How to improve the quality of your processes

Nigel Goddard

School of Informatics
University of Edinburgh

Prior question

Q: Why improve a process (let alone all your processes?)

To manage (reduce, predict and plan for) risk that bad things will happen to you.

Risk management

- Analyse risks during the (pre-)planning stage of a project, so you can
- Manage them when they happen during the project

Roughly, risks relate to the project, the product, or the business.

Examples of typical risks: staff loss, management change, missing equipment, changing requirements, delays in requirements analysis, mis-estimation of cost, competitor gets to market first.

Planning for risks

Identify the risks early, in various possible categories.

Analyse each identified risk. Is it minor, major, serious, fatal? What is the chance of it happening?

Plan how to cope with each risk. Can it be avoided by reducing the probability of occurrence? Can you plan to minimise the effect if it does happen? What is your contingency plan if it does happen?

Exercise: Consider how you might plan mitigation for:

- The lead designer on the human interface team leaves.
- (Bespoke software) The customer changes the requirements to adapt to a new company acquisition.
- (Shrinkwrap software) A competitor releases a product that already has a key feature of the product you're developing.

Consider also how you might plan to reduce the risk of these things happening in future projects.

Software quality

Q: What is it?

Ultimately, anything that the customer cares about.

Approaches to improving quality

May focus on

1. the software product itself
   Verification, validation, testing, code/design reviews, inspections, walkthroughs are product-focused approaches to quality improvement.
2. the process by which the software is produced

In this lecture we consider process-focused approaches.

Process focus

Advantages:

- potential to improve all products
- possibility of certifying a whole organisation
- some important things, especially planning (and thus time-to-market, cost etc.), are hard to approach in any other way.

Disadvantages:

- done badly, can easily prove very costly with low benefits: easy to spend time and money without improving any product

Centres of process-focused QA

Possibly directions of influence, each with own philosophy:

Organisation $\longleftrightarrow$ Project $\longleftrightarrow$ Individual

1. Organisation's management decrees, influencing projects whose managers direct individuals into desired behaviour.
2. Individuals introduce improvements which are rolled up and out to the rest of the project and the rest of the organisation.

Making these centres work together productively depends on the software engineering culture of the organisation and ethics of the individual.

"Quality is free, but only to those who are willing to pay heavily for it." (Philip Crosby/Tom DeMarco)

Areas and terminology

- Quality planning – how will you ensure that this project delivers a high quality product?
- Quality metrics – what measurements must you make in order to tell whether what you're doing is making the difference you intend?
- Quality improvement – what can you learn from this project to help you plan and run the next one better?
- Quality control – how can you ensure and prove that your quality plan was followed?
- Quality assurance – an umbrella term for the whole field.

Standard QA models

We'll look briefly at two examples with different aims:

- CMMI, the Capability Maturity Model Integration (from Carnegie Mellon's Software Engineering Institute), a development of the very popular Capability Maturity Model (CMM).
  Crucial idea here is that maturity increases: quality planning, control and improvement framework.
- ISO9000 - family of quality standards
  Less emphasis on improvement: quality control framework.

These can be complementary (resolution of a historical argument about which should be used).

Standards

There are many types of standard. Coding standards tell you how to format code. Documentation standards tell you how to organize documentation. Quality standards tell you how to . . . organize quality control.

ISO 9001 is an international standard for quality assurance. It specifies how to specify documents and procedures that a company should follow in its quality control. It does not specify or require any level of product quality.

CMMI

The Integrated Capability Maturity Model is a successor to the influential CMM. It provides a (generic, specializable to software) description of process areas, goals associated with each area, and practices that may achieve goals. Organizations are assessed at a maturity level according to how they achieve goals and follow practices. Levels are:

1. Initial: goals may be met, but by heroics
2. Managed: documented plans, resource management, monitoring, etc.
3. Defined: organization defines process framework. Measurements collected for use in improvement.
4. Quantitatively managed: use measurement and statistical etc. methods during process.
5. Optimizing: process improvement happens, driven by quantitative measures.

CMMI vs. ISO 9001: Case Study

Boeing Case Study, Dale Spaulding, 2005

Total Quality Management

Plan, Do, Check, Act – the Deming cycle.

TQM embodies the idea that improving quality is everyone's job – not just that of the QA department. Typical principles – these quoted from `http://www.siliconfareast.com/tqm.htm`:

1. Quality can and must be managed.
2. Everyone has a customer to delight.
3. Processes, not the people, are the problem.
4. Every employee is responsible for quality.
5. Problems must be prevented, not just fixed.
6. Quality must be measured so it can be controlled.
7. Quality improvements must be continuous.
8. Quality goals must be based on customer requirements.

Then there's Six Sigma, "TQM on steroids".

Bottom line

Things only get better when those involved

- have enough information to tell what's wrong
- think intelligently about it
- plan how to improve
- actually make sure the plan happens
- check whether it worked.

Many ways to achieve this, many ways to fail. Not specific to software.

Don't be seduced by Methodology

There is a big difference between *methodology* and *Methodology*. A *methodology* is a basic approach one takes to get the job done, consisting of:

- a tailored plan (specific to the work at hand)
- a body of skills necessary to effect the plan

A *Methodology* is an attempt to centralize thinking. All meaningful decisions are taken by the *Methodology* builders, not by the staff assigned to do the work. The overt case for the *Methodology* includes standardization, documentary uniformity, managerial control and state-of-the-art techniques. The covert case is simpler and cruder: the idea that project people aren't smart enough to do the thinking.

(adapted from Peopleware, Tom DeMarco & Timothy Lister, p115)

Management

Two types of management are relevant to software:

- people management
- project management

Both types are best seen as enabling activities: a good manager doesn't do the actual work of building software but aims to maintain an environment in which it's possible for people to get on and do so!

Of course many would argue that the bottom line is always financial.

A manager is not the same as a leader...

Software project management

A project manager arranges for the following functions to be fulfilled:

- Planning
- Organizing
- Staffing
- Monitoring
- Controlling
- Innovating
- Representing

Making stuff happen, getting obstacles out of the way.

Cost estimation

Before starting, or bidding for, any significant project, need to ~~know~~ estimate how much it will cost.

Many things are factors in this estimation: but the main factors are software size and complexity, and engineer productivity.

We will here consider only software size and complexity. (Productivity is ratio of this to time required.)

LoC

A simple measure of software size is lines of code (LoC).

Not very meaningful by itself. What is a line of code?

How many lines of Haskell correspond to how many lines of Java correspond to how many lines of C? What about library routines?

Still widely used; alternatives like "function points" exist.

Estimation

Metrics are all very well, but how do you ~~guess~~ estimate them for software that doesn't exist yet? Some approaches:

- algorithmic cost modelling: develop (from past data) a model relating size/complexity to ultimate cost
- expert consensus: get a bunch of expert estimates. Compare, discuss, repeat until convergence.
- analogy: relate the cost to that of similar completed projects
- available effort: that way madness lies...
- what the customer will pay: dangerous...

COCOMO

COCOMO is a long-standing algorithmic model, publicly available, well supported, and widely used.

Basic idea:

$$\text{Effort} = A \times \text{Size}^B \times M$$

for suitable A, B, M that (between them) vary depending on the product domain, the organisation, its process, etc.

Getting good values for these is highly non-trivial. B is typically between 1 and 1.5: a system twice the size takes more than twice the effort to develop.

Considerable data available. Versions/submodels/tweaks available to account for factors like reuse, generated code, etc. See Wikipedia page.

Why do projects almost always slip...

... relative to human intuitions of how long they should take?

(This is why we need something like COCOMO.)

Discussed in paper The Rational Planning of (Software) Projects, Mark C. Paulk

This paper discusses the effects of three features of human nature:

- "people tend to be risk-averse when there is a potential of loss"
- "people are unduly optimistic in their plans and forecasts"
- "people prefer to use intuitive judgement rather than (quantitative) models"

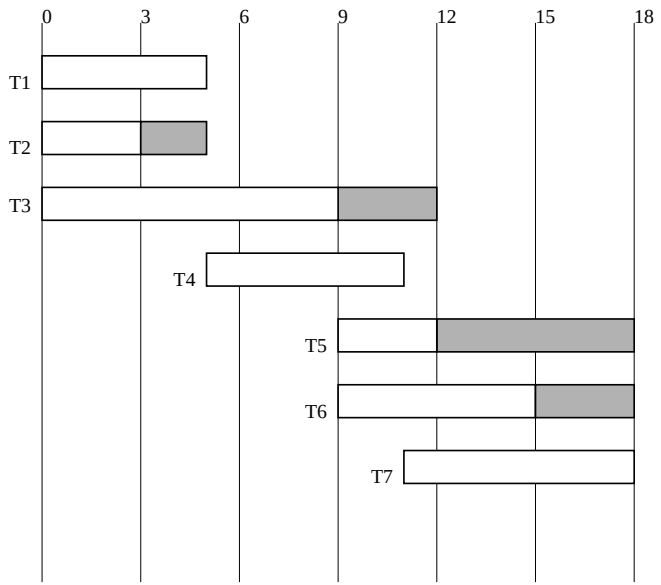
It goes on to discuss how a framework like the CMM can help.

Gantt charts

An example of a project planning tool, to help with scheduling.

Divide project into tasks, with milestones at the end. Analyse (e.g. in graphical network) dependencies between tasks. Now lay these out as bars running across time, respecting dependencies. This reveals the critical path of tasks for the project. Optionally, show permissible slippage with shaded bars.

- Task 1 takes 5 weeks.
- Task 2 takes 3 weeks.
- Task 3 takes 9 weeks.
- Task 4 takes 6 weeks, and depends on tasks 1 and 2.
- Task 5 takes 3 weeks, and depends on task 3.
- Task 6 takes 6 weeks, and depends on tasks 1 and 3.
- Task 7 takes 7 weeks, and depends on task 4.



Project tracking

The project manager needs to decide how and what to track. E.g.:

- how much time each person spent on each task, and when?

- just total effort expended?

- something in between?

Aim is to find a happy medium between having too little information to tell whether things are OK, and so much that it's very time-consuming to manage the information. Ideally want meaningful info!

Tools are available and helpful, esp. for big projects, but they don't create the data or decide what to do as a result...

Revising the project plan

As the project goes on, estimates have to be revised in the light of progress, unforeseen problems etc. Typically a large project will replan once a week.

Depending on the circumstances slippages may mean

- reallocating resource (but see The Mythical Man Month)
- cutting functionality
- asking the customer for more money
- losing profit

Tell them the most important metric is what proportion of the project staff's time is spent explaining to the customer why the project is late Anonymous software engineer.

Reading

Required: GSWEBOK Ch11

Suggested: The Rational Planning of (Software) Projects, Mark C. Paulk

Suggested: Sommerville Ch26, available on line, see web page.

Suggested: Sommerville §5.4, Ch 27,28 and/or Stevens Ch19,20.

Suggested: Sommerville Chs 4,5,17 and/or Stevens Ch 4.

Quotes of the day

> *Adding manpower to a late software project makes it later.*

("Brooks' Law") Fred Brooks in The Mythical Man-Month (1975), chapter 2

> *The ultimate management sin is wasting people's time.*

Tom DeMarco and Timothy Lister in Peopleware

> *Any process that tries to reduce software development to a "no brainer" will eventually produce just that: a product developed by people without brains.*

Any Hunt and Dave Thomas in Cook until done