Métodos Agiles para el Desarrollo de Software

Introducción

Software Development vs. Predictable Manufacturing

Predictable Manufacturing	**New Product Development**
It is possible to first complete specification and then build	Rarely possible to create up-front unchanging and details specs
Near the start, one can reliably estimate effort and cost	Near the beginning, it is not possible As empirical data emerge, it becomes increasingly possible to plan and estimate
It is possible to identify, define, schedule And order all the detailed activities	Near the beginning, it is not possible. Adaptive steps driven by build-feedback cycles are required
Adaptation to unpredictable change is Not the norm, and change-rates are relatively slow	Creative adaptation to unpredictable changes the norm. Change rates are high

Software Development

Most software is not a predictable or mass manufacturing problem. Software development is new product development

Software Development - Clients

- **The clients or users are not sure what they want**
- **They have difficulty stating all they want and know**
- **Many details of what they want will only be revealed during development**
- **The details are overwhelmingly complex for people**
- **As they see the product development, they change their minds**
- **External forces (such as a competitor's product or service) lead to changes or enhancements in requests**

Agile Software Development

- In the late 1990´s several methodologies began to get increasing public attention
- Each had a different combination of old ideas, new ideas, and transmuted old ideas
- Agile methods is principle-based than rule-based
- Agile methodologies follows developments adaptable, collaborative, delivery driven, people oriented, customer focused, guided by a vision, develop in short iterations, change requirements, communication, self-organizing teams, adaptive planning, test-driven development and continuous integration

Agile Software Development

- Agile movement is not anti-methodology, in fact, it want to restore credibility to the word methodology
  - It want to restore a balance
  - It embrace modelling but not in order to file some diagram in a dusty corporate repository
  - It embrace documentation, but not hundreds of pages of never-maintained and rarely-used tomes
  - It plan, but recognize the limits of planning in a turbulent environment

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping other do it. Through this work we have come to value:

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

That is, while there is value in the item on the right, we value the items on the left

Individuals over Process

- Agile methodologies remind that programming is a human activity
- Although AM emphasize individuals over process, the set of practices in an AM addresses the same kind of planning and commitment issues as the focus on basic project management at CMM Level2
- AMs provides just enough process to gain a reasonable payoff
- This goal is promoted by the emphasis on communication, with face-to-face conversation and daily meetings

Working Software over Documentation

- The idea is to have non-essential documentation rather than saying that the documentation is an inefficient waste
- The vital objective of the team is to continuously turn out tested working software. New releases are produced at frequent intervals, in some approaches even hourly or daily, but more usually bi-month or monthly
- The developers are urged to keep the code simple, and technically as advanced as possible, thus lessening the documentation burden to an appropriate level

Customer Collaboration over Contracts

- The relationship and cooperation between developers and clients is given preference over strict contract
- The negotiation process itself should be seen as a means of achieving and maintaining a viable relationship
- A barrier is likely to be an inability to establish and maintain close and effective customer collaboration
- Because of the changes the customer collaboration helps to re-organize the system

Responding to Change over Planning

- Planning for change is quite different from not planning at all
- Agile methodologies with their rapid iterations, require continual planning
- The group should be well-informed, competent and authorized to consider possible adjustment needs emerging during development
- One of the shifts in acquisition strategy in recent years has been toward prototyping, evolutionary development, and risk-driven life cycle. With their emphasis on addressing requirements volatility

Principles behind the Agile Manifesto (1)

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Principles behind the Agile Manifesto (2)

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

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

Working software is the primary measure of progress.

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

Principles behind the Agile Manifesto (3)

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Iterative & Evolutionary

Agile methods are a subset of iterative and evolutionary methods

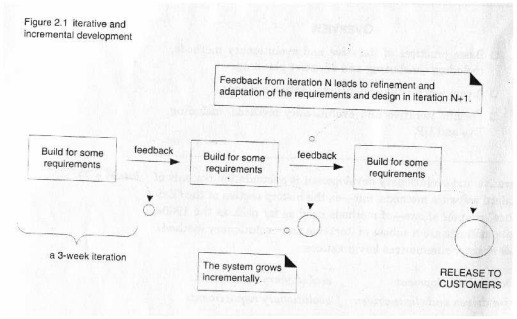
Iterative Development

- **Iterative development is an approach in which the overall lifecycle is composed of several iterations in sequence**
- **Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming and test**
- **The goal for the end of an iteration is an iteration release, a stable, integrated and tested partially complete system**
- **Most iterations releases are *internal***
- **The final iteration is the complete product, released to the market or clients**

Iterative & Incremental Development (IID)

- **The system grows incrementally with new features, iteration by iteration: Incremental Development**
- **Most projects have at least three iterations before a final public release**
- **In modern iterative methods, the recommended length of one iteration is between one and six weeks**
- **The software resulting from each iteration is not a prototype or proof concept, but a subset of the final system**

Iterative & Incremental Development



Projects Success Factors and IID

Success Factor	Weight of Influence
User involvement	20
Execution support	15
Clear business objectives	15
Experienced project manager	15
Small milestones	10

23.000 projects analysed

5 from the top ten factors for success are strongly related to IID practices

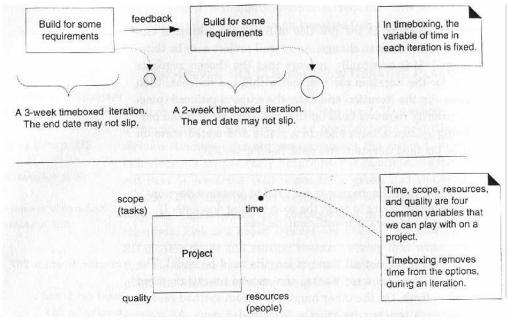
Risk-Driven and Client-Driven Iterative Planning

- **Risk-driven iterative development: Chooses the riskiest, most difficult elements for the early iterations. The highest risks are surfaced and mitigated early rather than late**
- **Client-driven iterative development: Implies that the choice of features for the next iteration comes from the client. The client steers the project, iteration by iteration, requesting the feature that the currently think are most valuable**
- **Apply both schemes: Clients do not always appreciate what is technically hard or risky. Developers do not always appreciate what has high business value.**

Timeboxed Iterative Development

- **Iteration timeboxing is the practice of fixing the iteration and date and not allowing to change. As the overall project**
- **If it eventually appears that the chosen requests for the iteration can't be met within the timebox, then rather than slip the iteration and date, the scope is reduced**
- **The timeboxing is not used to pressure developers to work more hours. If the normal pace work is insufficient, do less**
- **Not all timebox lengths need be equal**
- **Once the requests for an iteration have been chosen and it is underway, no external stakeholders may change the work**

Timeboxed Iterative Development



Timeboxed Benefits

- **People remember slipped dates not slipped features**
- **Delay a project three months from its original end date to include 100% of the desired features, and the "failure" will be remember. Deliver on time with 75% of the most important features and it's a success**
- **With a short two-week timeboxed iteration the team takes on manageable complexity, gets realistic about what they can do**
- **Early forcing of difficult decision and trade-offs**

Evolutionary and Adaptive Development

- **Evolutionary iterative development: Implies that the requirements, plan, estimates and solution evolve or are refined over the course of the iterations.**
- **Adaptive development: Implies that elements adapt in response to feedback from prior work-feedback from users, tests, developers, and so on**

Some methods or methodologies emphasize the term "iterative" while others use "evolutionary" or "adaptive". The ideas and intent are similar, although, evolutionary and adaptive development does not require the use of timeboxed iterations

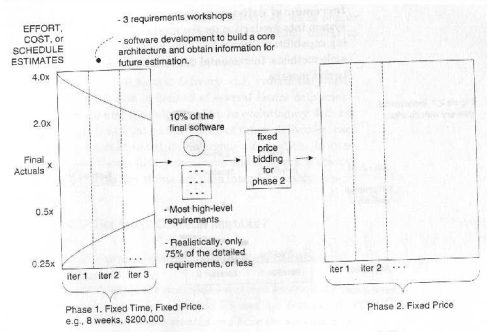
Evolutionary Development

- It is not true that 100% of the functional requirements need be known to start building the core architecture.
- The architect needs to know most nonfunctional or quality requirements and a smaller subset of representative functionality

Fixed-Price Contracts

- IID methods recommend running projects in two contract phases, each of multiple timeboxed iterations
- The first phase, a short fixed-time and fixed-price contract, has the goal of completing few iterations, doing early but partial software development and evolutionary requirements analysis. Partial software is produced, not merely documents
- The outputs of phase one are then shared with bidders for a phase two fixed-price contract

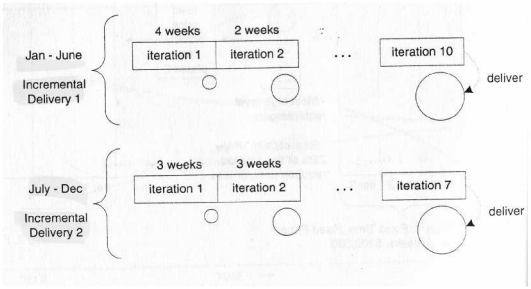
Fixed-Price Contracts



Incremental Delivery

- Incremental delivery is the practice of repeatedly delivering a system into production
- Incremental deliveries are often between three and twelve months
- A six-month delivery cycle could be composed of 10 short iterations
- The result of each iteration are not delivered to the market-place, but the results of an incremental delivery are

Incremental Delivery



Evolutionary Delivery

- **Evolutionary delivery is a refinement of the practices of incremental delivery in which there is an attempt to capture feedback regarding the installed product and use this to guide the next delivery**
- **The goal is to best meet some difficult-to-predict need, such as the most frequently requested new feature**
- **In evolutionary delivery there is no plan of future deliveries; each is dynamically created based on emerging information**

Agile Development

- **Agile development methods apply timeboxed iterative and evolutionary development, adaptive planning, promote evolutionary delivery and flexible response to change**
- **Short timeboxed iterations with adaptive, evolutionary refinement of plans and goals is a basic practice various methods share**
- **They promote practices and principles that reflect and agile sensibility of simplicity, lightness, communication, self-directed teams, programming over documenting and so on**

Agile Software Important Concepts

- **Close collaboration between the programmer and business experts**
- **Face-to-face communication**
- **Frequent delivery of new deployable business value**
- **Self-organization teams**
- **Refactoring. Restructuring the code in order to improve iterations integration**
- **Embracing change. Seeing change as an ally rather than an enemy**
- **Simple design. Designing for a battle, not for a war**

Agile Software Important Concepts

- **Simple design. Designing for a battle, not for a war. Two assumptions:**
  - **The cost of rework to change the software (refactoring), to support new, possibly unanticipated, capabilities will remain low over time**
  - **The application situation will change so rapidly that any code added to support future capabilities will never be used**

Agile Development Advices

- **Two to eight people in one room. Communication and community**
- **Onsite usage experts. Short and continuous feedback cycles**
- **Short increments. One to three months, allows quick testing and repairing**
- **Fully automated regression tests. Unit and functional tests stabilize code and allow continuous improvement**
- **Experienced developers. Experience speed up the development time from 2 to 10 timer compared to slower team members**

Agile Development Designed to

- **Produce the first delivery in weeks, to achieve an early win and rapid feedback**
- **Invent simple solutions, so there is less to change and making this changes is easier**
- **Improve designs quality continually, making the next iteration less costly to implement**
- **Test constantly, for earlier, less expensive, defect detection**

Manager Role in Agile Methods

- **The manager <u>does not</u> create a work breakdown structure, schedule, or estimates; this it done as a team**
- **The manager <u>does not</u>, usually, tell people what to do**
- **The manager <u>does not</u> define and assign many detailed team roles and responsibilities**
- **Project manager role emphasizes coaching, servant-leadership, providing resource, maintaining the vision, removing impediments, promoting agile principles etc**