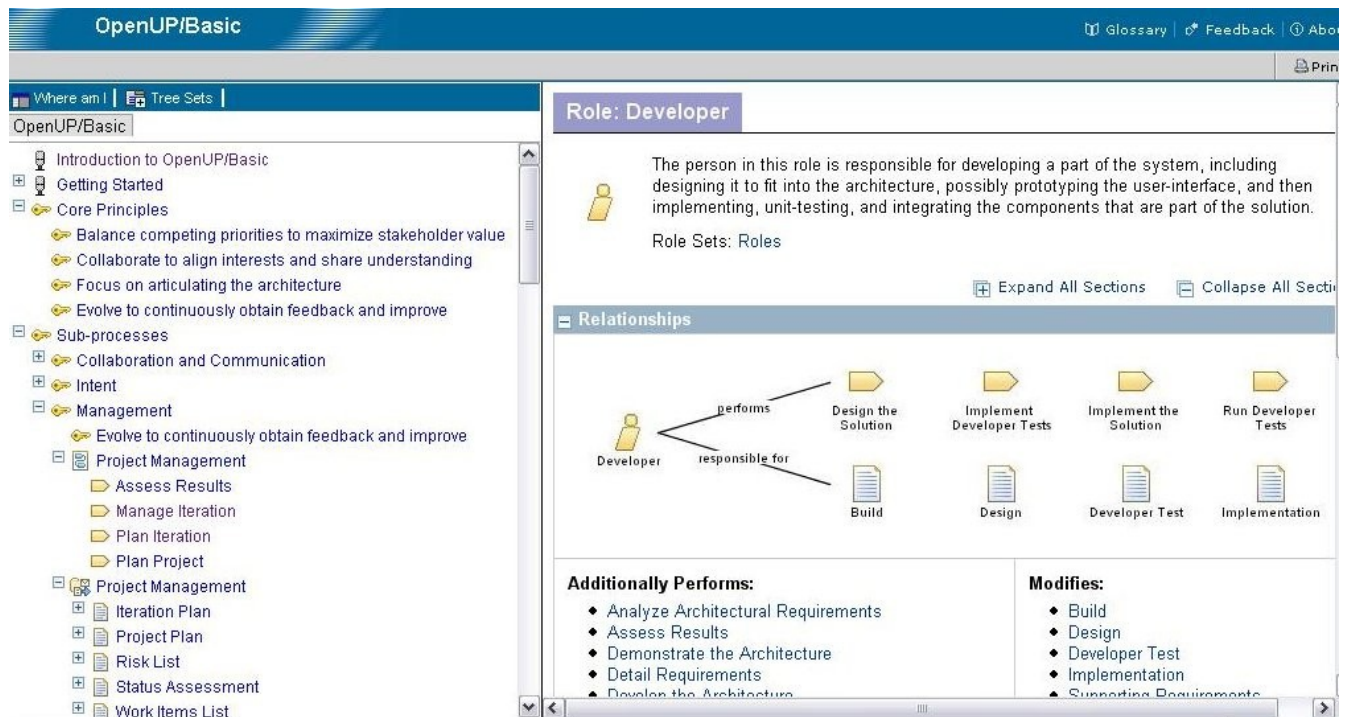
Developed by Alistair Cockburn (Highsmith, 2002), focuses on change. The practices are those of iterative development, feature-based planning, customer-focus group review and an Agile management philosophy called Leadership-Collaboration management (Chapter 23).

Appendix B - The OpenUP/Basic methodology published by the EPF

The Eclipse Process Framework has created a comprehensive web content that entirely describe the OpenUP/Basic process in terms of tasks/roles/work product/methodology.

As of today (march 28th, 2007), the OpenUP/Basic published version is 0.9 and can be retrieved at http://www.eclipse.org/downloads/download.php?file=/technology/epf/OpenUP/published/OpenUP_Basic_published-0.9-20061002.zip

This published website looks like:



It details the core principles, the roles, the tasks, work-products and even templates (documents, spreadsheets) for the different artifacts such as the work item list.

Appendix C - OpenUP/Basic Core Concept: Balance competing priorities to maximize stakeholder value

The following is part of the OpenUP/Basic version 0.9 published version 20061002, under Eclipse Public License v1.0

Key principle

Develop a solution that maximizes stakeholder benefits and complies with constraints placed on the project.

Introduction

Systems are rarely all things to all people. Often, attempts to make them so are wasteful and result in bloated systems.

Therefore, project participants and stakeholders must collaborate to develop a solution that maximizes stakeholder benefits and complies with constraints placed on the project. Achieving balance is a dynamic process because as both the stakeholders and project participants learn more about the system, their priorities and constraints change.

To be successful, stakeholders and the project participants must converge on a clear understanding and agreement of these three factors:

- Problem to be solved
- Constraints placed on the development team (cost, schedule, resources, regulations)
- Constraints placed on the solution

Collectively, these three items represent the requirements for the development of the system. The challenge for all project participants is creating a solution that maximizes value delivered to the stakeholders, subject to the constraints. Balance is about making the critical cost-benefit trade-offs between desired features and the subsequent design decisions that define the architecture of the system.

Discovering the balance point is challenging, elusive, and ongoing, because the balance point is dynamic. As the system evolves, stakeholder needs change, new opportunities appear, risks are resolved, new risks appear, and the development team discovers new realities about the system. Change happens throughout the development cycle. Therefore, stakeholders and developers must be prepared to re-evaluate commitments, reset expectations, and adjust plans accordingly as the system evolves.

Practices

The next sections describe the practices associated with this principle.

Know your audience

You cannot know how to make effective trade-offs if you do not know who the stakeholders are and what they really want.

Therefore, know your stakeholders. Better yet, work closely with them to ensure that you know

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

their needs. Start by identifying all stakeholders, and then maintain open and frequent communication and collaboration between them and the development team.

Separate the problem from the solution

Too often, we run headlong into a solution without understanding the problem. After all, we are taught how to solve problems, not how to define problems, so that's easier. However, this limits our understanding of the problem, imposes artificial constraints, and makes it difficult to balance trade-offs, or to even know what the trade-offs are.

Therefore, make sure that you understand the problem before you define the solution. By clearly separating the problem (what the customer needs) from the solution (what the system must do), it is easier to maintain the proper focus and easier to accommodate alternate ways of solving the problem.

Create a shared understanding of the domain

Domain experts often have limited technical expertise; developers, architects and testers often have limited domain expertise; and reviewers and other stakeholders often have limited time to commit to the project and learn the problem domain. As a result, people often have an inconsistent or poor understanding of the problem domain, which causes communication problems and increases the likelihood of delivering poor value to the stakeholders.

Therefore, enhance and share all parties' understandings of the domain. A concise and shared understanding of the problem domain enhances communication and project effectiveness. Start by defining the problem in the Vision document. As your understanding increases, capture key domain concepts and terminology in the Glossary to ensure a consistent shared use of the language of the domain.

Use scenarios and use cases to capture requirements

Many companies still document requirements as a list of declarative statements, which are sometimes called the "shall statements." These lists are often difficult for stakeholders to understand, because they require the end user to read through and mentally translate the list into a visualization of how the requirements will interact with the system. .

Therefore, use scenarios and use cases to capture functional requirements in a form that is easy for stakeholders to understand. Nonfunctional requirements, such as performance, stability, or usability requirements, are important and can be documented in the Supporting Requirements, using traditional techniques.

Establish and maintain agreement on priorities

Making poor decisions in deciding what to develop next can result in wasted effort, delivering capabilities that are never used, or identifying problems late in the project that result in delays and even project failure.

Therefore, prioritize requirements for implementation by regularly working with the stakeholders during product evolution. Make choices that deliver value and reduce risks, while building a system that can evolve.

Make the trade-offs to maximize value

Cost-benefit trade-offs cannot be made independent of the architecture. Requirements establish the benefits of the system to the stakeholder, while architecture establishes the cost. The cost of a benefit may influence the stakeholder's perceived value of the benefit.

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

Therefore, work with the stakeholders and developers to prioritize requirements and develop candidate architectures to implement those solutions. Use the candidate architectures to evaluate the cost of the benefits. Candidate solutions are considered at a high level when determining architectural feasibility. Different architectural perspectives can result in different assessment of cost versus benefit. The candidate architecture that provides the most coverage at the lowest cost is selected for further development.

Manage scope

Change is inevitable. Although change presents opportunities to enhance stakeholder value, unconstrained change will result in a bloated, deficient system and unmet stakeholder needs.

Therefore, manage change while maintaining the agreements with the stakeholders. Modern processes always manage change, continually adapting to changes in the environment and stakeholder needs, assessing the impact of changes, making trade-offs, and re-prioritizing work. Stakeholder and developer expectations must be realistic and aligned throughout the development lifecycle.

Know when to stop

Over-engineering a system not only wastes resources but also leads to an overly complex system.

Therefore, stop developing the system when the desired quality is achieved. Remember that “Quality is conformance to the requirements”. This is what gives a sense of closure to the practice. Separate the problem from the solution, ensuring that the solution does, indeed, solve the problem. After the critical requirements are implemented and validated, the system is ready for stakeholder acceptance.

Appendix D - OpenUP/Basic Core Concept: Collaborate to align interests and share understanding

The following is part of the OpenUP/Basic version 0.9 published version 20061002, under Eclipse Public License v1.0

Key principle

Develop collaborative practices that foster a healthy team environment. Good collaborative practices align the interests of project participants and help them develop a shared understanding of the project.

Introduction

Software is created by people with different interests and skills who must work together to create software effectively.

Therefore, develop practices that foster a healthy team environment. A healthy team environment enables effective collaboration, which aligns the interests of project participants (development team, quality assurance, product stakeholders, customers) and helps project participants develop a shared understanding of the project.

Practices

The next sections describe the practices associated with this principle.

Maintain a common understanding

Project participants require a common understanding of a project to cooperate effectively. Otherwise, disorder sets in, because the team members cannot align their understanding and interests and will work with different purposes.

Be proactive communicating and sharing information with project participants and do not assume that everyone will just find what they need to know or that each person has the same understanding of the project as everyone else. Use work products, such as the Vision, Work Items List, and Requirements to align the understanding between the stakeholders and developers. Use the architecture to focus and align the interests of the developers. At the end of each iteration, get agreement on whether the iteration goals have been reached, and, if not, what actions must be taken.

Foster a high-trust environment

People who do not feel safe will not communicate their ideas, take the initiative, or admit their ignorance. In these low-trust work environments, activities must be laboriously planned in detailed, carefully supervised, and extensively audited. A team working in a low-trust environment may not be able to keep up with rapid change.

Therefore, take actions that foster a high-trust environment:

- **Manage by intent.** Create an environment where teams manage themselves, and managers serve as mentors to teams to help them complete their objectives.
- **Tear down the walls.** Work to remove both the physical and mental barriers that inhibit

development of a common understanding among project participants.

- **Walk a mile (or 1.6 kilometers) in someone else's shoes.** Respect and try to understand the perspectives of others before criticizing their ideas or responding to their criticism.
- **Respond to conversations with relevance.** People, especially technical workers, often respond with argument or disagreement, which leads to rivalry and the establishment of a pecking order in which only a few people contribute to the discussion. Develop and encourage behavior that values curiosity and relevance over argument and disagreement.
- **Always look to yourself first for the source of communication problems.** Understand that everyone has a perspective that is largely invisible to the individual (although it is often obvious to everyone else). Develop the habit of identifying the assumptions and prejudices within yourself that lead to argument or lack of communication. Learn to overcome these in the moment of the conversation. This takes practice. There are times when others may be intractable, but often the problem can be addressed by wrestling with your own perspective.
- **Understand the constraints of the workplace culture.** Some organizations operate in a way that allows people to admit mistakes, ask questions, and experiment. Some organizations limit these expressions, but they may change, with time and effort. Some organizations have no tolerance for error, and workers put themselves in danger by admitting mistakes or experimenting. Understand your environment and protect yourself accordingly. Understand that low-trust organizations have more problems in achieving their goals and provide a less satisfying environment.

Share responsibility

There may be disadvantages for individuals when they work alone. Communication with the team can become sporadic, and then stop. People may get into trouble and not ask for help, or not realize that the team is in trouble and needs their help. Their understanding of the project may become misaligned with the rest of the team. In the worse situations, trust breaks down as individuals see the team working at different purposes to their interests.

Therefore, while individuals have primary responsibility for their work products, responsibility for work products is shared. Nothing is someone else's responsibility. This may mean either taking up slack and working with someone who is lagging for some reason or asking for help. Experienced staff should be extra-vigilant and watch over less-experienced staff, encouraging them to ask for help if necessary.

Learn continuously

Not only is software development a fast-developing field where technical skills rapidly become obsolete, it is also an empirical process, where software is developed in a manner that sometimes resembles trial and error. Furthermore, software is developed by teams of people who must work together to achieve results.

Therefore, continuously develop both your technical and interpersonal skills. Learn from the examples of your colleagues. Take the opportunity to be both a student of your colleagues, as well as a teacher to them. Always increase your personal ability to overcome your own antagonism toward other team members.

Organize around the architecture

As projects grow in size, communication between team members becomes increasingly complex. While all team members understand the overall system, they can focus primarily on the

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

one or more subsystems they are responsible for. Organizing around the architecture also helps with communication by providing the team with a common vocabulary and shared mental model of the system. Communication between team members becomes increasingly complex. Therefore, organize the team around the architecture and the vocabulary and shared mental model of the system. However, be watchful that individuals and teams organized this way do not form a so-called *silo mentality*, where they focus strictly on their subsystems and become ignorant of the other subsystems.

An architecture that reflects the organization's structure is not evidence that the team has successfully organized around the architecture. If organizations and teams are not organized around the architecture, then the architecture will naturally conform to the organization, as a result of political and cultural influences. In the end, the architecture and the organization will almost always be a reflection of each other. The goal is to guide team organization from the needs of the architecture as much as possible.

Appendix E - OpenUP/Basic Core Concept: Focus on articulating the architecture

The following is part of the OpenUP/Basic version 0.9 published version 20061002, under Eclipse Public License v1.0

Key principle

Articulate essential technical decisions through a growing architecture.

Introduction

Without an architectural foundation, a system will evolve in an inefficient and haphazard way. Such a system often proves difficult to evolve, reuse, or integrate without substantial rework. It is also difficult to organize the team or communicate ideas without the common technical focus that the architecture provides.

Therefore, use the architecture as a focal point for developers to align their interests and ideas by articulating the essential technical decisions through the growing architecture.

Practices

The next sections describe the practices associated with this principle.

Create the architecture for what you know today

As Albert Einstein said, make everything as simple as possible but no simpler. Software projects are resource-constrained, and the desire of developers to create elegant solutions may lead to a system of greater complexity than the stakeholder requires. Efforts to future-proof a system in a turbulent or uncertain environment will likely lead to code bloat , thus increasing overall cost with little actual benefit to show for it.

Therefore, create architectures that address the stakeholder's real needs, and provide appropriate flexibility and speed for the requirements as they are known today. Avoid the desire, no matter how well intentioned, to speculate on future requirements and thereby over-engineer the architecture: if you have the skill to architect something today, then clearly you must also have the skill to architect it tomorrow when you actually need to.

Cope with complexity by raising the level of abstraction

Software is complex, and people have a limited capacity for coping with complexity. As a system gets larger, it becomes difficult for the team to develop a common understanding of the system, because it is hard to see the bigger picture.

Therefore, use models to raise the level of abstraction to focus on important high-level issues, such as relationships and patterns, rather than getting bogged down in details. Modeling raises the level of abstraction and allows the system to be more easily understood from different perspectives.

Let the problem drive the solution

The architecture may become difficult to maintain and adapt to new stakeholder needs when technology, rather than the problem, drives the solution.

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

Therefore, let the needs of the stakeholders guide the architecture, instead.

Organize the architecture into loosely coupled, highly cohesive components

Tight coupling between components makes a system fragile and difficult to understand. Software is expensive to create, so if existing components can be reused, that may reduce effort required to create a system.

Therefore, organize the architecture of the system into components that to maximize cohesion and minimize coupling. This improves comprehension, increases flexibility, and increases opportunities for re-use.

Reuse existing assets

It is wasteful to build what you can simply reuse, download, or even buy.

Therefore, make every effort to reuse existing assets. Developers are often reluctant to reuse assets, because those assets do not exactly meet their needs or those assets are of poor quality. Be prepared to balance the savings you can realize using an existing asset, even if the asset requires distorting the architecture or relaxing a constraint.

Leverage the architecture as a collaborative tool

Lack of a common understanding by developers about a system leads to indecision and contrary opinions among developers and can quickly paralyze the project. Developers may have different mental models of the system and work at cross purposes to each other.

Therefore, create and evolve the system architecture with the intention of using it to align the developer's competing mental models of the system. A good architecture facilitates collaboration by providing a common vocabulary for all discussions regarding the system under development.

Appendix F - OpenUP/Basic Core Concept: Evolve to continuously obtain feedback and improve

The following is part of the OpenUP/Basic version 0.9 published version 20061002, under Eclipse Public License v1.0

Key principle

Divide the project up into short, time-boxed iterations to demonstrate incremental value and obtain early and continuous feedback.

Introduction

It is usually not possible to know all stakeholders' needs, be aware of all project risks, comprehend all project technologies, or know how to work with your colleagues. Even if it were possible to know all of these things, they are likely to change over the life of the project. Therefore, divide the project into short, time-boxed iterations to demonstrate incremental value and to get early and continuous feedback.

The intention behind this principle is to continuously get feedback and to improve both the product and the process of the project team. When you provide structure and create a mindset for continuous feedback and improvement, changes are accommodated more easily, feedback is captured early and often, high-priority risks are confronted early in the project. By constantly identifying and attacking risks, there is more confidence in project progress and quality.

Not only does the product evolve, but the team also finds better ways to work together and get involved with stakeholders. The process followed by the team can be adjusted accordingly to leverage lessons learned and adjust project pace and needs.

Practices

The next sections describe the practices associated with this principle.

Develop your project in iterations

Developing a system in a single, linear pass is difficult, because it makes it expensive to incorporate changes and new knowledge. Worse, it can delay the discovery and mitigation of risks, because development efforts are scheduled later in the lifecycle.

Therefore, divide your project into a series of time-boxed iterations, and plan your project iteratively. This iterative strategy enables you to incrementally deliver capabilities (such as an executable, usable subset of implemented and tested requirements) that can be assessed by stakeholders at the end of each iteration. This provides rapid and timely feedback loops so that issues can be addressed and improvements made at a lower cost. Also, this is accomplished while you still have sufficient budget and time left to do so, and you have not gone so far ahead that major rework is required. Iterative development enables teams to continuously improve software throughout the development lifecycle.

Focus iterations on meeting the next management milestone

Without a focus to bring closure to important project issues, such as stakeholder concurrence regarding scope and proving the proposed architecture, a project can appear to make progress while risks and unresolved issues pile up.

Therefore, divide the project into phases (such as Inception, Elaboration, Construction, and Transition), with each phase having a clearly visible management milestone. The focus of each iteration within a phase is on achieving that milestone.

Manage risks

Deferring difficult and risky issues until later in a project significantly increases the risk of project failure. Such procrastination may lead to investing in the wrong technologies, a bad design, or a set of requirements that may not address stakeholder needs.

Therefore, attack risks early, or they will attack you. Continuously identify and prioritize risks, and then devise strategies to mitigate them. Determine the focus of iterations based on risks. For example, architecturally significant risks should be addressed early in the project, no later than the end of Elaboration phase, when the architecture has been proven and baselined.

At the beginning of each iteration, the entire team should consider what risks they are facing, and update the risk list. Make it each team member's and stakeholder's responsibility to have the courage to speak up and openly discuss risks, as well as to have the courage not to criticize the people who do speak up, even though the risk may point to a flaw in their area of responsibility. For each risk, articulate a plan for tracking and mitigating the risk.

Embrace and manage change

Change is inevitable, and while change presents opportunities to enhance stakeholder value, unconstrained change will result in a bloated, deficient system and unmet stakeholder needs. Furthermore, the later in the development cycle a change is made, the more the change is likely to cost.

Therefore, both embrace and manage change. Embracing change helps you to build a system that addresses stakeholder needs, and managing change allows you to reduce costs and improve predictability of those changes. Changes made early in the project can usually be made with limited cost. As you progress in your project, changes can become increasingly costly.

To satisfy customer needs, you typically need to introduce changes to the project, but the customer must be made aware of the impact that those changes have on the project cost and schedule. Understand the impact of a change in the current phase, and isolate team members from disruptive changes during the current iteration. Change requests are reviewed and prioritized during the current iteration, but are not acted upon until assigned to a future iteration.

If necessary, document the changes. For informal projects, a discussion with stakeholders may be enough.

Measure progress objectively

If you do not objectively know how your project is progressing, you do not really know if it is failing or succeeding. Uncertainty and change make a software project's progress difficult to measure objectively, and people have a most amazing ability to believe all is well in the face of catastrophe.

Iterative Methodologies to Manage Web-Based Software: Focus on OpenUP/Basic

Therefore, get a clear picture of project status by objectively measuring progress. The best measure of progress is the delivery of working software, which is something that you do by taking an evolutionary approach. You can also define a set of objective metrics to collect during an iteration (for example, requirements that were implemented and validated, number of defects issued compared with number fixed) and review them as part of the iteration assessment. Do not rely on single metrics. Rather, use a combination of metrics and look for trends.

Continuously re-evaluate what you do

People make mistakes during a project. If we chose to hide those mistakes, then we risk repeating the same mistakes. In addition, such repressed social dynamic issues can poison the team.

Therefore, on a regular basis, ask questions and verify assumptions about the project. Regularly meet with the team to track status and identify risks and issues. This can be done daily when the team gathers to share the status of individual responsibilities and identify and address issues. At the end of iterations, assess the status of what has been done and look for areas of improvement that can be addressed in the next iteration. Have a retrospective review at the end of the project and capture lessons learned to run future projects in a more efficient way.

If we always challenge what we do and seek new, innovative ways to develop software, we improve how we work. This leads to improved project results.