

# Planning, Estimating and Correction in an Agile Project

ACCU 2005

Jutta Eckstein

## Agile Value System

- **Agile development is defined by the value system:**

Individuals and interactions  
over processes and tools

Working software  
over comprehensive documentation

Customer collaboration  
over contract negotiation

Responding to change  
over following a plan

### Agile Principles

- **Value system is based on the following principles:**
  - Early and continuous delivery of valuable software
  - Welcome changing requirements
  - Deliver working software frequently
  - Business people and developers work together
  - Trust motivated individuals
  - Face-to-face conversation
  - Working software is the primary measure of progress
  - Promote sustainable development
  - Technical excellence and good design
  - Simplicity is essential
  - Self-organizing teams
  - Team reflection and adjustment

### What can be measured?

- **Working software is the primary measure of progress**
  - tangible
  - reliable
  - executable
  - testable
- **Therefore:**
  - Ensure an always running system

Working Software

- **Motivation**
  - Running system vs. plan fulfillment
- **Marketing**
- **Test**
  - Verification of the concepts
  - Early establishment of acceptance tests
- **User feedback**
- **Culture of achievement**
- **Even:**
  - If deadline passes there is at least usable software

Where to Start

- **Development cycles**
- **Planning**
- **Estimation**
- **Measurement**
- **Reflection**

### Development Cycles

- **Duration**
  - Are they short enough to make small corrections?
  - Are they long enough to accomplish something measurable?
  - Are these *real* time-boxes?
- **Examination**
  - Functionality
  - Estimates

### Timeboxes

- **How often can you measure?**
- **The larger the team,  
the shorter the development cycles**
- **Promoted timeboxes:**
  - 1 week iterations
  - 1 month releases
  - 3 month external releases

How can this be planned?

- **Expectation:**
  - Plan – develop – deliver
- **Difficult:**
  - Activity-oriented planning
    - Not really measurable
      - E.g. what determines the end of design?
    - Neither executable nor testable
  - Component-oriented planning
    - Based on dependencies of the components
    - Fragile to change
    - Difficult to test regarding acceptance
- **Therefore:**
  - Result-oriented planning

Result-oriented Planning

- **Plan for accomplishing a business feature**
  - Accomplishment means integration and test
- **Detailed planning only for the next steps**
  - Estimating and planning are continuous activities
- **Sustainable development**
  - Amount of work must match amount of time
    - consider vacation and real time

### Planning Magnitudes

- **Four variables:**
  - time
  - scope
  - ressource
  - quality
- **If the requirements are able to change:**
  - You should control 2 variables in a project, but never more than 3.

### Fixed-Price Projects

- **Baseline**
  - Time, scope and ressources are fixed
- **Result**
  - Quality suffers
    - It's the least visible magnitude
  - Delivered scope is the requested scope
- **Better solution:**
  - Optional scope contract
    - Scope can change in content but not in size

### Release Planning

- **Releases are defined in terms of Use Cases**
- **Coarse grained release planning**
  - Steered by customer
    - Highest business value first
  - Marketing
    - Considers perception from outside
- **Shows big picture of the project**

### Iteration Planning

- **Iterations are defined in terms of Features (Scenarios)**
- **Fine grained iteration planning**
  - Responsibility of developers
    - With business value and highest risks in mind:
      - Important before optional features
  - Generally:
    - Plan gets more accurate the closer you're to the time you're planning

### Planning Tools

- **Conservative tools**
  - Flip charts
  - Index cards
  - Excel
- **Plans should be located at prominent place**
  - Easy to access
  - Make progress visible
- **Daily synchronization**

### Estimating the first Iteration

#### Problem:

- **Project velocity is unknown**
  - How much time do we have available?
    - Example:
      - Length of iteration: 1 week, 3 developers = 15 ideal days (ideal time)
    - Assumption:
      - Slowing down by 2,5 = 15 days / 2,5 = 6 real days (real time)
  - Therefore:
    - Team accepts tasks for next iteration  $\leq$  real time in total

### Estimating further Iterations

- **Determine the project velocity**
  - Measure completed tasks
- **Estimate by comparison – Yesterday's Weather**
  - Assumption:
    - Team will finish as many tasks in the next iteration as in the last one
  - Consequence:
    - Team accepts for the next iteration as many tasks as they *completed* in ideal time in the last iteration

### Hints for Estimation

- **Early estimation are typically optimistic**
  - Depends on culture
- **Don't be too precise**
- **Don't beautify the numbers**
- **Don't become a slave of the numbers**
- **Don't expect constants**
- **Examine reasons for heavy oscillation**

### Iterations – Measurement

- **Measure the result**
  - How does the code and the test base grow?
- **Feedback**
  - Does the software integrate?
    - Active marketing of concepts and interfaces
  - Do the tests succeed?
  - How does the code quality look like?
    - Regular reviews and code inspections
      - Internal and external

### Reflection on Iteration

- **Heartbeat retrospective**
- **Make achievements visible**
  - Everybody sees the big picture
  - Pride in work
- **Reflect on the achievements**
  - Is the planned functionality realized?
    - What do we learn for the next iteration?
  - Analysis of estimates
    - Improving effectiveness

Releases – Measurement

- **Measure the result**
  - Do the acceptance tests succeed?
    - Manual
    - Automatical (-> regression)
      - Watch out: Capture/replay-tools don't help
  - Is the planned functionality realized?

Reflection on Release

**Work chunk retrospective**

- **Obtain feedback**
  - Present the software to the customer
    - Better: Ask the customer to present the software
- **Reflect on the achievements**
  - Is the customer satisfied?
    - What do we learn for the next release?
    - How does this affect the plan?

### Changing Requirements

- **Complex requirement management processes**

*„Most management strategies are geared either to reduce the number of changes or to control changes. While these strategies are useful, it's more useful to embrace change than to control it.“*

Jim Highsmith

### Agility means Welcoming Change

#### Changing Requirements

- Clear sign that the customer gets a better understanding of the system.
- We have to help the customer to gain this better understanding

#### But:

- **A change means:**
  - Time delay
  - Elimination of another feature

Adjustment

**Plan driven versus planning driven**

- **There is no such thing as a *final* plan at the beginning of the project**
- **Feedback has a direct impact on the plan**

*„A plan is nothing;  
planning is everything.“*

**Dwight D. Eisenhower**

Common Issues

- **Timeboxes are not taken serious**
  - Insist on deadline
- **Achievements are not taken serious**
  - Insist on presentation
- **Estimates are far off**
  - First iterations are an investment
- **„Not doable in defined timeframe“**
  - Learn to partition features into tasks
- **Frequent disruption prevents progress**
  - Quiet Times

Course Corrections

- **Working software and retrospectives**
  - Continuous improvement of the plan
- **Development**
  - Working software provides direct and concrete feedback
    - Integration, quality, tests, customer acceptance, architecture, ...
    - Continuous small course corrections
- **Process**
  - Regular retrospectives
    - Permanent reflections and optimization of the process
    - Quality improvement of estimates and plan

Beware:

***„If a project is on time and in budget  
that doesn't mean it was a successful project,  
but a successful estimate.“***

Martin Fowler

Many Thanks!

**If you want to know more:**

Jutta Eckstein  
jutta@jeckstein.com  
www.jeckstein.com

