Teamwork

A team is:

> A group of people working together on a common goal.

Teams are often comprised of people with complementary skills that are able to accomplish teamwork:

> The ability to cooperate and communicate effectively with others to achieve a common goal.

In the software domain the kinds of people who are often involved are:

- developers
- testers/QA
- managers
- development leads
- architects
- operations
- product managers
- project managers
- sales
- customers

Each of these people bring specific sets of skills and knowledge to a project. Usually teams have only a small subset of these people actively engaged at any given time; the composition of a team will vary depending on the project and the work being undertaken.

Software teams tend to be fairly small (2-10 people). Teams above this size tend to get bogged down in administrative and communication

overhead. It is this overhead which decreases the productivity benefit of adding additional engineers to a project as captured by Fred Brooks Jr. in "Brook's Law":

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

That said, modern software systems are too large for individual developers to tackle alone and working with a team (and usually having many many teams working together) is often necessary. This is not all bad news, there are many positive reasons to work as a team:

- Teammates are a great source of extremely domain-specific feedback. No-one can understand a technical problem as accurately as someone working on exactly the same thing, while still being able to provide an alternate viewpoint for tricky technical problems.
- All team members bring different sets of background experience to a project. Over time, teammates contribute to each other's skills through both implicit and explicit knowledge transfer.
- Teams allow for specialization: as groups grow, members can take on specific tasks that they are good at enabling others to pursue tasks they are more effective at.
- The motivational factors of teams cannot be overlooked: we don't want to let down our co-workers and as such often work harder to ensure our shared success.
- Ultimately as software grows teams are needed to manage the complexity of modern software projects as no one person can effectively manage this on their own.

Working in a team can be extremely challenging; instead of just considering your own needs you must also consider the needs of others and of the complete team. How people are motivated and their degree of interpersonal maturity plays an important role in people becoming

effective teammates. Three coarse-grained levels of this maturity are:

1. **Dependent**: Software engineers who are completely dependent on their teammates are not able to contribute effectively on their own. These teammates are often challenging to work with as they cannot be relied upon to effectively complete tasks.
2. **Independent**: Engineers who are able to perform meaningful tasks on their own are independent. These developers are much more reliable than dependent engineers, although they often focus on their own activities over those most important for a team. These engineers are often intrinsically motivated: they are motivated by working autonomously, improving their skills, and accomplishing things on their own.
3. **Interdependent**: Software engineers who are able to keep a global view of the needs and goals of the team, and its members, are interdependent. These developers see the success of every team member as win-win for the organization and are able to effectively help all teammates be as effective as possible. These engineers are often extrinsically motivated: they are motivated by seeing the team succeed as a whole.

Every team is different, but some high-level guidelines are often useful when considering how your team works together.

- **Effective communication**: Communication is at the core of all team cohesiveness. Teammates must interact with each other in open and positive ways. In technical environments, it is natural to have disagreements about how specific decisions; being constructive during these discussions is important so the focus can stay on the merits of the technical issues being discussed and not be directed personally towards the people holding opposing views.
- **Project management**: All projects start with broad goals, but these

must be refined and more fully planned before they can be accomplished. By decomposing high-level tasks into more specific low-level tasks that can be better reasoned about, discussed, and scheduled, teams can gain a better understanding about what they are collectively trying to accomplish and how each of them is contributing to the overall success of a project. User stories and milestone plans are two common techniques for project management used by agile teams.

- **Conflict management**: Few people enjoy conflict with their peers and will go to great lengths to avoid it. Unfortunately, avoiding conflict often exacerbates the underlying problem. On technical teams, conflicts often arise due to communication failures; continual open dialog is one of the most effective mechanisms to ensure problems do not become overwhelming or project-threatening. Daily standup meetings in scrum-based projects exist for exactly this reason: to detect problems early and help the team find effective resolutions. It has been shown that treating conflict as an opportunity for problem solving or give-and-take negotiation is more effective than withdraw (avoiding the problem), smoothing (pretending the problem doesn't exist), or forcing (unilaterally applying one solution) in practice.

The Phoenix project details five team dysfunctions that can all be traced back to the discussion above:

- **Absence of trust**: Teammates can be unwilling to take risks when they fear personal attacks over their performance.
- **Fear of conflict**: Avoiding conflict and supporting artificial harmony is detrimental to overall team performance; taking advantage of conflict as opportunities to solve problems helps teams move forward.
- **Lack of commitment**: Insufficient communication and discussion

can cause ambiguity during project planning.
- **Avoidance of accountability**: Dependent and independent engineers focus on their own productivity can lead to counterproductive behaviours.
- **Inattention to results**: Failing to be mindful of the the win-win nature of team success instead of personal success can cause teammate to focus on the wrong success metrics for the work they are performing.

Teamwork and the course project

The course project represents a challenging development activity that must be completed in groups. To support this we provide the follow pieces of concrete advice based on past student experiences in the course.

It is extremely important to be clear about roles and responsibilities of each team member. Assumptions about scheduling, motivation, and communication are frequently challenging and are often not discovered until late in the term. You should be meeting with your partner regularly (this should be easy with three lectures and a lab each week); take a few minutes in each of these touch points to catch up and make sure you are both proceeding according to your shared plan.

Having a shared plan is key so you understand what your teammates are doing and how progress is proceeding. Don't just split the sprints into 'my half and your half', instead expand these into a concrete set of tasks with details about their validation and due dates so you can track progress. Using the GitHub issue tracker is a fantastic way to do this. Figuring out a core set of issues and developing a plan will only take a few minutes at the start of the sprint and is time well spent.

While the notion of a *teamwork agreement* might sound like overkill, they can be helpful for getting all team members on the same page with respect to their expectations for each other. This agreement should actually be written down (say in `OurContract.md` in your repo) and should detail some subset of:

- Expectations each teammate has for each other.
- How often meetings should take place.
- How communication should be handled:
    - Emails?
    - Texts?
    - GitHub issues?
    - How quickly should responses be given (aka 12h max turnaround)?
- How should the version control system be used:
    - Frequent or infrequent pushes?
    - Branches or all on master?
- What development guidelines should be used:
    - What is your test strategy?
    - Can failing tests be committed or must they all always pass?
    - Should all commits be explicitly linked to issues in the issue tracker?
    - What is the high-level test strategy?

These contracts only work if everyone reads and agrees to them. We have found more than any other thing communication breakdowns are the root cause of most teamwork problems in this course.

It is extremely important that you remember that your partner is your **best** resource in this class. Your success is their success. You must choose your partner **wisely** because you will be working with them for the full term, **no** team changes will be permitted.

References

- On dependence, independence, and interdependence from a psychological point of view.
- Another view on SE teamwork.
