Weekly status reports
1. Goals from previous week
2. Progress you’ve made this week, what you’ve done, what worked, where you had trouble, and what you’ve learned.
3. Outline your plans and goals for the following week (including who is responsible).

Today
• Teams
• Requirements (brief)

There is no “I” in TEAM.
(But there is a “ME” 😊)

Addressing software complexity:
Three related dimensions

<table>
<thead>
<tr>
<th>Artifacts</th>
<th>People Who does the...?</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Requirements</td>
<td>• Requirements</td>
<td>Requirements</td>
</tr>
<tr>
<td>• Design</td>
<td>• Design</td>
<td>Design</td>
</tr>
<tr>
<td>• Implementation</td>
<td>• Implementation</td>
<td>Implementation</td>
</tr>
<tr>
<td>• Testing plan</td>
<td>• Testing</td>
<td>Testing</td>
</tr>
</tbody>
</table>
| • ... | • ... | ...

• In other words, significant software can’t be built using a single artifact by a single person at a single instant
• Every significant piece of software is built with some view – albeit, often implicit – with respect to these three dimensions
• There are huge variations in each of these dimensions and in their composition

Team structures

• Tricky balance among
  – progress on the project/product
  – expertise and knowledge
  – communication needs
  – ...
• "A team is a set of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable."
  – Katzenbach and Smith

Why teams?

• Benefits
  – Attack bigger problems in a short period of time
  – Utilize the collective experience of everyone
• Risks
  – Personality conflicts
  – Coordination issues
  – Need to establish clear ownership or can have duplication of effort
  – Member can just “go along” instead of sharing potentially great ideas
  – Not taking individual responsibility/accountability because it’s a group
  – Need to be careful to have the “right” number
Common software team responsibilities

- Project management
- Functional management
- Developers: programmers, testers, integrators
- Lead developer/architect ("tech lead")
- These could be all different team members, or some members could span multiple roles.
- Key: Identify and stress roles and responsibilities

Organizational questions

- How do you decide who should be project manager?
  - What's the difference between project manager and tech lead?
- How do you divide your team into subgroups? Who will work on what, and with whom?
- How will we make decisions about our project?
- How will everyone communicate and stay in sync about important decisions and issues?
- What will we do if someone is not doing their share?
  - How can we motivate team members to prevent this?

Factors for success/failure

- Presence of a shared mission and goals
- Motivation and commitment of team members
- Experience level
- Team size – and the need for bounded yet sufficient communication
- Team organization – and results-driven structure
- Reward structure within the team – incentives, enjoyment, empowerment (ownership, autonomy)

Demotivators

- Micro-management or no management
- Lack of ownership
- Lack of effective reward structure
  - Including lack of simple appreciation for job well done
- Excessive pressure and resulting "burnout"
- Lack of focus in the overall direction
- Productivity barriers
  - Asking too much; not allowing sufficient learning time; using the wrong tools
- Too little challenge
- Work not aligned with personal interests and goals
- Poor communication inside the team
Communication: powerful, costly, tricky!

- Communication requirements increase with increasing numbers of people
- Everybody to everybody: quadratic cost
- Every attempt to communicate is a chance to miscommunicate
- But not communicating will guarantee miscommunicating
- But too much communicating may be the same as no communicating

Team structure models

- Dominion model
  - clear chain of responsibility
  - people are used to it
  - single point of failure at the commander
  - less or no sense of ownership by everyone
- Communion model
  - a community of leaders, each in his/her own domain
  - inherent sense of ownership
  - people aren't to this (and this scares them)

Leadership model?

- Who makes the important product-wide decisions in your team?
  - One person?
  - All, by unanimous consent? What if there isn't unanimity?
  - Other options?...
- Is this an unspoken or an explicit agreement among team members?
Organizing around function

- Pragmatic Programmer tip: “Organize around functionality, not job functions.”
- What are some benefits of organizing teams around function vs. around job functions/titles?
- Who will do the ... scheduling? development? testing? documentation (spec, design, write-ups, presentations)? build/release preparation? inter-team communication? customer communication?

Kinds of teams

- problem-resolution: a focused attack on specific bugs, problems, issues
- creativity: coming up with and exploring new ideas
- tactical-execution: carries out a defined plan

- Some team models
  - business: tech lead and a bunch of equal devs
  - chief programmer / surgical: lead dev does most of work
  - skunkworks: turn the devs loose
  - feature
  - search-and-rescue: focused on a specific problem
  - SWAT: skilled with a particular advanced tool(s)
  - Professional Athletic: carefully selected people w/ very specialized roles
  - Theater: "director" assigns roles to others

Surgical/Chief Programmer Team

[Baker, Mills, Brooks]

- Chief: all key decisions
- Copilot: chief’s assistant
- Administrator: manages people, hardware, resources
- Editor: edits chief’s documentation
- Secretaries (2): for administrator and for editor
- Program clerk: keeps all project records
- Toolsmith: builds programming tools for chief
- Tester: develops and runs unit and system tests
- Language lawyer: programming language expert, advises chief

Microsoft’s team structure [microsoft.com]

- Program Manager. Leads the technical side of a product development team, managing and defining the functional specifications and defining how the product will work.
- Software Design Engineer. Codes and designs new software, often collaborating as a member of a software development team to create and build products.
- Software Test Engineer. Tests and critiques software to assure quality and identify potential improvement opportunities and projects.
Toshiba Software Factory [Y. Matsumoto]

- Late 1970's structure for 2,300 software developers producing real-time industrial application software systems (such as traffic control, factory automation, etc.)
- Unit Workload Order Sheets (UWOS) precisely define a software component to be built
- Assigned by project management to developers based on scope/size/skills needed
- Completed UWOS fed back into management system
- Highly measured to allow for process improvement

SCRUM: pigs and chickens

- **Product Owner** represents the customer
  - Ensures that the team maintains a proper business perspective
  - Writes user stories, prioritizes them, etc.
- **ScrumMaster** facilitates
  - Acts as a buffer between the team and distracting influences
  - Ensures that the Scrum process is respected
- **Team** delivers the product
  - Typically 5-9 people with skills to do the work (design, development, testing...)
- **Users** to whom the software will provide value
- **Stakeholders** (customers, vendors) who enable the project and for whom the project will produce the agreed-upon benefit
- **Managers** who set up the environment for the product development organizations
- These roles are far less directly connected to the process

Results-driven structure

- Clear roles and responsibilities
  - Each person knows and is accountable for their work
- Monitor individual performance, hold people accountable
  - Who is doing what, are we getting the work done?
- Effective communication system
  - Available, credible, tracking of issues, decisions
- Fact based decisions
  - Focus on the facts, not the politics, personalities, ...

Alverson suggests

- **Pragmatic Programmer**
  - Pragmatic Teams, p. 224-230
- **An interview with Patrick Lencioni on “The Five Dysfunctions of a Team”**
- **Software Project Survival Guide**
  - p.103-107 on team organization
Requirements (in brief, see web page)

- DRAFT DUE: Friday April 9 by 6PM.
- REVISED "FINAL" VERSION DUE: Friday April 16 by 6PM
- Your first deliverable is a set of requirements documents (sometimes called "Software Requirements Specification" or SRS). These describe the goals of your project and how users will interact with it (the high-level UI design). You will also document your plans for completing the project. Note that this document will be a living document. You will be asked to provide updates to it at periodic points in the development cycle.
- Submit a document, 4-5 pages in length not counting the UI diagrams or use cases, that answers the questions in the "Product Description" and "Process" sections below. One per team.

External requirements

- The product should be as usable as possible, even for people who are not expert computer users (this holds less for the clone detection project, where the users are likely to be expert computer users). The product must be robust against errors that can reasonably be expected to occur, such as invalid user input, lost network connections, etc.
- The scope of the project must match the resources assigned.
- Beyond these requests, you are largely free to take the next turn of the product development spiral and firm up your product requirements. This requirements document will essentially be a contract with the course staff for what you plan to deliver. Consequently, you should talk to us as you plan.

This document....

- ... will essentially be a contract with the course staff for what you plan to deliver. Consequently, you should talk to us as you plan.

Product description

- What is your product? Who is the target audience you expect to use the product? What problem is it solving?
- What alternatives exist, and what are their strengths and weaknesses? How will your system be different, from the user's point of view?
- What are its major features? Include at least 4 major features you will provide, along with at least 2 other minor features or aspects you hope to complete.
What will the UI look like?

- Submit diagrams (at least two, possibly more) containing rough sketches of your product’s user interface. These diagrams should depict the major UI used to complete the use cases you submit. For example, if one of your use cases was (for a different project) to Purchase Stocks, you should draw the initial UI that is presented when the user wishes to purchase a stock, along with any other major windows, messages, etc. that appear as the user navigates through this use case. The diagrams can be drawn by hand or computer. If a window leads to a dialog box, drop-down box, etc., include it as a sub-diagram. Your diagrams do not need to be pretty to get full credit, but they should be clear and legible. The main point is to illustrate your thoughts about what information should be shown and how the user will use the software.

Use cases

- The format is up to you, but these aspects must be clear: actors, preconditions, minimal/success guarantees, the list of steps to the success scenario, extensions/alternatives/failures, and failure-handling remedies as appropriate (or a statement that you do not have one and what you will do to generate one). It is impossible to think of every possible use case or failure mode ahead of time, and not useful to drown the reader in boring details. Make a brief, informal argument that your set of use cases covers the important scenarios (perhaps by referring back to the product description).

Process

- Software Toolset: What programming languages, data sources, version control, bug tracking, and other tools will you use? You must use a version control system and you must use a bug tracking system. The course staff must have access to the bug tracking system.
- Group Dynamics: For the most part, your group organization is up to you. However, we require that you choose a single person to serve as your Project Manager (PM). Who will be your project manager? What will be the other members’ roles? Will everyone share in the development, or will you have designated designers, testers, etc.? Why have you chosen these roles? Will the roles differ for different parts of the project?

Schedule/timeline

- Provide a rough schedule for each member or sub-group within your team. For example, how long you believe your developers will spend working on each major feature listed in your product description? Who will work on the design, and how much time do you expect it will take? Which features are “beta” features? Provide reasonable guesses as much as possible, but you will not be graded on the accuracy of these predictions.
Risk summary

- What are the major risks regarding completing your project? What are you most worried about, and why are these the most serious risks? Describe specific experiments that you will perform to gather information that will reduce the risk. Also describe what you will do if you are unable to overcome the problems — for example, if you cannot get an external component to work, or if you fall behind schedule. This might include feature cuts, but not every one of them may be a feature cut; others might include adjusting your group dynamics, time schedule, testing, etc.

- As a special case of risk reduction, describe at what point(s) in your process feedback from an external user (that is, the staff in this case) will be most useful, and how you will get that feedback.

Questions?