Structures for Software Development Teams

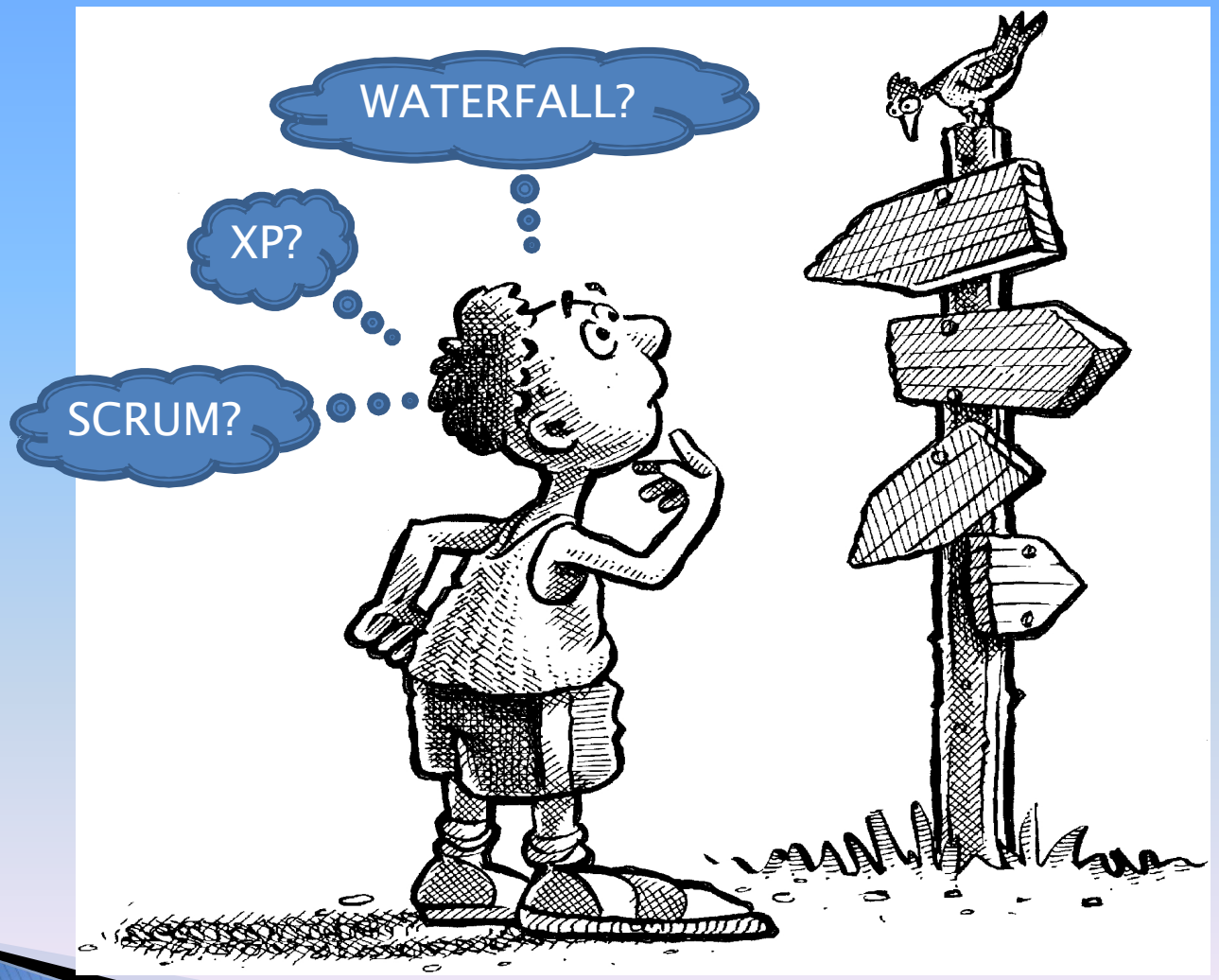
By: Ronny Trefftzs

CSCI 5828: Foundations of Software Engineering

Spring 2012

Professor: Kenneth Anderson

CONFUSED ABOUT FORMING YOUR SOFTWARE TEAM?



What is the Solution to the Structure of Your Software Team ?

- While there is really no standard solution, the following presentation will hopefully guide you into making the best choice for your situation.
- You must first understand what drives your team structure.
- High level explanations of software lifecycles will be presented.
- Typical roles within these lifecycles will be shown.
- Finally, guiding principles to develop a successful software team are shared.

What Is A Team?

- Software Development teams are made up of different types of individuals with different opinions, perspectives, and talents that need to work together and communicate effectively in order to successfully complete a project.
- Your software team may include architects, designers, testers, application developers, and/or experts in technology. Throughout the project all of these different members must communicate and resolve issues.
- Different teams and team members will interact to form a unique set of behaviors that must be managed.

What Are the Typical Roles?

- Project Manager
  - – This person develops the project plan which includes schedule, budget, and priorities. The project is usually staffed by this person. This person is responsible for the daily leadership of the team.
- Architect
  - – This person is responsible for the overall design of the system. The architect is key to project success. Brooks states, “Conceptual Integrity is the most important consideration in System Design.”

What Are Typical Roles?

- Architect
  - – The architect is the foundation for all to build on in order to help maintain that conceptual integrity. You will want to consider the architect a key team member. Since they may be scarce in your organization and cannot be a direct team member, you must schedule time and interface your team with the architects.
- Business/System Analyst
  - – These people generally model the processes and have a key role in developing the requirements.

What Are Typical Roles?

- Chief Programmer
  - – They usually define the functional and performance specifications. Also involved in design, coding, and documentation.
- Developer
  - – They typically are responsible for the development and implementation of the software.
- Tester
  - – Although listed last here, this person is a critical part of the team. They ensure test plans are developed and that system testing is carried out.

Team Communications

- As Brooks states, “large software projects suffer major issues due to division of labor.”
- As the number of communication paths, n, increases, the effort increases by:

  $n(n-1)/2$

  This indicates a critical need to develop a team that communicates with the least amount of paths necessary in order to help maintain schedule.

Team Communications

- The leader of the team will definitely influence the success of the team by the very nature of their communication style. Effective communication by a leader keeps the team focused and leads successful team performance.

Beginning to Build the Team

- Of course in a perfect world one could find the best-of-the-best team members with all the best skills needed. For finding developers, the most likely scenario will be to hire ones that are generally competent and offer training to develop whatever skills are necessary to build an effective team. Some training will naturally occur during development. But it is critical to get the right training upfront before the project begins.

Beginning to Build the Team

- At the start of a project the skill levels of needed team members should be assessed in the following areas:

1. Domain knowledge. Is this an Embedded System or a Business application on a client? Each requires a totally different set of skills relevant to their particular domain.
2. Development knowledge. Here any project will need skills in design and a particular programming language(s). Will the project need single experts in UML, C++, Java, Ruby, etc. Or will Polyglots be needed?

Beginning to Build the Team

3. Development Environment knowledge – List all the development tools available or that will be needed.
4. Software Technology Expert knowledge – Will expert architects or systems experts be needed?

- Make sure to have all computers, development environments, compilers, etc. ready for your team on day 1.
- Build the team around subsystem components. This allows management of smaller tasks. It also has the advantage of isolating issues to a smaller part of an entire system.

Beginning to Build the Team

- Tools
  - Make sure of what tools will be needed and what training team members will need. The list of useful tools is definitely long, and one usually will not find one person that is expert, or even efficient at all of them.
  - Do you need?
    - Requirements capture tools
    - UML tools for architecture capture.
    - Test automation tools
    - Configuration Managers
    - Code editors, compilers, linkers…

What Drives Team Structure?

- Company Status Quo
- Application Type
- Project Scale
- Skill Level of Available People
- Architecture

Company Status Quo

- Existing Architecture – If the company already has an existing architecture from which to build that is a great opportunity to reuse and you may have team members that are already very familiar. The team will definitely have an advantage in effort estimation using an existing, familiar architecture.
- Development Life Cycle in place – We don't always have a choice between Waterfall and Agile. The team will have to be staffed accordingly. However, a wise team will apply Agile techniques where it can, even if forced to Use Waterfall. In fact, today some companies will overlay an Agile approach on top of their Waterfall process.

Application Type

- Client Server – This type of application will require a skill set in networking and perhaps Object Oriented Technology.
- Mobile Device – Depending on whether team members are developing application on top of Android or iOS, or whether they are developing actual OS, JVM, or BSP software requires totally different skill sets.
- You must completely understand the application for your project in order to staff correctly.

Project Scale

- Large projects need a lot of manpower in order to develop in a timely manner.
- According to Brooks, on large projects, team size must be weighed against the number of communication paths. Too many paths and the effort increases exponentially.

Skill Level of Available People

- In a February 2010 study by the GAO, one of the major reasons given for software project failure was competent people.
- Skill is Object Oriented Design is more difficult to master than Functional Design.
- A high level of competence may be needed in use and understanding of UML.

Architecture

- James Rumbaugh defined architecture as “the organizational structure of a system,…components, connectivity, interaction, and principles that guide the design of the system.”
- In software development, architecture is the foundation for the entire software project.
- A sound software architecture is indeed critical for managing the work effort, partitioning the design and implementation, and provides a common ground for communication for the team.
- Use an existing architecture, or make sure to establish one before proceeding with feature implementation.

Architecture

- Use UML to show your architecture from different views.
  - Use Case View – Shows the behavior and functionality of the system. Items may include Use Case Diagrams, Scenario Diagrams, and Activity Diagrams. Use Cases help develop a communication path between your team and users of the system.
  - Logical View – Shows the system structure, and in UML can be represented by Class Diagrams.

Architecture

- Use UML to show your architecture from different views.
  - Component View – Shows how the logical pieces are grouped and can be represented by UML Package Diagrams.
  - Deployment View– Maps software subsystems to actual hardware.

Typical Tasks in All Software Lifecycles

- No matter which lifecycle chosen, all have similar tasks. The tasks are just performed or captured differently.
  - Feasibility Study
  - Requirements Analysis and Capture
  - Design
  - Coding
  - Testing
  - Maintenance

Traditional Development Cycles

Waterfall Lifecycle

- The earliest of the formal software lifecycles.
  - Even though it is presented in a bad light today, waterfall is still the most prolific software lifecycle used today.
- It seems to have been the basis for most, if not all, life cycles in subsequent years.
  - Spiral
  - Functional Decomposition

Waterfall Lifecycle – What is it?

- Tasks are serialized into phases.
- Team members can work across all phases, but typically do most of their work in the phase designed for their specialty.
- Requirements are developed and agreed to upfront, so your team members doing Analysis would need to play an active role in this phase in producing a Requirements Spec.
- Design follows next, so Technical Leads and Architects would be heavily involved.

Waterfall Lifecycle – What is it?

- After the design is complete, coding begins with all programmers on deck. In traditional waterfall, the requirements and design are set, so any issues caught in coding have major impact. More modern waterfall approach would allow some iteration to go back to a previous phase. So your software developer in the coding phase would have an “easier” time going back to change a design document.
- Finally, we have verification and deployment. Once these phases are reached, it is very costly to back up into other phases. Your team better have it right in these phases!

Waterfall Lifecycle – Roles

- Customer – Plays a limited role, if any, after the Requirements phase.
- System Engineer/Analyst – This person typically is responsible for developing the Requirements Specification, to include functional and non-function specs.
- Chief Engineer/Programmer – Responsible for the design of the implementation and the work of junior programmers.
- Architect – Responsible for the interfaces to software and hardware. Typically decides on the operating system and BSP for embedded system.

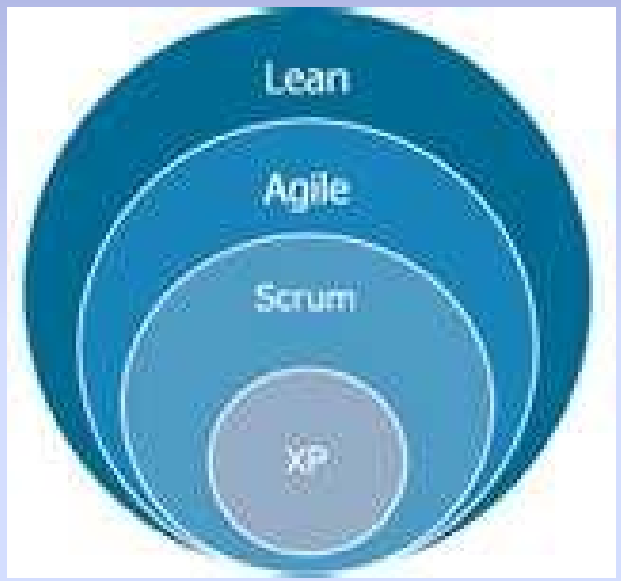
Waterfall Lifecycle – Roles

- Lead Test Engineer – Will work with the Chief Engineer and Developers towards the end of the Design Phase to develop the test plan and begin any upfront work to develop automated tests.

Waterfall Lifecycle – Team

- Team tends to be of a horizontal structure. That is, the team is full of specialists.
- Many requirements (Use Cases) are usually attacked by the team all at once.
- Tendency is for the team not to really have anything “done” until late in development. Pieces of many Use Cases are “done” early, instead of a few Use Cases being completely done early.

Agile Lifecycles



Agile

- Agile is a managed software process that allows frequent inspection and adaptation to change that provides continuous improvement.
- Desired team members are more than generalists. We want members that know a lot about a lot of things.
- Leads to a vertically oriented team structure, whereby User Stories can be assigned to a developer or two and worked from beginning to end by the same person or small group.

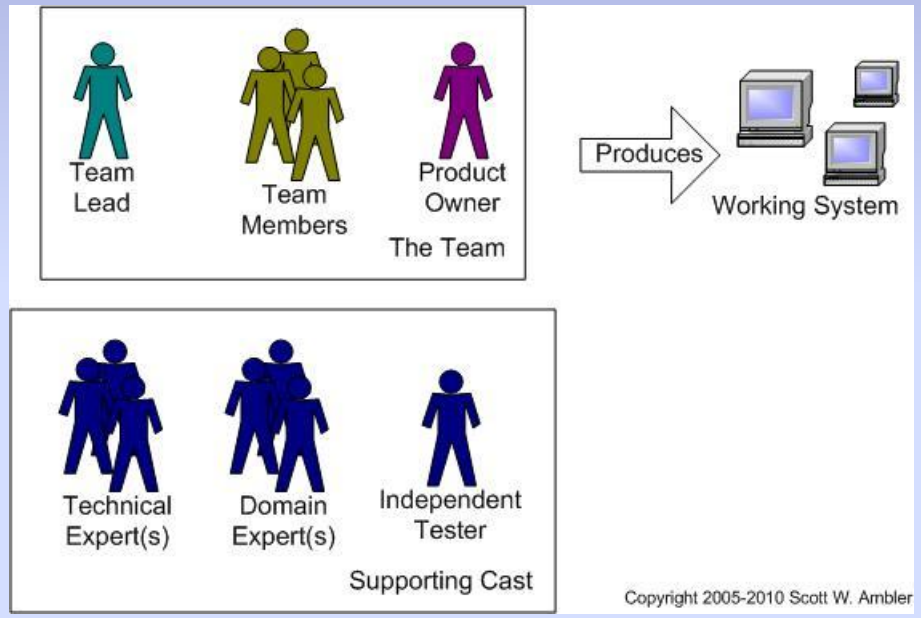
Agile

- In the real world, these type of members are not always readily available, so while team members develop these skills, interfacing to outside specialists will be necessary.
- The team members will be kept together and not shuffled around to other projects while working on a particular project. This leads to immense increases in productivity due to team synergy and increasing team members' skills.
- Team members deliver working software to the customer about every two weeks.

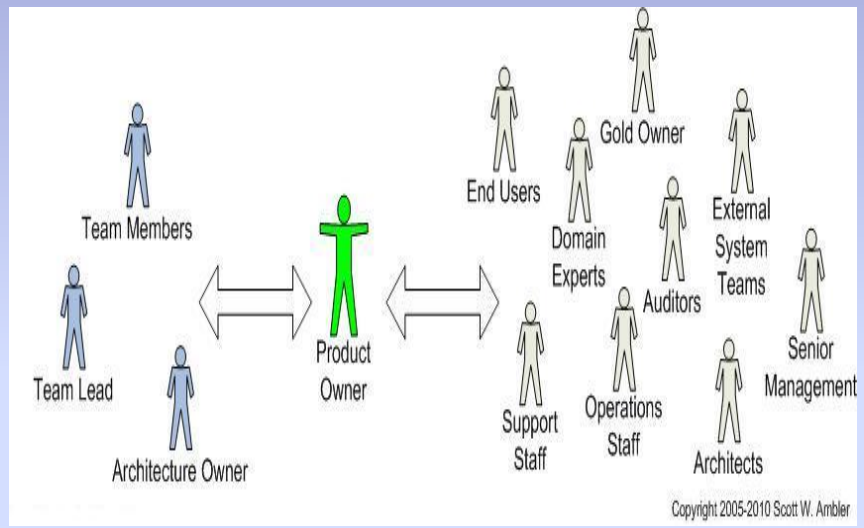
Agile

- The team must realize that since each person can assume multiple roles, the responsibility of a team member is usually much higher than with traditional life cycles.
- The team is cross-functional, self organizing, and accountable for every part of development, including screw ups!

Agile – Small Team Structure

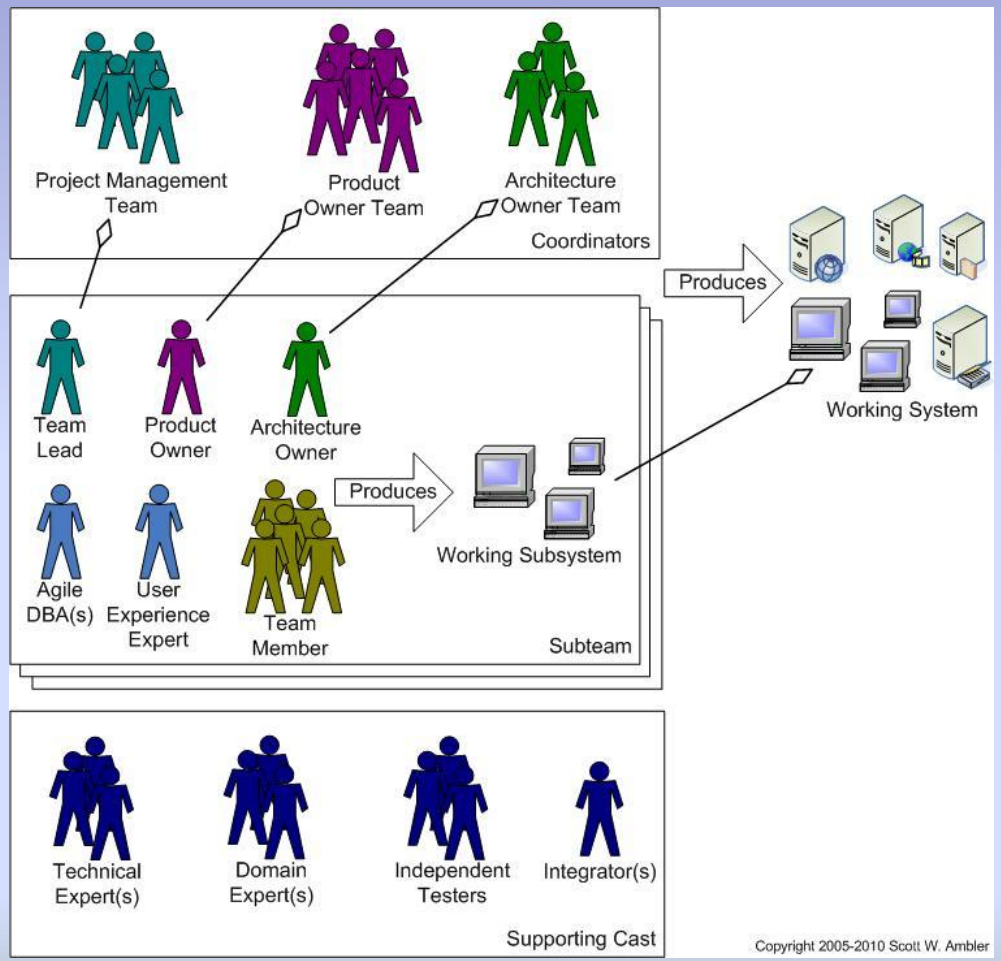


Agile – Small Team Structure.



- The Product Owner represents the interface between the team and stakeholders.

Agile – Large Team Structure



Agile – Large Team Structure

- Product Owner Team
  - Composed of one or more Product Owners
- Project Management Team
  - Composed of one or more Team Leads
- Architecture Owner Team
  - Composed of one or more Architects
- Key difference from small team is that large Agile team must have an architect. The architects keep conceptual integrity.

AGILE – XP


In Person Meeting
Trust
Agile Success Factors
Developer Time Estimation
Regular Working Hours

Agile – XP: What is It?

- XP is an Agile process that takes “everything to the Extreme.
- No production Software is deployed without a pair of your team members developing it.
- Testing is done immediately, as no software is developed except through first having failing tests.
- Team members run tests after every change.
- Development is iterative and easily adapted to changing customer requirements.

Agile – XP: What is It?

- Everyone on the team owns the software and anyone has the potential work on any portion.
- Team members are encourage to “grow” software, not build it.
- No one needs to work more than 40 hours a week!
- Team members are co-located. This brings immediate awareness to the current state of development.
- Team members constantly refactor code in order to get rid of “smelly” software.
- The team has the shared goal of providing value to the customer.

Agile – XP: Roles

- On-Site Customer – As stated in the Agile Samurai, “source of the truth” from which all requirements flow on an agile project. An actual customer or customer representative is key to the team. Some XP teams require one to be co-located. The customer sets the priorities for the team.

- Manager – They track how well the team is doing by monitoring things such as velocity, burn down charts, and work-in-progress (WIP). They keep external distractions from the team and remove any impediments that affect the teams progress. They make sure the team is co-located and that it has all the things it needs to be successful in the way of tools, computers, etc.

Agile – XP: Roles

- Analyst – This person helps to write User Stories. They do the detailed analysis. When a feature comes up for development, they are responsible for figuring out how things need to work.
- Programmer – The programmer takes the User Stories and creates working software. They help in the estimation and planning during reviews and planning sessions. They make technical decisions about tools, design and architecture.
- Tester – Writes the test cases for User Stories. These test cases are normally part of, or are, the Acceptance Criteria. The tester verifies that the deployed software works as expected.

AGILE – SCRUM

Agile – SCRUM: What is It.

- SCRUM is an Agile process that borrows many of the XP practices, but may scale them.
- Requirements are captured via User Stories and stored in a Backlog. A product owner sets the priorities of the stories, but the team decides what to work on within these priority boundaries.
- It should be noted, however, that SRUM is not a methodology. It is a framework within which an Agile development process is employed.

Agile – SCRUM: What is It.

- It should be noted, however, that SRUM is not a methodology. It is a framework within which an Agile development process is employed.
- The teams performance, whether good or not, is highly transparent. This allows a team to continuously improve.
- The Scrum Master mentors the team.
- The team must participate in a daily “standup” meeting where each member gets a change to let the team know what they did yesterday, and what they plan to do today.

Agile – SCRUM: Roles

- Product Owner- Represents the stakeholders. Responsible for the backlog and setting priorities. They make decisions that should be customer focused.
- Scrum Master – Responsible for facilitation, obtaining resources for the team, and isolating from external factors. This person will need project management skills outside of planning and scheduling. Their chief job is to maximize team productivity. They facilitate meetings and remove any impediments that hinder, or may hinder, team productivity.

Agile – SCRUM: Roles

- Developer/Team Member – Try for 10 members +- 2. They create and deploy the working system/software. In addition these team members have the responsibility for estimating User Stories. They have full autonomy and authority during a Sprint. This aspect is sometimes difficult for “old school” style management.

Agile – SCRUM: Old Roles Replaced

- Program Manager -> Scrum Master
- Business Analysis -> Product Owner
- Planning and Scheduling -> Team

Guiding Principles to Develop a Successful Software Team

1. Architecture comes first!
2. Build teams around the system(subsystem) architecture.
3. Plan to develop a team that can handle change. Change happens!
4. Brooks truism: A key to developing successfully is the correct amount of communication.
5. Use Object Oriented Technology.
6. Adopt an Incremental Development Strategy.

Guiding Principles to Develop a Successful Software Team

7. Allow continuous integration and testing to occur within the team, but keep system testing separate from the team.
8. Clearly define team members roles. This doesn't mean each member has just a single role. Competent generalists are normally favored.
9. Embedded developer and PC developers are typically not interchangeable in the PC to embedded direction without a lot of training.
10. Putting a large number of members on a team creates more interplay than effort. Don't do it!

Guiding Principles to Develop a Successful Software Team

11. Look for team players. Find developers that don't mind blurring the lines between analysis, design, coding, and testing.
12. Brooks Truism, "Make a clear distinction between architecture and implementation."

References for Further Reading

- Grady Booch, James Rumbaugh, Ivar Jacobson. The Unified Modeling Language, Addison-Wesley, 1999.
- Frederick Brooks. The Mythical Man Month, Addison-Wesley, 1995.
- Dean Leffingwell, Agile Software Requirements, Addison-Wesley, 2011.
- Jonathan Rasmusson, The Agile Samurai, The Pragmatic Bookshelf, 2010.
- Ken Anderson, Agile Methods and Teams lecture, CSCI 5828, Spring 2012.

- Internet References
  - http://www.ambysoft.com/essays/agileRoles.html
  - http://www.epmbook.com/structure.htm