

CSC 7003 : Basics of Software Engineering

J Paul Gibson, A207

`paul.gibson@int-edu.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/>

Planning Teams

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7003/L7-Teams.pdf>

Development Teams

A good software process facilitates/encourages working in teams

Good team work depends on clearly defined roles, responsibilities and interactions

A development team should provide the structure upon which good team work can be built

All engineering processes involve teamwork, including software development

The software process has other unique problems associated with it --- flexibility, cost-spectrum, complexity

Software development teams must understand these properties

People are more important than process

On small teams, each individual's ability contributes more to the success of the project than any other factor

On large teams, the law of averages forces us to deal with fewer exceptional people and many more average people (or worse)

On large teams, it is easier for the poor people to hide

A good process must leverage the talents of the average people, and identify those who contribute nothing (or worse)

Good people with a good process will always outperform good people with no process

Just hiring lots of people is not always enough

Just hiring good people is not always enough ... see next slide

KEY: the team should be more than the sum of its parts

Just hiring good people is not always enough

There are 3 main problems with organising a development team around a bunch of smart people:

- there are not enough smart people to go around
- there is a big difference between creating a simple, brilliant program that demonstrates one cool concept very well, and turning that program into a product which meets the needs of industrial strength users
- some problems are so large (not necessarily complex) that they simply require a lot of hard, sustained labour which the smart people may find odious because it is not as much fun, not as challenging or requires engineering compromises

What size should our teams be ??

In general the best size for a team depends on: the problem, the deadlines, the budget, and the existing infrastructure. But in general, we differentiate between:

- **Individual** --- 1 or 2 people
- **Small** --- around 5 people
- **Average** --- 10 to 20 people
- **Large** --- peak staff of around 50
- **Geopolitical** --- 100's of developers

The larger the number of people, the more chance for failure. Smaller teams are easier to work with and more nimble. OO projects tend to require smaller teams but, as complexity grows, one will always need more staff. The software process should support *teams within teams*.

KEY: process is important in building teams because it is repeatable, it helps address issues of scale and it leads to sustainable practice.

Software development is a human endeavour

Even if you organise a team with the right roles and responsibilities, your project will flounder if you fail to recognise that development is indeed a human activity.

A successful development process promotes the following:

- A project must recognise that its developers are not interchangeable parts
- A project must honour and respect the role of every one of its developers
- Individuals must understand the roles of all team members with whom they are in direct contact
- Individuals must be honest about their strengths and weaknesses

KEY: recognise that every complex software project requires a balanced mixture of different skills that no single individual is ever likely to have

People: roles and responsibilities

Team size:

- In small projects, team members have many roles
- In average projects, team members usually have few roles
- In large projects, each role usually requires many members

In traditional development, the **core roles** were based on the utopian *waterfall method* of development ---

requirements, design, implementation, testing, maintenance

In OO, where incremental and iterative development is the guiding light, these conventional roles are blurred.

KEY: the roles should be tailored to the development method

Role Classification

The development team includes any person who has an impact upon the success or failure of a project. Successful teams incorporate layers of responsibility. For example:

- **Core** --- software production
- **Supplemental** --- support for the core workers
- **Peripheral** --- on project boundary

KEY: Remember that all layers are equally important, even though they may not require the same resources, and they should be symbiotic not competitive.

Core Role Organisation

Organise a core development team as follows:

- **an architect** who is responsible for system's overall structure
- **abstractionists** who manage architecture of subsystems
- **application engineers** who implement and assemble code

In very large systems: the system architect may be supported by an architecture team

In very small systems: 1 or 2 people can play all these roles

In general, the more resources spent on architecture and abstraction then the better for the long-term needs of a project and its environment

KEY: when developing software try not to have more than 50% of resources allocated to *coding*.

Architects

An architect should possess the following skills:

- Experience --- expertise in the domain and in software development
- Leadership --- focus, confidence, charisma and authority
- Communication --- speak the language of customers and engineers
- Pro-active & goal oriented --- make decisions, take risks, produce something

It is also useful if they are reasonably good programmers!

Abstractionists (OO)

- Identify classes, class clusters, class hierarchies, interfaces, tests, increments
- Interface between architect and application engineers
- Be ready to become an architect or application engineer, if necessary

Application Engineers

Largely responsible for producing code --- *coders* or *programmers*

Also contribute to design documents and user documentation

Turns abstractions into reality

In small projects they are also abstractionists

In large projects they become specialised ---

- GUIs, Databases, Networks, Security, Devices, Algorithms and maths

KEY: Good application engineers should make good abstractionists and architects, but some projects rely on some of their best people remaining as humble programmers!

The supplementary roles

The supplementary team is there to support the core:

- Project manager
- Analyst
- Integration
- Quality Assurance
- Documentation
- Toolsmith
- System Administrator
- Librarian

The Project Manager

The most important supplementary role:

- Responsible for project's success and active management
- May not 'cut code' but is just as important as the architect
- Drives the rhythm of the project
- The manager must not abdicate responsibility to the core
- The core is responsible for designing software, the project manager is responsible for designing the core!

Main responsibilities:

- deliverables, schedules, staff, resource allocation, budget, communication with patrons and users.

KEY: project managers and architects should be as close as possible

The Analyst

Responsible for evolving and interpreting end user requirements

All core members play role of analyst

In large projects, this should be a distinct role (or roles)

Analysts must be problem domain experts

Main duties:

- user interviews, identifying functional and performance needs, building analysis models, detecting and resolving requirements contradictions, detecting and resolving ambiguities, validation using scenarios, interfacing between clients and engineers.

Integration

Assembles compatible versions of released artifacts (stand alone software) in order to form a deliverable and executable release:

- In traditional approaches integration is monolithic
- With OO it is a continuous process
- It is best not to under-use a good core developer in this role
- It requires clerical skills
- It requires proficiency of all project development tools
- It needs good management skills

KEY: a good integrator must be able to work in very focussed bursts of labour (usually during the night!)

Quality Assurance

One of the most thankless tasks:

- core developers believe they are obstacles to overcome
- managers view them as costly overheads to be humoured

Their main duties involve measurement:

- A software process cannot mature unless it can measure properties of the projects which are run using it.
- Individual projects cannot evolve unless there is efficient feedback to all component parts.

It is best, in large projects, if this feedback is as *good* as possible.
Quality assurance must guarantee:

- objectivity
- accuracy

KEY: Quality Assurance entails testing *everything that is usefully testable* (through well-founded measurements)

Documentation

This involves producing *end-user documentation*. Some great programs are not commercial successes because the end-users can't understand how to use them!

In small projects it is OK for the core developers to do this.

If programmers are writing more text than code then this is time to appoint a documentor or documentation team.

The documentor should be a professional technical writer

They work directly with the core

KEY: Documentation is normally not a full-time role in any single project --- documentors often work on parallel projects.

Toolsmith

They create, adapt and maintain software tools that facilitate the production of project deliverables, especially generated code:

- A common task is to integrate the project's selected commercial development tools with the organisation's standard yet proprietary configuration management tool.
- In an incremental and iterative development process, the toolsmith is often asked by the core to produce small tools for very specific tasks.
- In small projects the role is normally a part-time job for a programmer
- In medium projects, the toolsmith is usually shared
- Only larger projects require a dedicated, full-time toolsmith

KEY: The toolsmith must have intimate knowledge of programming and the underlying support system and the core requirements

System Administration

Responsible for managing the physical computing resources: the same role as the toolsmith with respect to system hardware.

The role may sound mundane, but it is critical to the overall success of any project.

Librarian

Should be found in any organisation that is serious about re-use: of code, architectures, documentation, ...

This role is often done in conjunction with the toolsmith

They advise the core on abstraction and generality

Peripheral Roles

They represent people who are ultimate consumers of the system built by the core:

- **Patron** --- project champion (funder?)
- **Product Manager** --- coordinates marketing, training and support
- **End Users** --- provide *unequivocal* measure of success or failure
- **Technical Support** --- on the front line was the product is released

KEY-- The customer is always right (within reason). Don't ever believe that: they get in the way, are clueless, create conflict or have the right to ** up the project!**

Resource Allocation and Team Organisation

Understanding all the roles of a project is not enough, one must also organise these roles in a coherent whole:

A good software process provides the structure upon which these roles are fitted:

- Good processes encourage a centre of gravity around the project manager and the architects
- Within subteams, always have one person responsible for communication with other subteams (but ensure that all subteam members are capable of playing this role)
- In larger projects, set up *tiger teams* who wander from problem to problem across all areas of development (like a reserve cavalry) ready to fill the breach in times of need!

KEY: choose (re-use) a structure which maximises resource use, inter-resource communication and mutual understanding