# L5 - Project management during inception; Agile & XP approaches

CS3028 - Principles of Software Engineering

**Ernesto Compatangelo**

Department of Computing Science

UNIVERSITY OF ABERDEEN



## 7.1 Reminding past issues and mapping them to current topics



### Where are we now?

Software development paradigms

⇒ The Unified Process (UP) paradigm

⇒ UP phases and UP disciplines (activities) within each phase

⇒ Inception (first UP phase)

⇒ Activities and deliverables during inception

⇒ . . . . . .

⇒ Software project management during inception

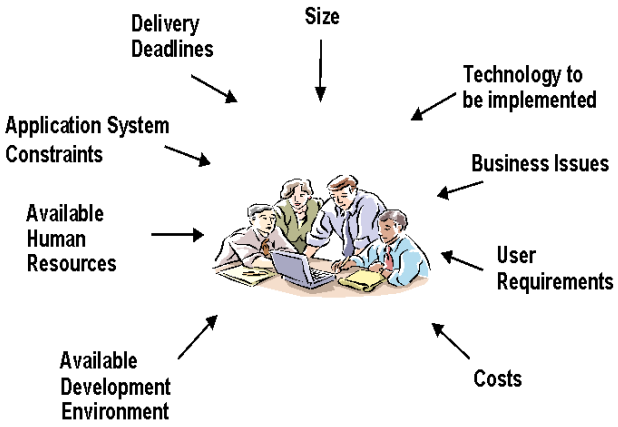
⇒ . . . . . .

# 8 Inception project management

## 8.1 Addressing Software project management issues



### Software project management concerns




Three things influence SW projects and thus software development:

PEOPLE – PROCESS – PROBLEM (PRODUCT)

In this specific order of importance!

## Software project management problems

- Poor estimates and plans;
- Lack of quality standards and measures;
- Lack of guidance about making organisational decisions;
- Lack of techniques to make progress visible;
- Poor role definition — who does what?
- Incorrect success criteria.



### 8.2 Setting up a software project management strategy

Involves the following activities:

- **Estimating**: Assessing the time and money cost of each activity)
- **Planning**: Deciding what is to be done (and when)
- **Organising**: Making arrangements
- **Staffing**: Selecting the right people for the job
- **Directing**: Giving instructions
- **Monitoring**: Checking on progress
- **Controlling**: Taking action to remedy problems
- **Innovating**: Coming up with new solutions
- **Representing**: Liaising with clients, users, etc.

#### 8.2.1 A few issues

- The larger and the more complex the project – the more difficult the communication.
  - **Client**: May have general idea of what (s)he wants, but may not understand how software works or is developed.

  - **Developer**: Knows a lot about software & development, but typically little or nothing about the domain.
  - **Both sides**: Have different interpretations of domain and technical terminology.

- **Combinatorial** explosion of communication paths:
  ... in formulae: n(n-1)/2, where n is No of team members

- **Verbal communication often contains gaps**: Written communication is needed to ensure accuracy.

- **Team members often change**: People come and go, but project documentation remains!

## 8.3 A disciplined approach to software project management

### 8.3.1 Team management strategies

- Establishing project objectives and scope — includes the identification of technical constraints.
- Deciding on measures and metrics — means adopting suitable metrics to measure *process* (software engineering) and *product* (software).
- Planning and estimating — means (i) decomposing the overall software development tasks into a series of homogeneous subtasks and (ii) deriving estimates of required human effort, overall project duration and detailed cost for each sub-task and for the overall task.
- Managing risks — means addressing typical areas of uncertainty such as: (i) the real understanding of client's needs; (ii) the achievement of each functionality within the project duration; (iii) the timely explicitation of hidden technical problems.
- Deriving a project schedule — means: (i) identifying a set of project activities (tasks) and interdependencies amongst these activities; (ii) estimating the effort within each activity; (iii) assigning people and other resources; (iv) developing a project time-line.
- Monitoring and controlling — means keeping activities under control so that impact of slippage can be assessed. Resources may need to be redirected, activities re-ordered and so on.

- **Team composition**: Teams need different types of people to succeed.
- **Project managers need**: (i) Problem-solving ability; (ii) Managerial skills; (iii) Communication and people skills; (iv) Ability to reward achievement; (v) ability to control risks.
- **Different projects need different leadership styles**: (i) people-oriented vs. task-oriented; (ii) autocratic vs. democratic; (iii) directive vs. permissive.

### 8.3.2 Belbin's team roles

- **Chair**: Not necessarily brilliant leaders, but good at running meetings; calm, strong but tolerant.
- **Plant**: Good at generating ideas and potential solutions.
- **Monitor-Evaluator**: Good at evaluating ideas and solutions, and helping to select best one.
- **Shaper**: A worrier; helps direct team to important issues.
- **Team Worker**: Skilled at creating a good working environment.
- **Resource Investigator**: Adept at finding the needed resources.
- **Completer-Finisher**: Concerned with completing tasks.
- **Company Worker**: Good team player, willing to undertake non-attractive tasks if needed.

### 8.3.3 Typical project team organisations

- **Hierarchical Team**: Formal organisational structure, with top-down management. Controlled, centralised approach.

  *(For large projects and large organisations only)*

- **Decentralised Team**: No permanent leader. Task coordinators appointed for short durations, and then replaced by others. Decisions made via group consensus. Communication is horizontal.

  *(For experienced, stable teams only)*

- **Controlled Decentralised Team**: Leader coordinates certain tasks. Secondary leaders responsible for sub-tasks. Problem-solving a group activity, but solution implementation partitioned amongst sub-groups by leader. Communication is horizontal amongst sub-groups and individuals, but vertical along the hierarchy.

  *(For large projects and large organisations only)*

- **Chief Programmer Team**: Limit number of people actually producing the software, but give them lots of support.

  - *Chief Programmer* — Defines specification and designs, codes, tests and documents the SW.
  - *Copilot* — discusses issues with CP; writes some code.
  - *Editor* — Writes documentation drafted by CP.
  - *Clerk* — Maintains code.
  - *Tester* — Performs SW testing, verification and validation.

  *(For medium projects and small organisations only)*

  **Not to be used in the CS3015 project!!!**

- **Egalitarian team**: programmers read other people's programs; programs become common property of the team; peer code reviews.

  *(For any kind of project and small-to-medium organisations only.very dynamic but very rewarding)*

  **The organisation model you MUST use in the CS3015 project!!!**

### 8.3.4 Management 'Myths'

- *We have lots of standards and procedures for software.*
  Are they used? Are they complete?
  Do they reflect modern practices?

- *We have state-of-the-art computers.*
  It takes more than the latest model PC to achieve high quality software development; CASE tools (e.g. Together) are more important than hardware (e.g. computing speed). Are these tools used effectively?

- *If we fall behind schedule we can add more programmers.*
  Adding people to a software project makes it later.

## 8.4 The evaluation of software projects



### Assessing software and its development

Both clients and developers need to estimate/assess:

- The **quality** of the software product.
- The **productivity** of the people who build the software.
- The **benefits** of new/existing SE methods and tools.
- The **needs** for new tools/training.
- The **risks** associated to a development project. This means devising a **contingency plan** if things do not go according to the original plan.



### 8.4.1 Software measurements in general

- **Direct Measurements (hard data):**
  - Number of people assigned to project;
  - Effort (person-months) expended on project tasks;
  - Duration (months) for each project task;
  - Documentation length, source code (LOC), test cases;
  - Number of defects found and reported.
- **Indirect Measurements (soft data):**
  - Business constraints on project;
  - Skill of project team & experience of project manager;
  - Stability of requirements as a function of time;
  - Difficulty of problem to be solved;
  - Software quality, efficiency, reliability, maintainability;
  - User satisfaction with end product.

### 8.4.2 Software sizing

Accuracy of project estimate is influenced by many factors. An accurate estimate of the software 'size' is crucial.

- **Function point sizing** estimates the characteristics of the information domain.
- **Standard component sizing** estimates the number of occurrences of each standard software component (reports, screens, modules, etc.) Historical data are used to compute the size of each standard component.
- **Change sizing** estimates the number and the type of changes to be made (e.g. adding, deleting, modifying, and re-using code).

**Size-oriented metrics**

- Normalise quality and/or productivity measures by considering the size of the software artefact.
  (K)LOC = Length Of Code (KByte) is one measure of size.
  - Productivity = KLOC/effort in man-month
  - Quality = Defects/KLOC
  - Cost = £/KLOC
- **Pros:** LOC can be easily counted (artefact of development)
- **Cons:** LOC measures (i) are programming language dependent and (ii) penalise well-designed (smaller) programs.

**Function-oriented metrics** Do not count LOC, but focus on program 'functionality'.

**Function points** are derived using an empirical relationship based on countable (direct) measures of the software's information domain and assessments of software complexity.

In formulae, $F_p = (\Sigma_k N_k) \times (\alpha + \beta \times \Sigma_i F_i)$ where

- $N_k$ = Information domain measure = one of (No of user inputs; user outputs; user enquiries; files; external interfaces)
- $F_i$ = complexity adjustment value associated with each $N_k$;
  (Does the system require reliable backup and recovery? Are data communications required? Are there distributed processing functions? Is performance critical? and so on. . . )
  $0 \ \ (irrelevant) \leq F_i \leq 1 \ \ (essential)$,
- $0 \leq \alpha \leq 1; \ \ 0 \leq \beta \leq 1; \ \ \alpha \sim \beta \times \Sigma_i F_i$

### 8.4.3 Estimating software development cost and effort

**Measurements** for software (product & process) are needed. These are denoted as **software metrics**.

A key element of project planning is the generation of reasonable estimates of resources, cost and schedule.

- Before estimation begins, identify and assess:
  - *Software scope*, i.e. data and control, function, performance, constraints, interfaces and reliability.
  - Feasibility, i.e. whether a project is feasible and the resulting software can meet its scope
- Approaches to cost & effort estimation:
  - Estimates based on similar completed projects.
  - Simple decomposition techniques (divide and conquer).
  - Empirical models that use historical data.
  - Automated estimation tools.

**The COnstructive COst MOdel (COCOMO)**

- **COnstructive COst MOdel** I & II (Boehm, 1981 & 1996)
  - Hierarchy of software estimation models.
  - Use empirical formulas to predict data for planning.
  - Estimates based on LOC approach
- **Basic COCOMO**: Computes development effort and cost as function of program size. Gross estimator.
- **Intermediate COCOMO**: Considers 15 cost driver attributes. Assesses product, hardware, personnel, etc.
- **Advanced COCOMO**: Phase- sensitive effort multipliers enable effort to be modified and updated as project continues.

**The basic COCOMO**

- Effort (person-month) $= \alpha \times KLOC^{\beta}$
- Development time (months) $= \gamma \times EFFORT^{\delta}$
- **Organic SW**: Small, simple project; small team with good application experience; less than rigid requirements.

  $\alpha = 2.4$ $\beta = 2.05$ $\gamma = 2.5$ $\delta = 0.38$
- **Semi-detached SW**: Intermediate project; team with mixed experience level; mix of rigid/less than rigid requirements.

  $\alpha = 3.0$ $\beta = 1.12$ $\gamma = 2.5$ $\delta = 0.358$
- **Embedded SW**: Tight hardware, software and operational constraints. $\alpha = 3.6\beta = 1.20\gamma = 2.5\delta = 0.32$

**COCOMO examples**

- Semi-detached project with an estimated size = 33.3 KLOC
  - Effort (E) $= 3.0 \times KLOC^{1.12} =$ **152 person-month**
  - Development Time (DT) $= 2.5 \times E^{0.35} =$ **14.5 months**
  - Recommended Staff (RS) $= E/DT = 152/14.5 =$ **11 persons**
- Organic project with an estimated size = 3.3 KLOC
  - E $= 2.4 \times KLOC^{1.05} =$ **8.4 person-month**
  - DT $= 2.5 \times E^{0.38} =$ **5.6 months**
  - RS $= E/DT = 8.4/5.6 =$ **1.5 persons**
  - On a part-time basis (1 day per week = 20% of full time)
    Development time of 5.6 calendar months requires $1.5 \times 5 = 7.5$ persons (i.e. approx. 8 people for 5 months)

## 8.5 Risk analysis



### Risk analysis — a step-by-step approach

Risk analysis and management help a team to understand and handle uncertainty. It encompasses the following steps:

- **Identification**: Define the likely risks for the project.
- **Projection (or estimation)**: Indicate the quantitative likelihood (e.g. percentage) that each risk will occur.
- **Assessment**: Evaluate accuracy of projections and prioritise risks.
- **Management & Monitoring**: Move to avert risks that are of concern and monitor circumstances that may lead to risk.



## 8.6 Common risks

- **Team**:
  - Inexperienced leader and/or project team;
  - Team working part-time and/or High staff turnover.
  - To know more on team building, see BuildYourOwnTeam in CS3028 MyAberdeen
- **Client/User**:
  - Low confidence in developer and/or hostile/unsure users;
  - Users do not spend time to specify & define.
  - Very tight deadlines.
- **Application**:
  - Complex problem and/or environment and/or data;
  - New business area and/or technology;
  - Limited development environment;
  - High dependence on other systems.

## Risk assessment

- Examine accuracy of time/cost estimates made.
- Prioritise risks.
- Think about ways to avert/control risks.
- Determine whether risks might lead to (early) project termination.



### 8.6.1 Some reasons behind late software delivery

Projects fall behind schedule because of

- Unrealistic deadlines imposed by management
- 'Ever changing' user requirements
- An honest underestimate of the development effort
- Unforeseen risks; technical and human difficulties
- 'Miscommunication' among project members
- **Failure to recognise that a project is falling behind schedule**

### 8.6.2 The role of software project scheduling

- **Modularisation:** split the project into a number of manageable activities and tasks.
- **Interdependency:** establish time constraints between activities and task: parallelism, sequentiality.
- **Time allocation:** define start and end dates, full- or part-time activity, person-days.
- **Effort validation:** check that there are enough people to do all the activities scheduled at any particular time.
- **Defined responsibilities:** assign each scheduled task to a specific team member.
- **Defined outcomes:** every activity or task must produce one or more well-specified deliverables.
- **Defined milestones:** every task, activity, or group of them should be associated to a milestone reached once the corresponding deliverables **are approved**.

### 8.6.3 Identifying a project task set

'A collection of SE tasks, milestones and deliverables that must be accomplished to complete a project'.

- Some tasks are common to any kind of SW development project, *e.g.*, elicit, model, and analyse requirements, design system, implement, test, and deploy application.
- Other tasks are project-specific, *e.g.*, technology risk assessment, prototyping, customer reactions.
- Different kinds of projects may involve a task at different levels of width (extent) and depth (rigour).
- While 'exploratory projects' are characterised by an 'in-width' approach with 'casual' (*i.e.*, minimal) rigour, 'reengineering projects' are characterised by an 'in-depth' approach with 'strict' (*i.e.*, maximal) rigour and lack of any exploratory task.

Different task sets are defined for the following different kinds of projects:

- **Concept development projects** explore new business concepts or the application of new technology.
- **New application development projects** are undertaken following a specific customer request.
- **Application enhancement projects** occur when existing SW undergoes major modifications to functions, performance, or interface that are observable by the end user.
- **Application maintenance projects** correct, adapt or extend existing SW in a way that may not be immediately obvious to the end user.
- **Re-engineering projects** rebuild an existing (legacy) system in whole or in part.

**Scheduling project task sets**

- A set of tasks is defined for a project which defines its Work Breakdown Structure (WBS)
- A timeline chart (also called 'Gantt chart' is created to show the evolution of a task-based software development project as a function of time. The Gantt chart (see below) shows start and end dates, milestones, and deliverables associated to each task.

| ID | WP No | Work Package Name | Begins | Ends |
|---|---|---|---|---|
| 1 | **WP1** | **Modelling** | **01/09/2005** | **31/05/2007** |
| 2 | WP1.1 | Repr. of editing changes | 01/09/2005 | 31/03/2006 |
| 3 | WP1.2 | Ontol. of editing changes | 01/11/2005 | 31/03/2006 |
| 4 | WP1.3 | Repr. of semantic operations | 03/04/2006 | 31/10/2006 |
| 5 | WP1.4 | Ontol. of semantic operations | 01/06/2006 | 31/10/2006 |
| 6 | WP1.5 | Repr. of evolutionary mappings | 01/11/2006 | 31/05/2007 |
| 7 | WP1.6 | Ontol. of evolutionary mappings | 01/01/2007 | 31/05/2007 |
| 8 | **WP2** | **Analysis** | **01/09/2005** | **28/02/2008** |
| 9 | WP2.1 | Reas. about change recording | 01/09/2005 | 31/05/2006 |
| 10 | WP2.2 | Reas. about change modelling | 01/06/2006 | 28/02/2007 |
| 11 | WP2.3 | Reas. about evolution mapping | 01/03/2007 | 30/11/2007 |
| 12 | WP2.4 | Reas. about change assessment | 01/06/2007 | 28/02/2008 |
| 13 | **WP3** | **Infrastructure** | **01/09/2005** | **29/08/2008** |
| 14 | WP3.1 | Versioning knowledge environment | 01/09/2005 | 31/05/2006 |
| 15 | WP3.2 | Semantic web architecture | 03/04/2006 | 29/12/2006 |
| 16 | WP3.3 | Platform & evolution functionalities | 01/11/2006 | 29/08/2008 |
| 17 | **WP4** | **Evaluation** | **01/09/2005** | **29/08/2008** |
| 18 | WP4.1 | Analysis of usability questionnaires | 01/09/2005 | 29/08/2008 |
| 19 | WP4.2 | Analysis of recorded information | 01/06/2006 | 29/08/2008 |

# 9 Agile and XP approaches to software development and project management

## 9.1 Criticising the Waterfall and the UP paradigms

- The Waterfall is unrealistic and rigid; it is not the natural way software is developed in practice. At no time before delivery can requirements, design, or code be considered as 'finalised'
- the UP needs a lot of project management infrastructure to keep everything under control and may result in the production of too many intermediate paper artifacts (documents) to be continuously updated

Project management during inception

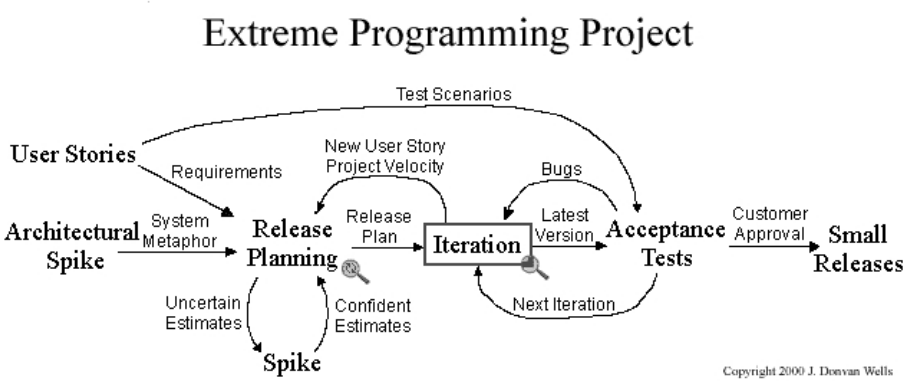
### Lessons learnt in almost 40 years of software engineering

- **Too much analysis and design beforehand is useless** - the implemented system will widely differ from the initial 'conceptualisation'
- Software robustness, reliability, quality, minimality, simplicity, etc. can only be achieved through **code review and refactoring**
- UML (and diagrams in general) are mostly **useful in reverse engineering rather than in direct engineering**
- Involving only one programmer in any single development task is, in the long term, more expensive than **involving a team of two programmers** jointly working on the task



## 9.2 The Agile/XP approach

**A diagrammatic view of the Agile/XP approach**



## The Agile & XP approach in a nutshell

- XP programmers **communicate** with their customers and fellow programmers
- XP programmers keep their design **simple** and **clean**
- XP programmers get feedback by **testing their software starting on day one**
- XP programmers **deliver** the system to the customers **as early as possible** and **implement changes as suggested**

*Risky projects with dynamic requirements are perfect for XP*





## Does an Agile/XP approach contradict the UP?

**NO**, because

- The UP does not advocate any specific discipline structure or phase length and weight
- The UP does not impose lengthy UML developments (UML as a detailed specification) upfront
- The Agile approach does not advocate scrapping requirements and design; it just de-emphasises them reducing their upfront size

## 9.3 Agile requirements specification: user stories



### User stories: how the Agile approach captures requirements

A user story describes functionality that will be valuable to a software system user or purchaser. User stories focus on three aspects:

- A written description of the functionality used for planning and as a reminder
- Conversations and all sorts of informal materials about the story that serve to flesh out the details of the story
- Tests that convey and document details and that can be used to determine when a story is complete

**Example of description:**

A user can post Form P60 to the Inland Revenue Website

To know more: Mike Cohn. User Stories applied. Addison Wesley, 2004 (**it's on Safari!**)

## 9.4 Agile estimating and planning



### Simple and effective estimating: the Agile approach

After estimating the cost of infrastructure/technology (the easy bit) the development costs must be assessed (size, time, and thus expenditure). This is the challenging bit. Estimating focuses on the following sequence

- **Estimate software size**
- **From size, estimate YOUR velocity**
- **From velocity, estimate YOUR development time**
- **From time, estimate YOUR development costs**





### How do we estimate size Agile-wise?

- Look at all stories
- Assign story points to each story; the longer/more complex a story is w.r.t. the others, the higher the more the points
- Story points are relative measure; once you have decided what is the minimum '1 point' all the rest becomes a multiple
- Revise your estimate ruthlessly until when you are confident it is a realistic measure

To know more: Mike Cohn. Agile estimated and Planning. Prentice Hall, 2005 (**it's on Safari!**)

## How do we derive our velocity Agile-wise?

- Define the length of your iteration (one or two weeks), in which you perform one development minicycle from beginning to end
- Assess how many stories you can complete in that given time frame
- Add the points from all stories you estimate to complete in the given time and divide the result by the number of week (one or two); this is your weekly velocity
- Once you have your velocity, calculating the time you need and thus the development costs you incur is easy
- Revise your estimate ruthlessly until when you are confident it is a realistic measure

To know more: Mike Cohn. Agile estimated and Planning. Prentice Hall, 2005 (**it's on Safari!**)



### 9.5 Preparing for the topic ahead



## Next lecture...

**Moving towards elaboration**

More specifically, we will focus on:

- Inception to elaboration: change of perspectives