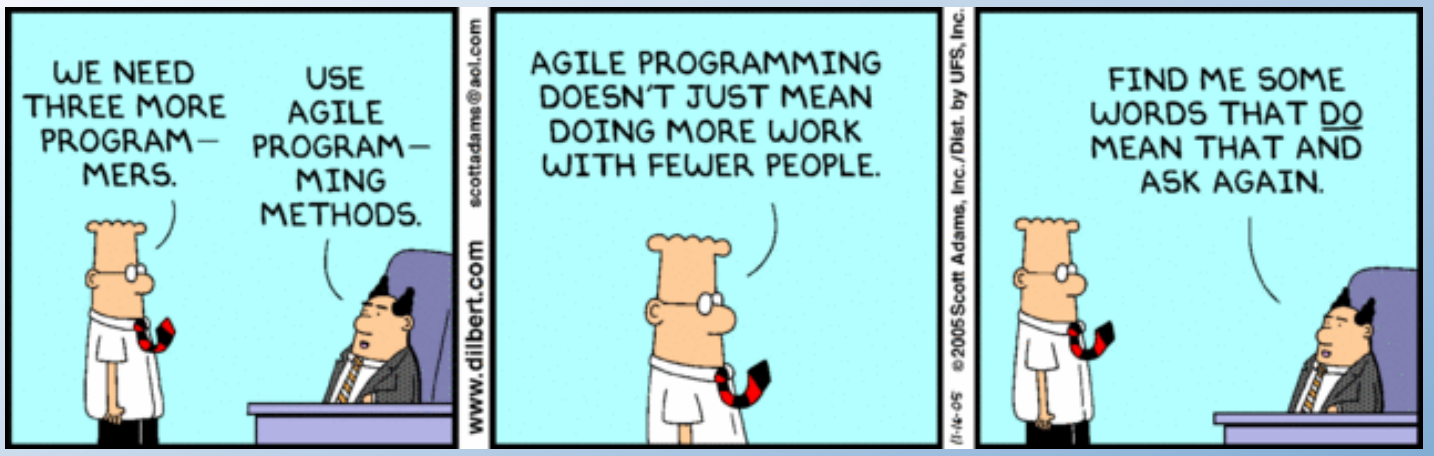
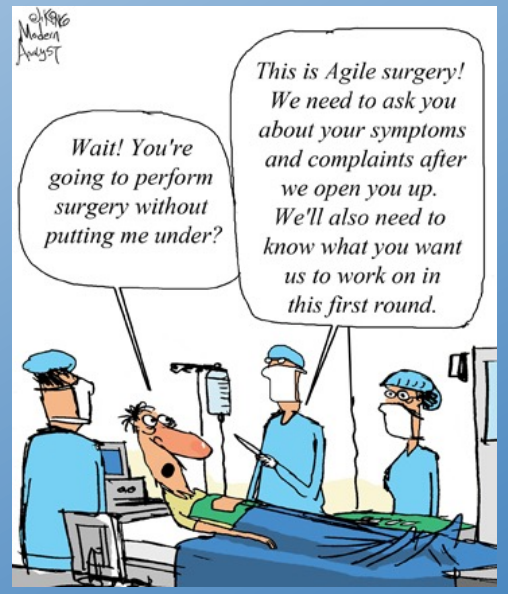
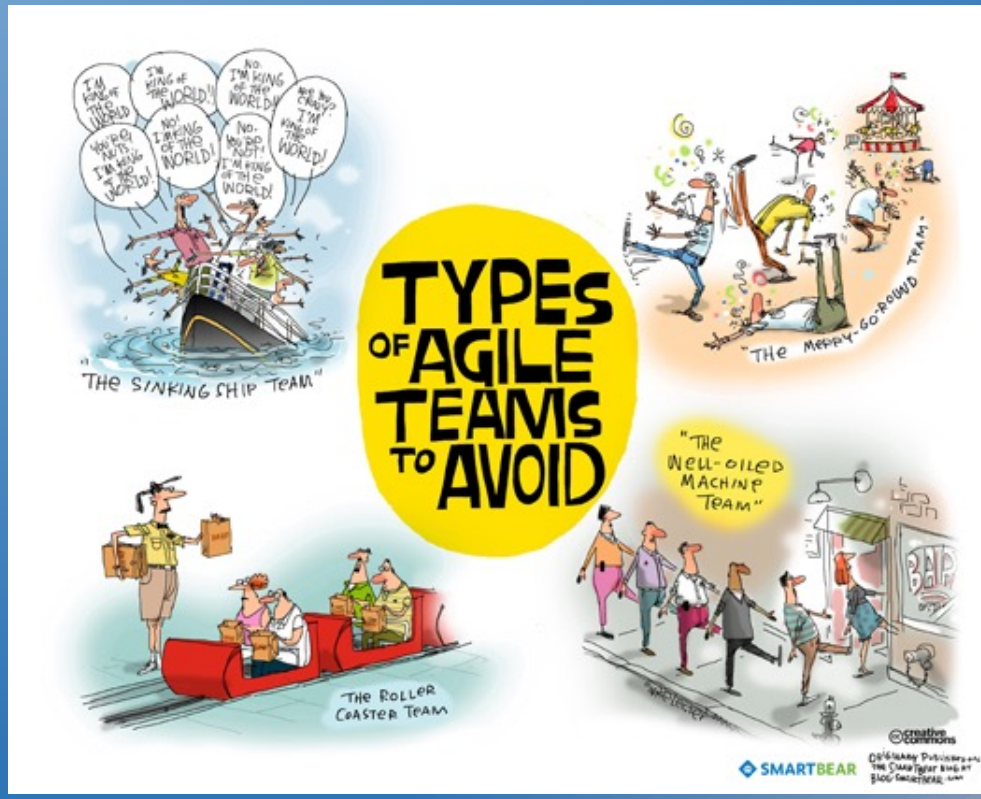




Team Management

CSCI2340: Software Engineering of Large Systems

Steven P. Reiss



Continuing Preparation for Programming

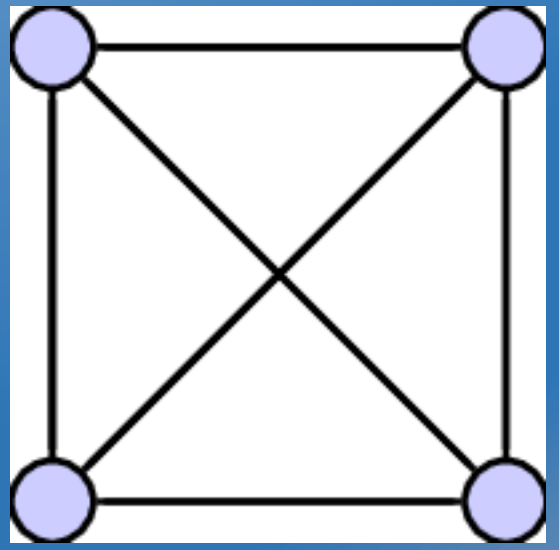
- Working in teams
- Collaborative software development
- Workflows and actions
- Cost Estimation
- Test-Driven Development

Working in Teams

- Required for large projects
  - Too big to do on ones own
  - Multiple people can work in parallel
- Requires Effort
  - Coordination, organization, management
- Difficult to make productive
  - Mythical Man Month (Fred Brooks)
- Tools can help
  - Software process programming
  - Modern team management tools

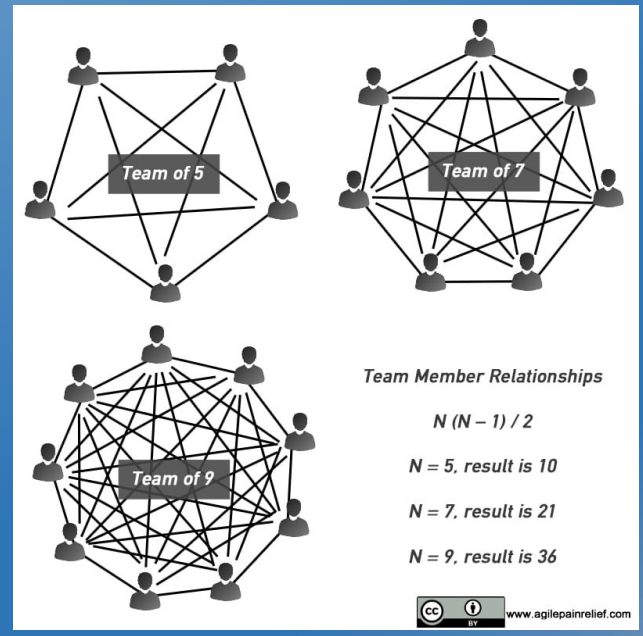
Team Size Matters

- Small teams (<= 4)
  - Can get everyone together
  - Number of communication paths is small
  - Meetings can be productive, even if not organized
  - Easier to move people around
  - People can learn the whole system
  - Easy to divide project into logical pieces
    - Front end, back end, database, business logic, …
    - People can work independently

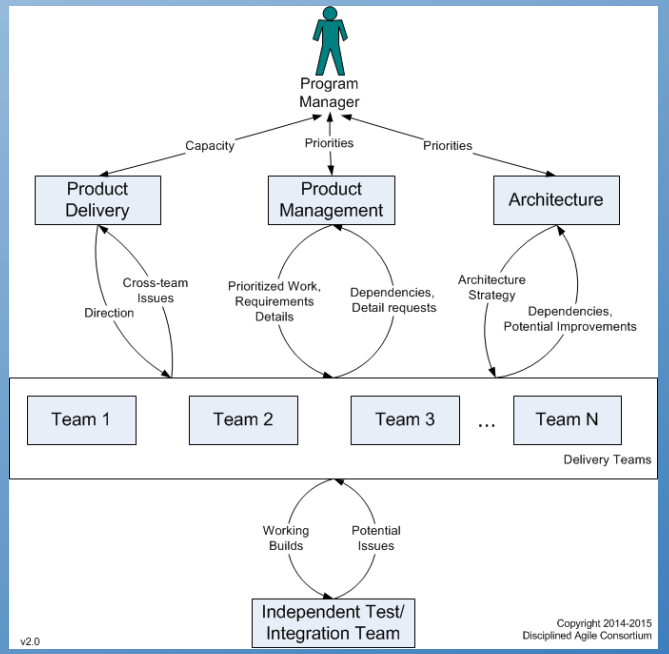


Team Size Matters

- Moderate Teams (~6-12)
  - Assume at least one person will be absent from any meeting
  - Assume at least one person will flake out (get sick, …)
  - Can’t communicate with everyone
  - Meetings take longer & get less done
  - Need better coordination
  - Project needs to be managed
  - More difficult to divide into independent pieces
  - More critical paths (things that can go wrong)
  - Difficult to understand the whole system



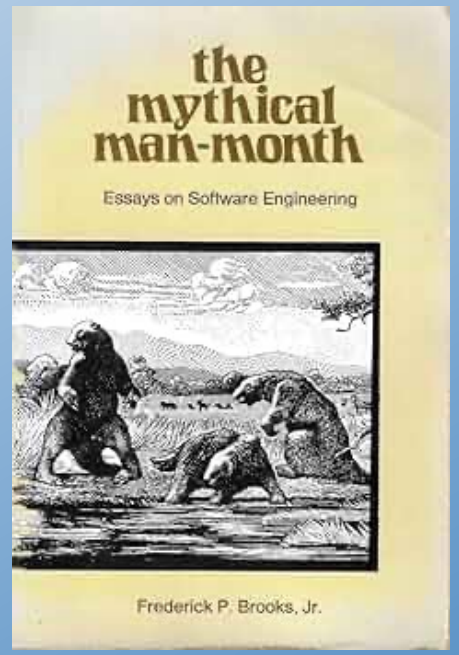
Team Size Matters



- Large Teams (15+)
  - Require a hierarchy
  - Target system generally quite large
    - No one knows all the details
    - Most members don't know the complete system
      - Concentrate on your piece and how it fits in, not everything else
  - Different strategies are used
    - These are out of favor except for very large systems
    - Divide the system into separable components
      - Build these independently using smaller teams
      - With common libraries or frameworks (separate teams)

Mythical Man-Month

- Adding people to a project can delay it further
  - Can’t measure effort in terms of man-months



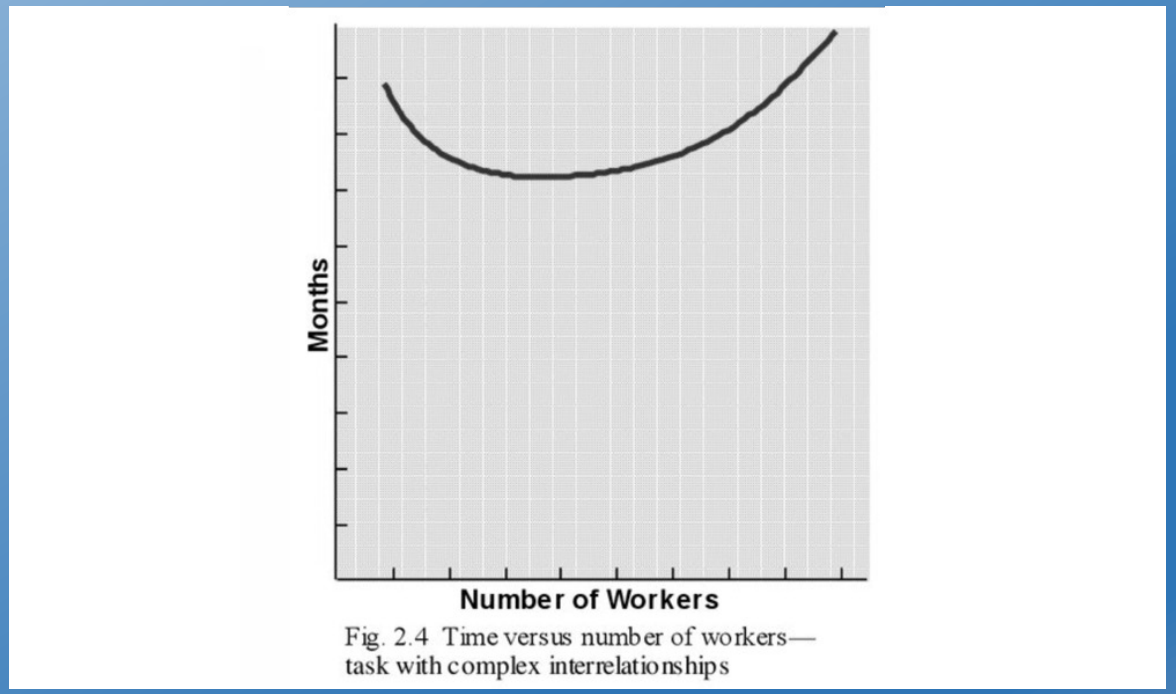


Fig. 2.4 Time versus number of workers—task with complex interrelationships

Team Management

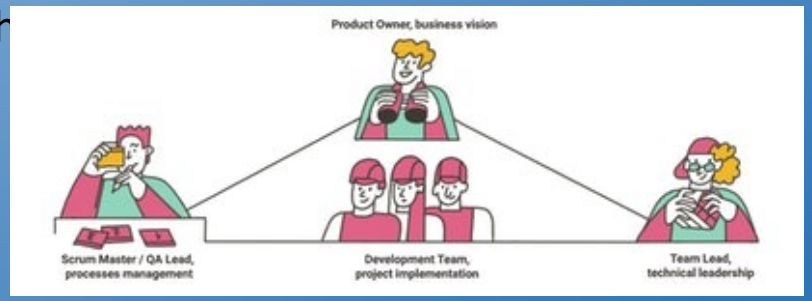
- Software teams need to be managed
  - Ensure equality of workload
  - Ensure consistency of decisions
- Democracy doesn't work
  - Someone must make hard decisions
  - Someone must ensure consistency
  - Someone must make sure everyone is pulling their weight
    - People aren't getting in over their head
    - People work on things they are best at

Team Leader

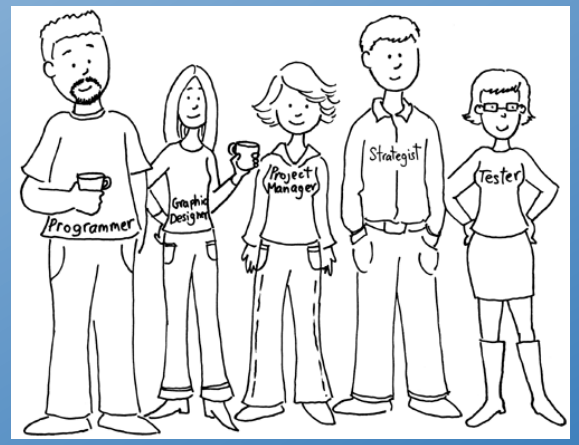
- Need a team leader or manager
  - Break project into tasks
  - Assign people to tasks
  - Keep track of what everyone is doing
  - Coordinate where needed
  - Reassign people as needed
  - Make critical decisions
  - Ensure consistency
  - In charge of presentations

Team Management: Other Roles

- Assistant Manager or Leader
  - Manager not available 24/7, might get ill, might need help
- Product Managers
  - In charge of a portion of the software (e.g., front end or back end)
- Documentation Manager
  - Organize files, documents, requirements, repository, versions & branch
    - This is what I see to grade you on
- Quality Assurance (QA, Testing) Manager
  - Create system test cases, supervise testing, …
  - Approve code for release
- User Interface Guru
  - Ensure a consistent, easy-to-use, nice-to-look at user interface
  - Designing look and feel, icons, logos, etc.
- Provisionary
  - Handles setting up or provisioning AWS, VMs, Containers, databases, ...



Team Management: Other Roles



- Security, Privacy and Ethics Czar
  - Ensure system is secure & fair
  - Security testing, fairness testing
  - Set privacy policy, enforce privacy policy
- Performance Analyst
  - Determine where and when there are performance problems
  - Performance testing
- Scribe
  - Take notes at meetings; ensure information is current
- Domain Experts & Users
  - Provide essential information about the problem being solved
- Skeptic
  - Question everything. Avoid risks and failure.

PROJECT

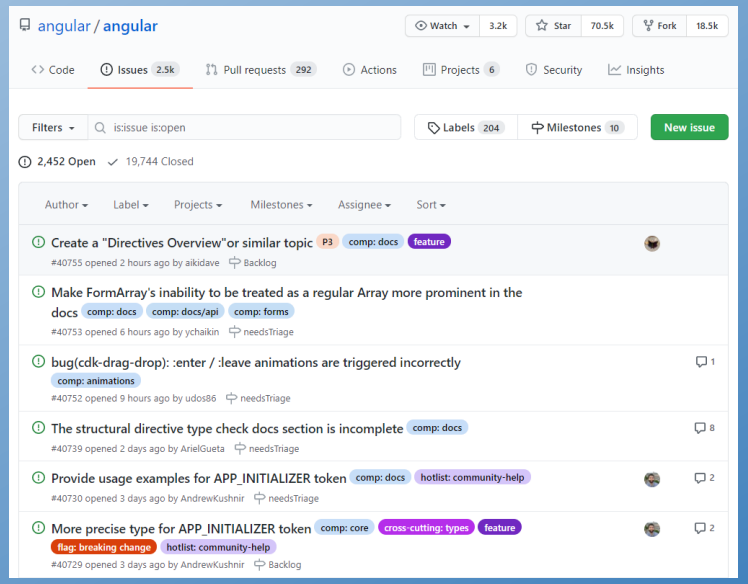
- Let's have a short project meeting
  - Discuss roles
  - Tentative role assignments
  - Check status
  - What should be done by whom this week
    - Requirements, specifications and software architecture
  - 10 minutes
- Then we'll get into tools & other topics

Team Communication

- Regular communication is key to a successful team
  - Knowing how your part fits with others
  - Getting things done in a time fashion
  - Negotiating who does what
  - Negotiating interfaces between components
- Physical meetings can be difficult
  - Team members might be distributed around the world
  - People have other commitments
- Tools for communication
  - SLACK
  - Discord
  - JIRA
  - GITHUB (wiki, issues)

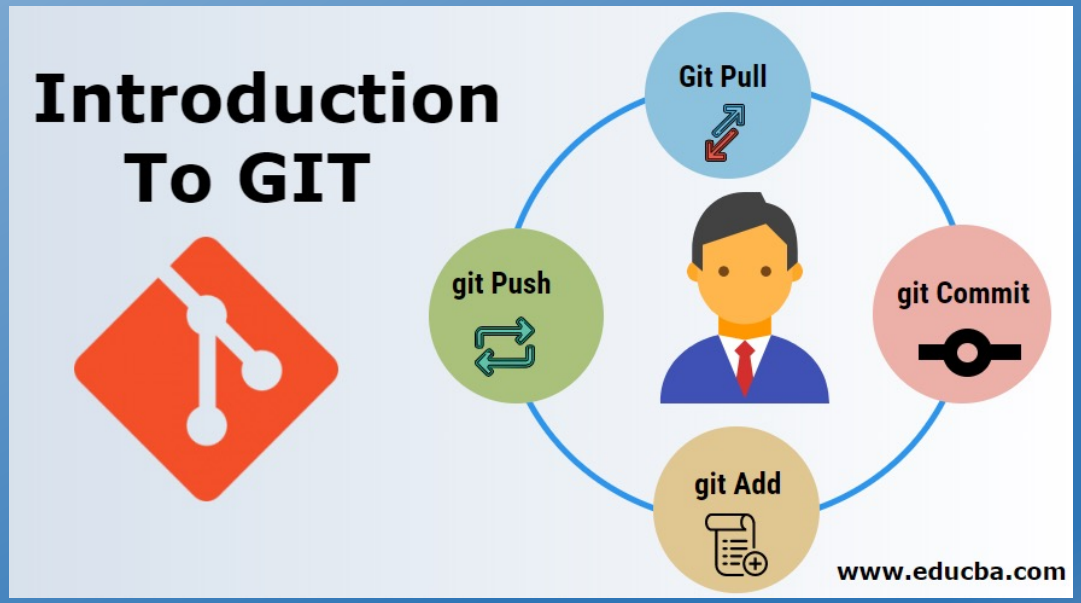
GitHub Communication Tools

- GitHub Wikis
  - Good for documentation, notes, comments
    - Internal as well as external documentation
  - Let others provide feedback
- GitHub Issues
  - Items in repo to plan, discuss and track work
  - Bug tracking
  - Combines slack-like messaging with project planning

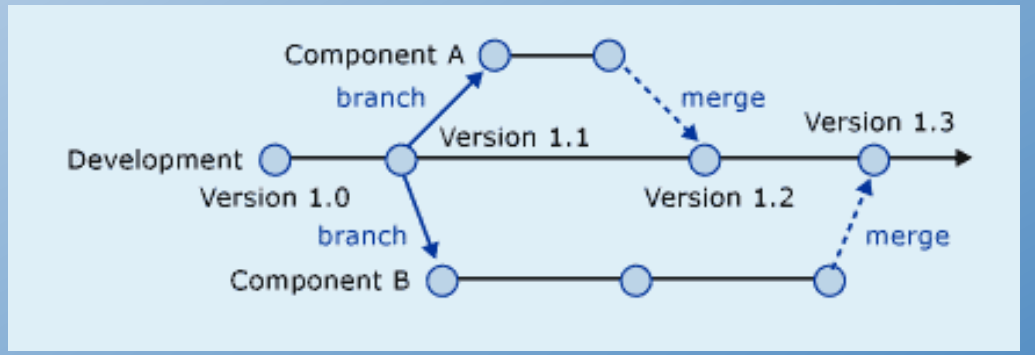


Version Management: GIT

- Designed for modern software development
  - Flexible repositories
  - Ease of creating branches
- Designed for agile development
  - Good for all development



Software Branching



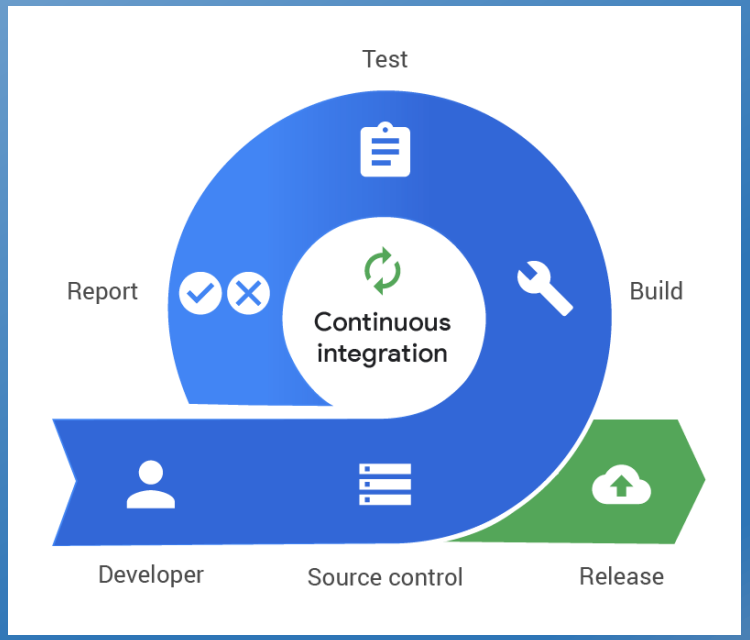
- Individuals (or smaller teams) work on an extension or feature
  - Interim system might not be usable
  - Extension might not work or be desirable at the end
  - Multiple such extensions developed at once
- Want to allow these teams to work productively
  - Without interfering with one another
- This can be accomplished with branches
  - Different, independent versions of the software
  - Developers (or small teams) work on their own branch
- But branches aren't necessary
  - Can develop a substitute method or class
  - And make using that class conditioned on a flag or environment variable
  - A bit more work as the status quo has to be maintained

Avoiding Branches

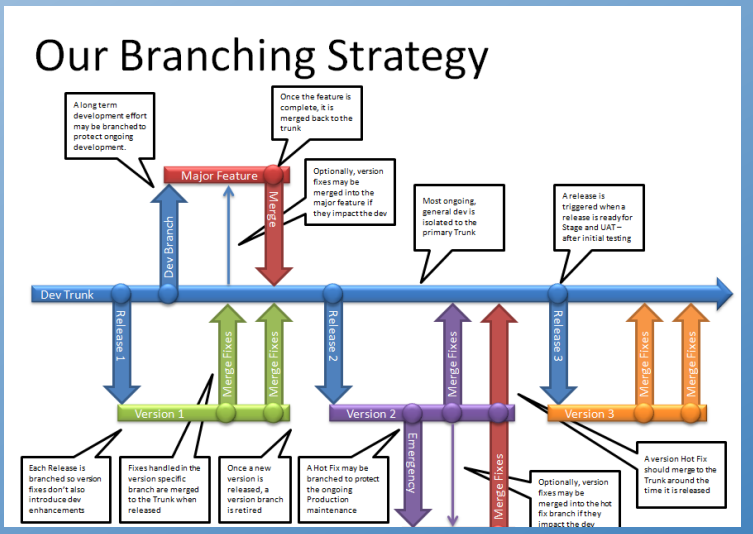
- Branches can create confusion
  - Coding for the past, not the future
- Eventually branches need to be merged
  - With each other
  - With the main system
- Problems arise when there are conflicting changes
  - Ideally these are avoided
  - Selecting features to avoid conflicts
  - In practice, they are common
  - Branching makes these worse
- Need a team strategy for handling merge problems
  - Team manager, negotiation, code ownership, …

Continuous Integration

- Always have a working version of the system
  - Minimize branching for everyday development
  - Merge all code changes regularly (daily/weekly/…)
  - Build and test the system on merge
  - Automate this process as much as possible
- Pros
  - Simplifies merging
  - Helps find bugs faster (continuous testing)
  - Improves productivity
  - Supports dogfooding
- Cons
  - Requires development that can be merged
  - What to do when tests fail

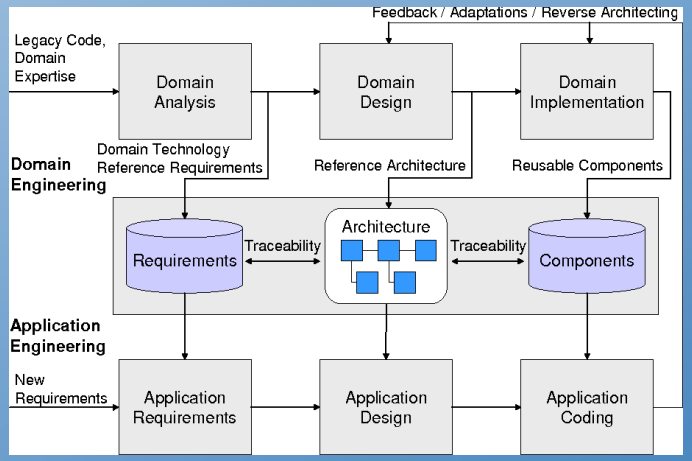


Branching and Maintenance



- Continuous integration obviates branching
  - Everything done in the main branch
  - Everyone works in the main branch
  - Generally fewer and less severe merge conflicts
  - Requires more thought while coding
- Branching might still be needed for maintenance
  - Current stable version of the system for users
  - Current development version of the system
  - Previous user versions of the system
- Security patches need to be made in all of these

Families of Software Systems

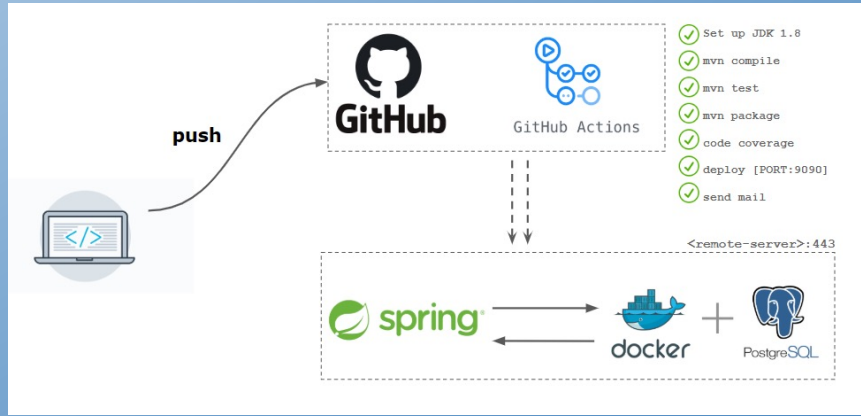


- Set of applications with common set of features
  - All developed at once as one system
  - System can be configured in multiple ways
- Might be related versions of the same system
  - State & Federal tax programs
  - IntelliJ enterprise versus community
  - Different versions of an operating system
- Might be separate systems with a common framework
- This is another approach that is used
  - But should be considered in architecture & design
  - Generally, for larger systems

Workflows and Automation

- Continuous Integration requires automation
  - Building, testing, deploying the system
  - Useful in general
- Workflows are a way of doing this automation
  - Triggering event
  - Actions to be run at that point
- Workflows are more general
  - Can be used for lots of different things
  - On check out; on push; on pull; on reviews
  - Supported by tools

GitHub Actions

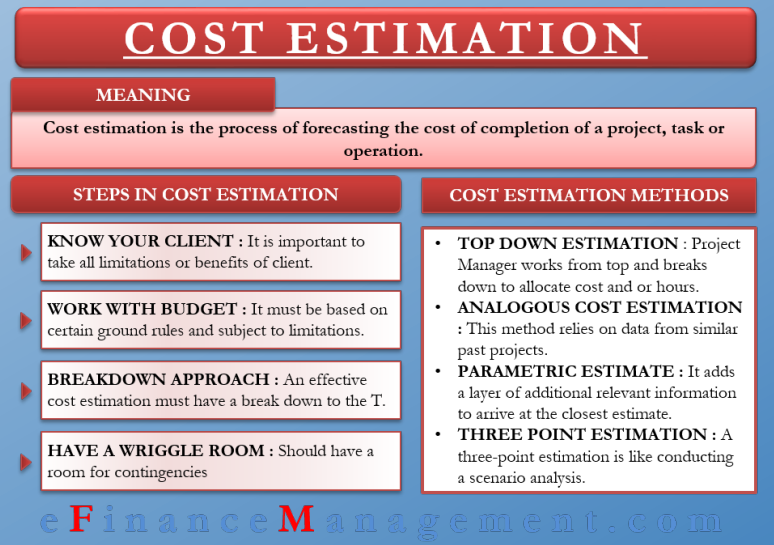


- A general approach to automation
  - Implementation of workflows
  - For continuous integration or otherwise
- Actions have a trigger event
  - Push, pull most common
  - Lots of others available
- Actions have one or more jobs
  - Common jobs (e.g., git actions, email, build & test)
  - Scripts
  - Actions can be conditional, run locally or globally, …

Using GitHub Actions

- Defined in a yaml file in .github/workflows in project directory
  - Useful for team projects
  - But these run in GitHub
- AWS CodePipeline and other alternatives exist
- Git provides a local alternative: hooks
  - Executable scripts in .git/hooks directory
  - pre-commit, prepare-commit-msg, commit-msg, post-commit
  - post-checkout, post-merge (pull), pre-push
- Can also just create your own shell scripts
  - Or ant tasks or make tasks
- You should think about how these could be used in project
  - To simplify your work

Estimating Time and Effort



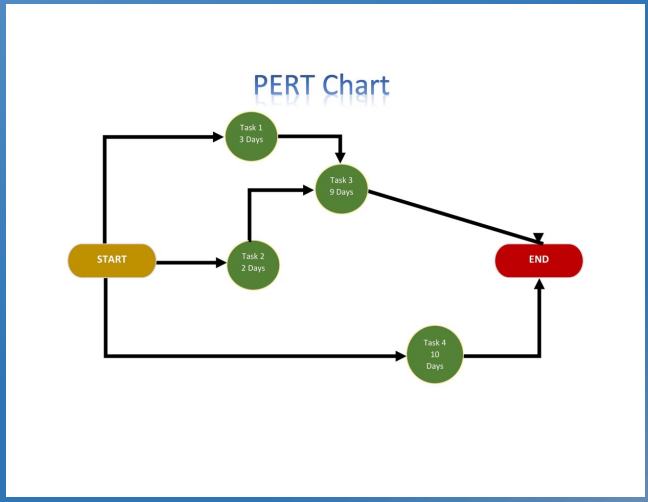
- Required for Team Organization
  - Divide the project up fairly
  - Get critical pieces working early
    - Especially when others are dependent on them
  - Determine what will be done when
  - Determine who will do what
  - You need to know how long it will take to build the software
- Based on
  - Complexity and size of code involved
  - Complexity of interaction with other components
  - Knowledge & abilities of programmer assigned

Estimating Time and Effort

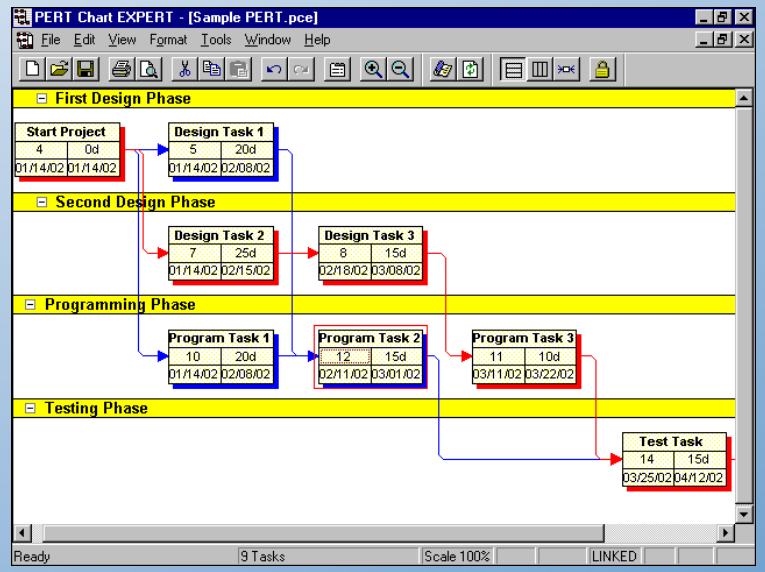
- Known systems – take various factors into account
  - COCOMO model is the best known
  - Expert experience is most widely used
- My Approach
  - Expert estimate based on similar code, size, complexity
  - Multiply by 4 (for myself; different factor if others coding)
- Easier to do for smaller pieces of code
  - Rather than the whole system at once
  - Agile sprints – only estimate the week's work, not the project

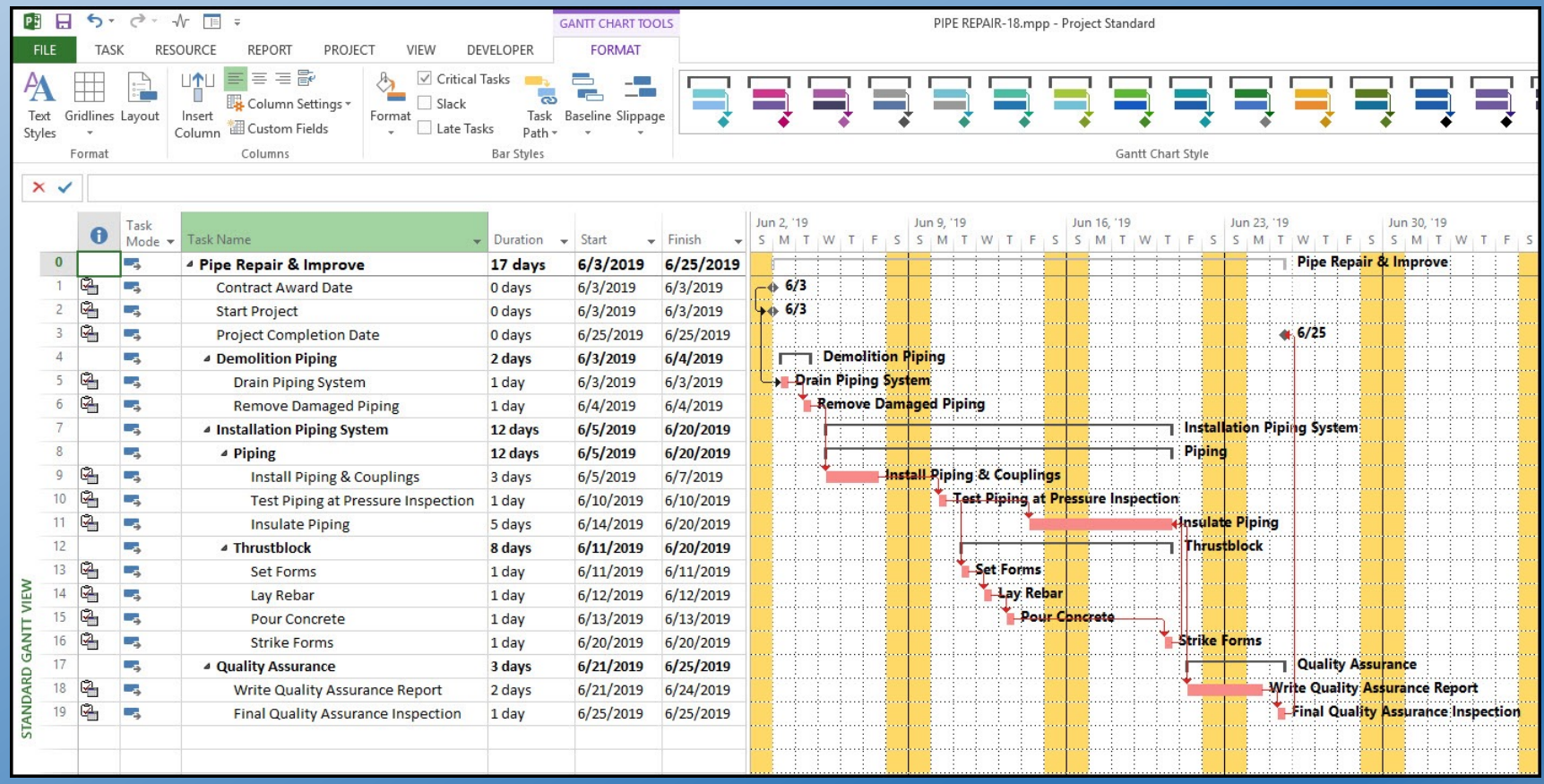
Project Management

- Some pieces of the project are more important than others
  - Required before other pieces can be tested (or even written)
  - Required as a framework for building other pieces
- Project management tries to identify these dependencies
  - Project dependency graph (PERT & GANTT Charts)
- Add time estimates to this graph
  - Identify critical path (what takes the longest)
  - When to start each piece so it gets done in time
  - Who to assign to each piece



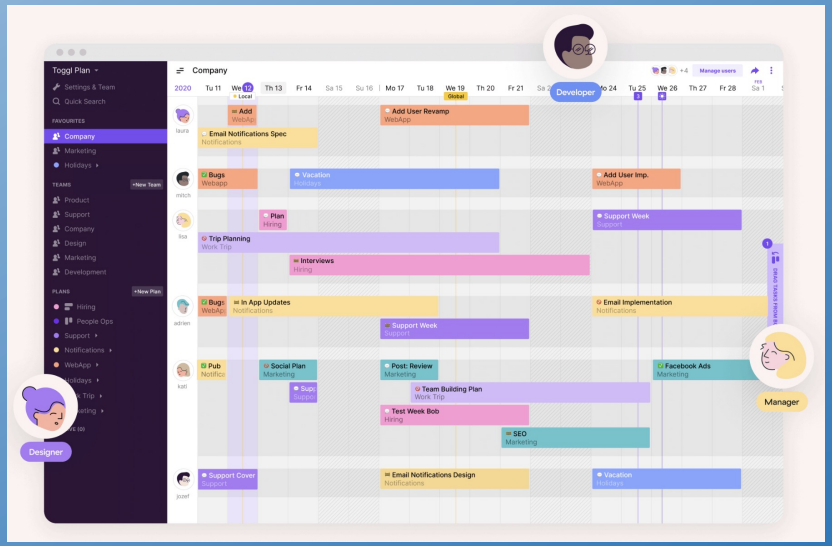
PERT and GANTT Charts

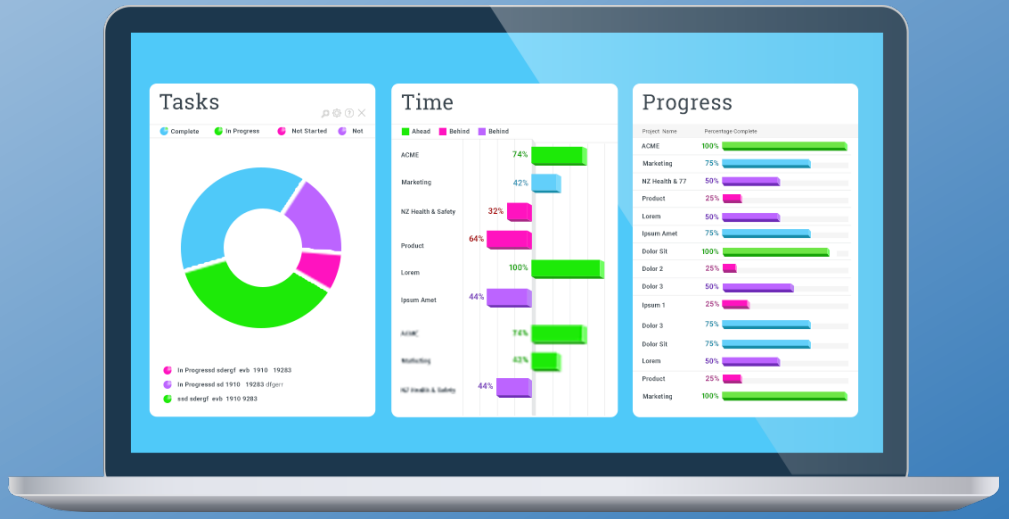




Project Management Software

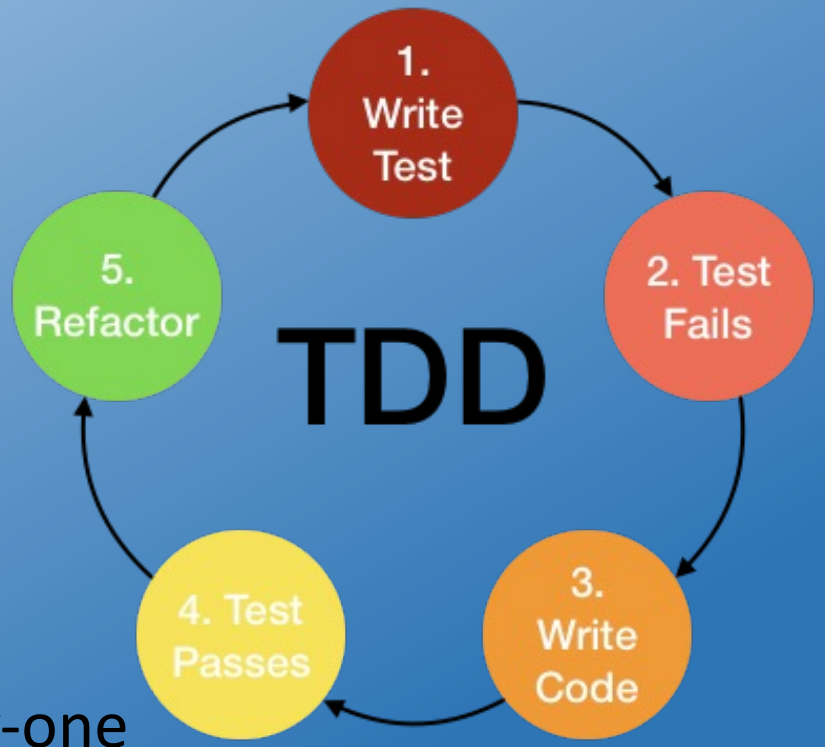
- Microsoft Project Manager
- JIRA
- GitHub issues (gantt chart generator)





Test-Driven Development

- This is emphasized in agile development
  - Used in earlier courses
- Write the test cases first
  - Develop the code to pass the test cases
- Design the code & system so it can be tested
  - This is not as easy as it sounds
  - Especially for UI-based, interactive systems
- Write the user interface first
  - Develop code to handle user interactions one-by-one
  - Simulate the user interactions with test cases
  - Good alternative or addition for UI-heavy applications

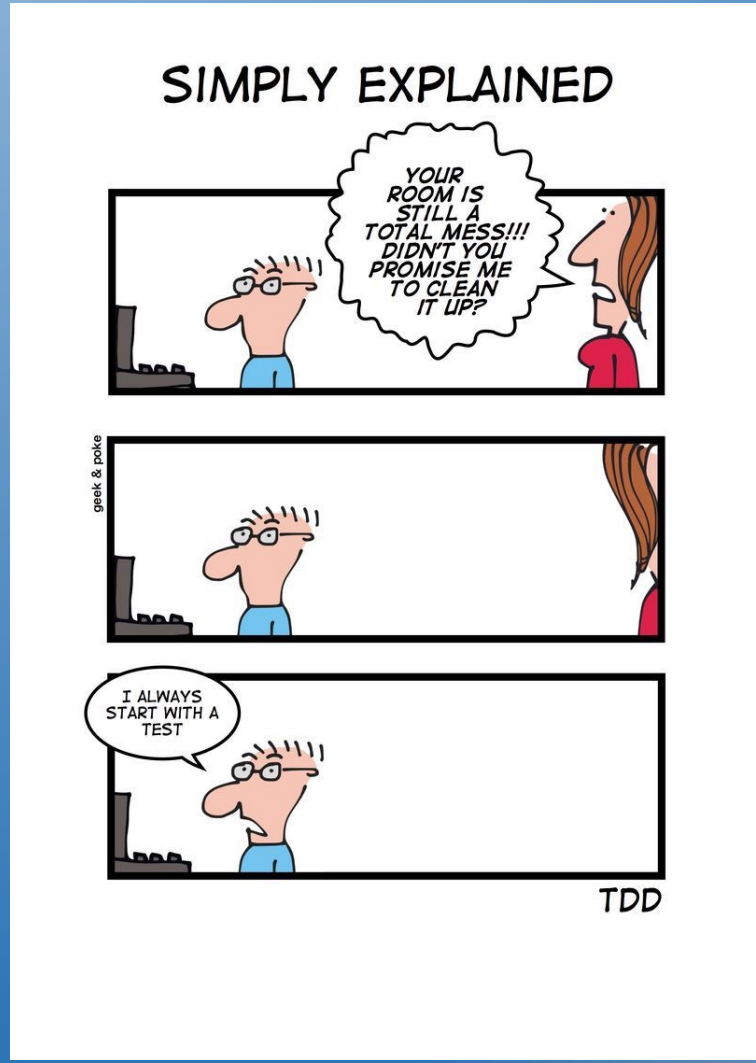


Test-Driven Development Pros/Cons

- Advantages
  - You can see how well you are doing
  - Provides a focus for code development
  - Better understanding of what needs to be written
  - Ensures code works as it is written
- Disadvantages
  - Test cases can be difficult to write, especially interactive/graphical
    - Can be as much work as actually writing the code
  - Might need to write mocking (dummy) libraries, etc.
  - Test cases and testing code can be buggy
  - Evolving code means evolving the test cases as well
  - Can overfit code to test cases, not look forward
  - Files, external systems, etc. get in the way

Test Cases

- Are an essential part of the system
  - Whether written before, during or after coding
  - Needed to avoid regression errors
  - Needed to approve code before general use
- Can clarify design before coding
- Can need as much effort as the code
  - Typically, with more bugs
  - Often not maintained as well
- Alternatives
  - Dogfooding
  - Automated bug reporting
  - Alpha and beta testing



Research in Programming Teams

- Continuous Integration and workflows
- User interface testing
- Design for software families

PROJECT HOMEWORK

- Check out various communication tools
  - SLACK, GitHub Issues, …
  - Choose one or more for your team
    - Set it up, start using it
- Finalize project roles
  - Ensure absent people agree to their role
  - Ensure everyone is “happy”
- Set up an initial PERT/GANTT chart for your project
  - Possibly using GitHub issues
  - Little to put in there now since we haven’t done design
- In addition to deciding on an overall software architecture
  - Based on requirements and specifications
  - Software architecture hand-in due Thursday (one per team)

HOMEWORK / Further Reading

- Code up the bouncing balls assignment
- Make your programming assignment code match the coding style for your project
  - With a personal package name or equivalent
  - With personal class names matching package
  - Modified for different programming language if needed
- Programming assignment should be working by class Thursday

Further Reading

- https://github.com/minhloc2011/books/blob/master/Peopleware%2C%203rd%20Edition.pdf
- https://martinfowler.com/articles/continuousIntegration.html