The Agile Software Development Process

A Process

- A process defines the values, principles and practices used to achieve a goal
- It aims to:
  - Promote best practices
  - Reduces risk
  - Increase productivity
  - Satisfies customers' real needs
  - Infuse a common vision and culture in a team
- Software engineers are highly skilled individuals and how they work as a team is defined by software development processes

What is a Software Development Process?

- A development process is the way one organises the creation of software systems
  - Where you start
  - How you gather requirements (interviews, brainstorming sessions, information resources, etc.)
  - The number and types of milestones during development
  - The deliverables at each stage
  - How it is determined that each stage is complete
  - And generally how you go about the entire project
  - The project manager + all team members are involved
  - Good approach is to select a suitable existing development process, and tailor it to your needs

Aspects of a Software Process

- A software development process deals with:
  - People - A wide-range of people, with different skill-sets, are involved
  - Technologies - Infrastructure upon which software will be based
  - Tools - Software development and project management tools
  - Organisational Patterns - How team members interact
- A balance is needed to create a stable process
  - Aim is to produce a product which satisfies current requirements and is flexible for future enhancements
  - Everything else is secondary
    (this does not mean irrelevant)

Agile Software Development

- Strives for a more “agile” approach to software development
- More emphasis on getting the right software written fast, rather than strict adherence to lots of “procedure”
- Get the productive involvement of all stakeholders (co-operative effort)
- Flexibility
- Not a single approach, but a family of techniques sharing many interesting ideas

XP < AP > UP

- We define the generic term “Agile Process” (AP) to mean all the processes that lie between XP and UP
- UP has most of the ideas of agile development
  - Can be tailored down (most users do not do so)
- XP has some good ideas, but it is not tolerant to adaptability(
  - To perform XP, you must follow all its practices
  - If you ignore one (e.g. pair programming) then that will have detrimental effects (tacit knowledge is not being shared)
  - General agile processes are more tolerant of local tuning
- Many overlapping ideas in the agile processes though each has some interesting unique features

What is an Agile Process?

- In this presentation we examine each of these agile processes

- CRYSTAL
- Lean Development
- Agile Project Management (APM)
- Agile Modelling (AM)
- SCRUM
- Feature Driven Development (FDD)
- Agile + Discipline
  - Not really a process, but discusses the boundaries between agile and plan-driven

Crystal

- Proposed by Alistair Cockburn in his book:
  - “Crystal Clear: A Human-Powered Methodology for Small Teams”, ISBN: 0201699478, Addison Wesley
  - Very highly recommended!
- Cockburn's Synopsis:
  - “The lead designer and 2-7 other developers in a large room or adjacent rooms, with information radiators such as whiteboards and flipcharts on the wall, having access to key users, distractions kept away, delivering running, tested, usable code every month or two (OK, three at the outside), periodically reflecting and adjusting on their working style.”

Properties

- Frequent Delivery
- Osmotic Communication
- Reflective Improvement
- Personal Safety
- Focus
- Easy Access to Expert Users
- Technical Environment
    - Automated tests, config management, frequent integration
- Collaboration across organisational boundaries

Strategies

- Exploratory 360º
- Early Victory
- Walking Skeleton
- Incremental Rearchitecture
- Information Radiators

Techniques

- Methodology Shaping
- Reflection Workshop
- Blitz Planning
- Delphi Estimation using Expertise Rankings
- Daily Stand-up Meetings
- Essential Interaction Design
- Process Miniature
- Side-by-side Programming
- Burn Charts

Cycles

- The Project Cycle
- The Delivery Cycle
- The Iteration Cycle
- The Integration Cycle

Lean Software Development

- Proposed by Mary & Tom Poppendieck in their book
  - “Lean Software Development – An Agile Toolkit”, ISBN: 0-321-15078-3, Addison Wesley
- Brings ideas from new product development in highly innovative companies such as 3M and Toyota to software
- Consists of 7 lean principles and 22 knowledge tools
  - Discussed in detail in chapters 1-7 of the Poppendiecks' book – we will summarise over the new few slides

1. Eliminate Waste

- “Anything that does not create value for a customer is waste”
- Tool 1 – Seeing Waste
  - The seven primary wastes in software development are: partially done work, extra processes, extra features, task switching, waiting, motion and defects
- Tool 2 – Value Stream Mapping
  - Identify how value gets added as a customer request flows through development process (and identifies non-value adding activities and waiting periods)

2. Amplify Learning

- Writing software is development, not production
  - The root problem with many heavy-weight methodologies is that they approach software as a production problem
  - Variability is expected during development, to be avoided during production
- Tool 3: Feedback
    - Learning from early effort can be extremely helpful later
- Tool 4: Iterations
    - Using iterations incorporates learning
- Tool 5: Synchronisation
    - Frequently merging work of team (avoids late breakage)
- Tool 6: Set Based Development
    - Keep design options open within constraints

3. Decide As Late As Possible

- Depth-first vs. breath-first design approaches
- Tool 7: Options Thinking
  - Predictive processes and adaptive processes
- Tool 8: The Last Responsible Moment
  - Making decisions too hastily or too late is not recommended
  - You will make the best decision just after you have learned as much as possible about the situation, and just before the decision is really needed
- Tool 9: Making Decisions
  - Intuitive decision making
- Give professional developers the training, resources and authority and the team will come up with the right decisions

4. Deliver As Fast As Possible

- Why do customers like to see things happen quickly?
- Tool 10: Pull Systems
  - “Let customers' needs pull the work rather than have a schedule push the work”
- Tool 11: Queuing Theory
  - Cycle time is how long it takes for piece of work to get done
  - Modify through rate of arrival & rate of service of workitem
  - Smaller workitems lead to shorter cycle times
- Tool 12: Cost of Delay
  - First to market is very valuable so need to seriously consider total costs of delays

5. Empower The Team

- Frederick Taylor's Scientific Management
  - Measure workers' actions with stopwatches
  - The problems with CMM and ISO 9000
- Tool 13: Self-Determination
  - Managers coach/train/assist/support/protect/resource teams
  - Teams make their own decisions to tackle the work
- Tool 14: Motivation
  - Helped by clear purpose, access to customers to appreciate needs; own commitments
- Tool 15: Leadership (setting direction)
- Tool 16: Expertise
  - Developing and sharing expertise across team

6. Build Integrity

- Tool 17: Perceived Integrity
  - “totality of the product achieves a balance of function, usability, reliability and economy that delights customers”
- Tool 18: Conceptual integrity
  - “system's central concepts work together as a smooth cohesive whole”
- Tool 19: Refactoring
  - Good design happens throughout a project – refactoring is how it gets into the existing codebase
- Tool 20: Testing
  - Ensures product satisfies customers' demands

7. See The Whole

- Heavy-weight processes will not solve problems
- Tool 21: Measurements
  - Avoiding sub-optimisation
  - Measuring unstructured work
  - What to measure
- Tool 22: Contracts
  - Trust across organisational boundaries
  - Contract types: Fixed-Price/Fixed-Scope, Time and Materials, Multistage, Target-cost, Target-schedule, Shared benefit
  - Handling Optional Scope

Agile Project Management (APM)

- Proposed by Jim Highsmith in his book
  - “Agile Project Management”, ISBN: 0-321-21977-5, Addison Wesley, 2004
- How to guide agile projects according to agile principles and practices
- Adaptive ecosystems
- Exploring & experimenting
- Delivering customer value

Key Business Objectives

- Continuous Innovation
- Product adaptability
- Reduced delivery schedule
- People and process adaptability
- Reliable results

Selecting Practices

- Practices need to be:
  - “Simple
  - Generative, not prescriptive
  - Aligned with agile values and principles
  - Focused on delivery (value adding), not compliance
  - Minimum set (just enough to get the job done)
  - Mutually Supportive (a system of practices)”
    - APM Book – page 86

Envision Phase

- Product Vision Box and Elevator Test Statement
- Product Architecture
- Project Data Sheet
- Get the Right People
- Participant Identification
- Customer Team / Developer Team Interface
- Process and Practice Tailoring

The Speculate Phase

- Product Feature List
- Feature Cards
- Performance Requirements Cards
- Release, Milestone and Iteration Plan

The Explore Phase

- Workload Management
- Low-cost Change
- Coaching and Team Development
- Daily Team Integration Meetings
- Participatory Decision Making
- Daily Interaction with Customer Team

The Adapt and Close Phase

- Product Review
- Project Review
- Team Review
- Adaptive Action

Building Adaptive self-organ ising/self-disciplined teams

- “Getting the right people
- Articulating the product vision, boundaries and team roles
- Encouraging interaction and information flow between teams
- Facilitating participatory decision making
- Insisting on accountability
- Steering, not controlling”
    - APM Book – page 65

Agile Modelling (AM)

- Proposed by Scott Ambler in his book
  - “Agile Modeling”, ISBN: 0-471-20282-7, Wiley, 2002
- Definition
  - “Agile modeling (AM) is a chaordic, practice-based methodology for effective modelling and documentation of software-based systems”
  - Meaning of chaordic - Between chaos and order
    - Which is exactly where most software projects are for most of their lifetime
    - Chaos of simple models and order of more formal software modelling artefacts

Core Principles

- Software is the primary goal
- Setting up for the next effort is the secondary goal
- Travel light
- Assume simplicity
- Embrace change / Incremental change
- Model with a purpose / Multiple models
- Quality work
- Rapid feedback
- Maximise stakeholder investment

Practices

- Iterative and incremental modelling
  - Using the right artefact, creating multiple models in parallel, modelling in small increments
- Effective Teamwork
  - Modelling as a team, stakeholder participation, collective ownership, public display of models,
- Enabling simplicity
  - Simple content, depicting simple models, simple tools
- Validating work
  - Testability, proving concepts with running code
- Improving productivity
- Agile documentation

Documenting Agile Projects

- Chapter 14 of “Agile Modeling” contains an interesting discussion of how to document agile projects
  - Need to create documentation to satisfy customer, to define interface boundaries, to communicate with external groups, to support future development team members and to help evaluate designs
  - Difference between permanent and temporary documents
  - The model and document concepts are orthogonal
- For a more extensive discussion, also see this book:
  - “Agile Documentation”, Andreas Ruping, ISBN: 0-470-85617-3, Wiley, 2003

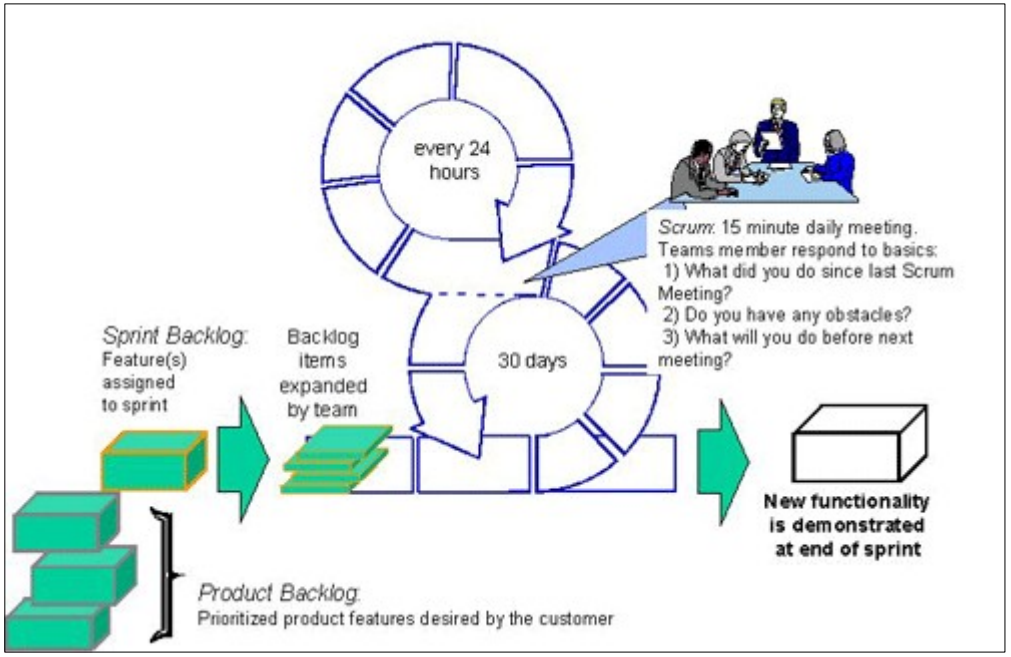
Scrum

- Created by Ken Schwaber and Jeff Sutherland
  - Good book is “Agile Project Management with Scrum”, Ken Schwaber, ISBN: 0-7356-1993-X, Microsoft Press, 2004
- “Scrum” metaphor from the world of rugby
  - Team working together towards some common goal
- Very simple and compact approach, which means it is easy for entire teams to quickly pick it up and start using it

Principles

- Self-organisation
- Emergence
- Visibility
- The Inspection/Adaption cycle
- Paradigm shifts
    - From control to empowerment
    - From contracts to collaboration
    - From documentation to code

Scrum Diagram



- Diagram from http://www.controlchaos.com/about/

Every 24 Hours

- Scrum is based around two cycles – daily & monthly
- Daily scrum is work completed in one day
- Team briefly (daily inspection) meets once a day to answer three simple questions (daily scrum meeting)
  - “What have I done since the last Daily Scrum?
  - What am I going to do between now and the next daily scrum?
  - What is preventing me from doing my work?”

Every 30 Days

- The monthly cycle is known as a sprint
- Starts with sprint planning meeting and concludes with sprint review meeting (monthly inspection)
- Delivers increments of potentially shippable product functionality
  - Product owner may decide whether to actually ship (make the iteration a release)
- Select items from the product backlog that Product Owner wishes, and that can be completed within the next sprint
- Review ensures Product Owner is satisfied with what has been delivered

Sprint Planning Meeting

- Occurs at start of each sprint (once a month)

- 4 hour session for selecting product backlog
  - ScrumMaster, ProductOwner, Team
  - Team provides estimates for items
  - ProductOwner selects what is in each sprint
- 4 hour session for preparing a sprint backlog
  - Team creates a list (sprint backlog) of tasks and assignments that must be completed to deliver features
  - Sprint backlog will evolve during sprint

Sprint

- Team can solicit help from external people
  - To learn more, evaluate alternatives, discussions
- Once team has committed to delivering sprint backlog, the team (and no one else) is responsible for arranging the work (self management)
- List of items to be delivered during sprint is fixed
  - If, during the middle of a sprint, a Product Owner demands that a new item be implemented immediately, then the current sprint is aborted and a new sprint planned
- Estimated hours needed/used are correlated

Sprint Review & Retrospective Meetings

- Occurs at end of sprint
- Review with ProductOwner
  - (4 hours+1 hour preparation)
  - Demoing completed functionality (running code)
  - Work not competed or non-code artefacts cannot be demoed
  - Bi-directional communication between stakeholders & team
- Retrospective (3 hours)
  - Team and ScrumMaster – optionally ProductOwner
  - “What went well during sprint?”
  - “What could be improved in next sprint?”
  - Process improvement and tuning for local needs

Product Backlog

- To start a Scrum project, need a vision and a product backlog
  - Vision is a (e.g. 1-2 page) high-level statement defining the goals for the project, who it is aimed at, and its benefits
  - Product backlog is a list of functional and non-functional requirements / features
    (Scrum does not mandate exactly how requirements are specified – could be use cases or user stories)
  - Priorities and time estimates are attached to each item
  - Product backlog is divided into what is in the current sprint, which will likely be in the next immediate sprint and future (items are the current sprint are subject to ongoing change)

ScrumMaster

- Facilitates projects
  - Replaces the role of project manager and process engineer from other methods (though not equivalent to these)
  - Empowered teams are advised and coached by ScrumMaster, rather than directed or managed by him/her
- Ensures sprint can proceed with little external distractions
- High-level interaction with product owner

Product Owner

- Creates the product backlog
- Priorities the items
- Interacts with team during sprint
- Participates in sprint planning meeting
- Responsible for funding the team
- Plays the role of the customer
- Looks out for project's ROI (return on investment)

Development Team

- Self-directed
- Must focus on sprint's backlog (no additional work)
  - Often such extras are the root cause of missing deadlines
- Teamwork
- Team is jointly responsible for achieving progress

Notes On Scrum

- Interesting discussion of user stories and SCRUM is provided in chapter 15 of “User Stories Applied For Agile Software Development”, Mike Cohn, ISBN: 0-321-20568-5, Addison Wesley, 2004
- ScrumMaster certification available from: http://www.controlchaos.com

Feature-Driven Development (FDD)

- Proposed by Peter Coad, Eric Lefebvre and Jeff De Luca in their Book:
  - “Java Modeling Color with UML” (chapter 6), Prentice Hall, ISBN: 0-13-011510-X, 1999
- Covered in more detail in:
  - “A Practical Guide to Feature-Driven Development”, Stephen R Palmer & John M. Felsing, ISBN: 0130676152, Prentice Hall, 2002
- See interesting diagram on
  - http://www.featuredrivendevelopment.com/files/FDD%20Process%20Model%20Diagram.pdf

FDD Concepts

- Features are defined using this formula
  - <action> the <result> <by|for|of|to> a(n) <object>
  - e.g. Removal of an shopping cart item by the customer
- Feature sets are defined using this formula
  - <action>ing a(n) <object>
  - e.g. Editing the shopping cart
- Feature Teams are subsets of the development team that tackle specific features
- Archetypes (“a form from which all things of the same kind more or less follow”)
  - Contrast with inheritance and interfaces (“must follow”)
- Colour as an added dimension in UML

Process Model

- A simplified process that focuses on delivering features
- Five major steps
- Each has:
  - A clear entry criteria
  - A set of tasks (where the work is completed)
  - Verification steps (to ensure the task is really complete)
  - A clear exit criteria

1. Develop an Overall Model

- Entry – Team has been selected
- Tasks – Domain walkthrough, group models, team model, informal (initial) feature list
- Verification – Assessment of work
- Exit – Chief architect is satisfied with high-level model, architecturally significant sequence diagrams and notes

2. Build A Feature List

- Entry – “Develop an Overall model” process completed
- Tasks
  - Form the Feature List Team (who know about the application domain and development)
  - Create the feature list (consisting of feature sets, which in turn consist of features)
  - Subdivide large features; prioritise features
- Verification – Assessment of work
- Exit – Feature list completed

3. Plan By Feature

- Entry – “Build A Feature List” process completed
- Tasks
  - Planning team decides sequence of development
  - Feature sets are assigned to chief programmers
  - Classes are assigned to developers
- Verification – self assessment
- Exit
  - Development plan available

4. Design By Feature

- Entry – “Plan By Feature” process completed
- Tasks
  - Domain walkthrough
  - Examine documentation
  - Create sequence diagrams
  - Extend object model
  - Expand class description with function signatures
- Verification – Design inspection
- Exit – Design packages

5. Build By Feature

- Entry – “Design By Feature” process completed
- Tasks
  - Write classes
  - Code inspection
  - Unit testing
  - Check in completed code to code repository and promote to build
- Verification – Code review and unit testing
- Exit – Completed features

Agile + Discipline

- There are benefits to both the agile approach and the plan-driven approach
- Barry Boehm and Richard Turner discuss the tradeoffs and selecting best from both in their book
  - “Balancing Agility and Discipline”, Addison Wesley, ISBN: 0-321-18612-5, 2004
- Software has become ubiquitous
  - In any field of software, there are many competitors
  - To succeed, need innovative quality products that truly satisfy customers' needs
- Home ground states which works best

Agile vs. Plan-driven

- Think of each of the following as a continuum, with agile favouring the left-most and plan-driven towards right-most

- Personnel
  - Higher to lower ratio of experienced/self-directed staff
- Criticality
  - Loss of comfort – loss of funds - loss of life
- Size
  - Small teams – medium sized teams – large teams
- Dynamism
  - Large – to – small requirements changing every month
- Culture
  - Thriving on chaos vs. thriving on order

Agile vs. Plan-driven

- How to compare approaches?
- Application Characteristics
- Management Characteristics
- Technical Characteristics
- Personnel Characteristics

CRACK Customers

- Customers play a vital role in all software development projects, especially agile
- Customers must be:
  - Collaborative
  - Representative
  - Authorised
  - Committed
  - Knowledgeable

Quality Assurance

- In SW-CMM, QA means adhering to specifications and process compliance
  - Compliance leads indirectly to customer satisfaction
- In Agile, QA means everything needed to ensure true customer satisfaction
  - Very direct focus on this
  - If you are the customer, which would you prefer?

Plan-Driven Approach

- Engineering concept
  - Formal roadwork to how to carry out each part of software development
- Much more substantial process infrastructure
  - Can even have process engineer dedicated just to the project
- Fits well within larger engineering projects
  - Aircraft development project
  - Large-scale telecoms project

Agile Approach

- Craft concept
- Must include all of the following:
  - Iterative and incremental
    (rich feedback possible)
  - Self-organising teams
    (considerable responsibility delegated to team)
  - Emergence
    (processes, principles and practices can evolve/emerge during the project)

Risks

- From page 102 of “Balancing Agility and Discipline”

- Environmental Risks
  - Technology uncertainties, coordination of diverse stakeholders, complex system of systems
- Agile Risks
  - Scalability and criticality, over-simplified design due to YAGNI, personnel churn on team (and tacit knowledge)
- Plan-driven Risks
  - Rapid change, custom demands for rapid results, emerging requirements, learn curve for process itself

Agility & Discipline

- Conclusions from “Balancing Agility and Discipline”
- “Neither agile nor plan-driven methods provide a silver bullet
- Agile and plan-driven methods have home grounds where one clearly dominates the other
- Future trends are towards application developments that need both agility and discipline
- Some balanced methods are emerging
- It is better to build up your method than to tailor it down
- Methods are important, but potential silver bullets are more likely to be found in areas dealing with people, values, communication and expectations management”