[CS3704] Software Engineering

Shawal Khalid

Virginia Tech

02/28/2024

Announcements

- **HW2 due next week (3/11) by 11:59pm!**

- **Spring break** 🍂
  - **No class**

Project Management

Project Management

Software Configuration Management

Task Management

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- Use techniques and processes to create and analyze requirements for an application
- Use techniques and processes to design a software system
- Identify processes, methods, and tools related to phases of the SDLC
- Explain the differences between software engineering processes
- Discuss research questions and current topics related to software engineering
- Create and communicate about the requirements and design of a software application

Warm-Up

Discuss: How do you manage the tasks that you need to complete (personal, class, etc.)? What tools or processes do you find helpful?

Project Management

- The process of leading a team to achieve project goals within given constraints.
  - **Constraints:** requirements, scope, time, budget,...
- Activities include planning, scope management, estimation, scheduling, people and resource management, risk management and mitigation, processes, etc.
  - Not just the manager’s job–these activities involve everyone!

Effective Project Management

Effective project management focuses on the 4 P's:

1. **People:** the most important element
   a. recruiting, training, performance management
2. **Product:** the software to build
   a. Project objectives, scope, alternative solutions
3. **Process:** define activities and tasks involved
   a. Milestones, work products, QA points
4. **Project:** progress control
   a. Planning, monitoring, controlling

First Law

“No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.” **[Bersoff 1980]**

- *What are these changes?*
  - Changes in technical requirements
  - Changes in user requirements
  - Project plan
  - Code
  - Data
  - …

Version Control

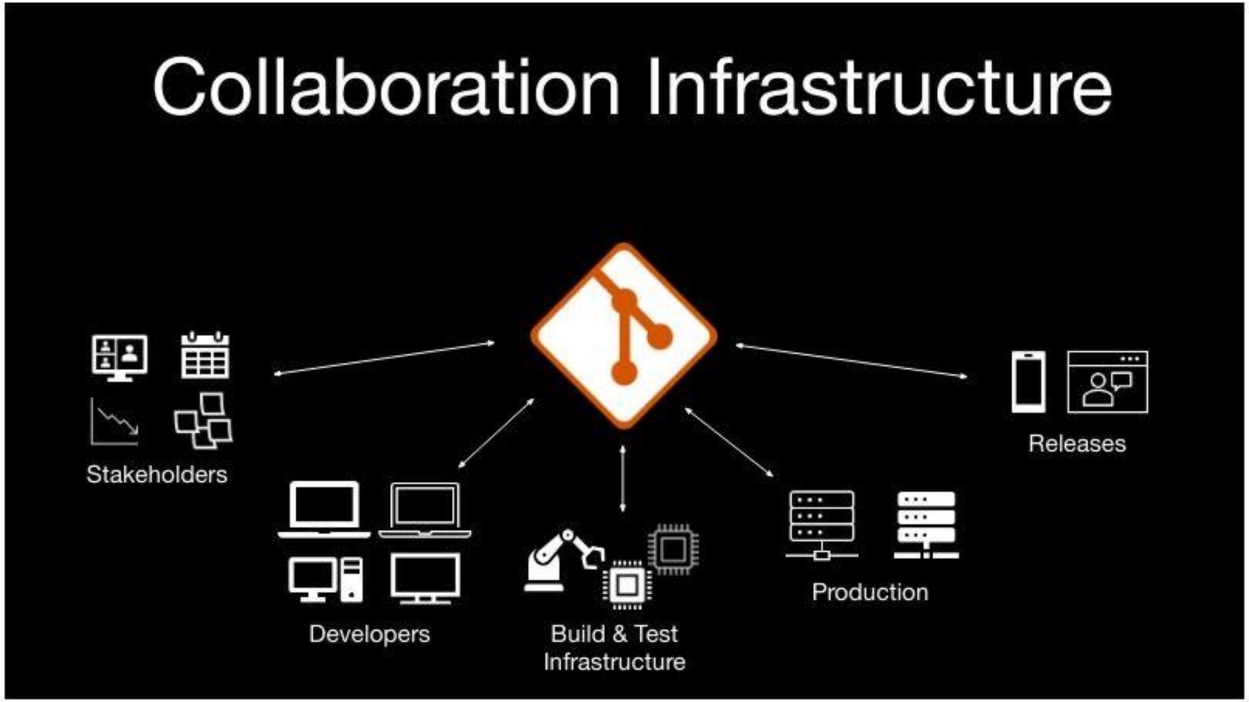
Version Control systems *manage changes* to documents, programs, websites, and other collections of information.

– Ex) Git (*i.e.*, GitHub, GitLab, etc.), Subversion, Mercurial

Different implementations:

- Subversion: One SVN server can hold many repositories, one repository can hold many projects
- Git: Distributed version control, everyone has their own local version control repository

Version Control (cont.)



Collaboration Infrastructure

Change Management

What Do We Mean by “Manage Changes”?

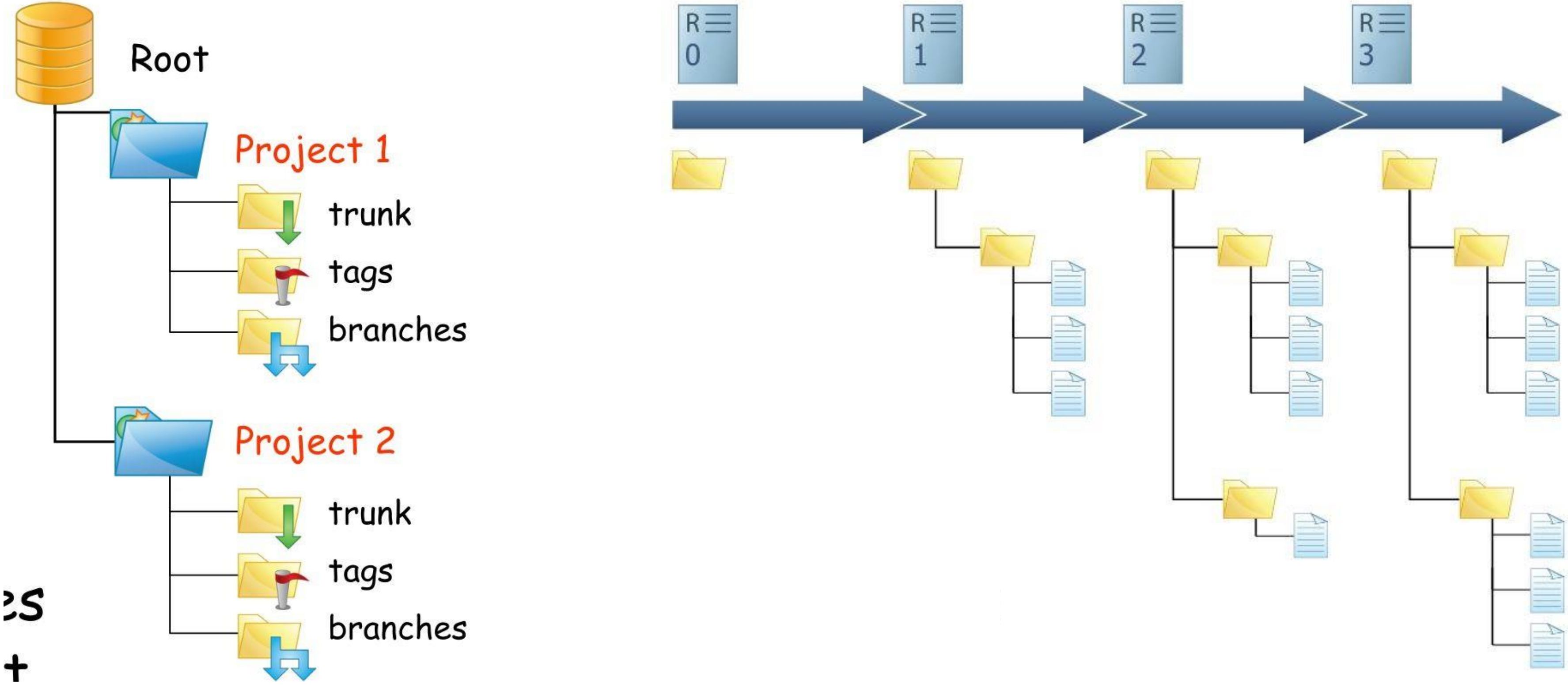
- What changes have been made?
- Why are the changes made?
- Who makes the changes?
- Can we redo/undo some changes?
- Can we branch the project?

Software Configuration Management

The task of tracking and controlling changes in software.

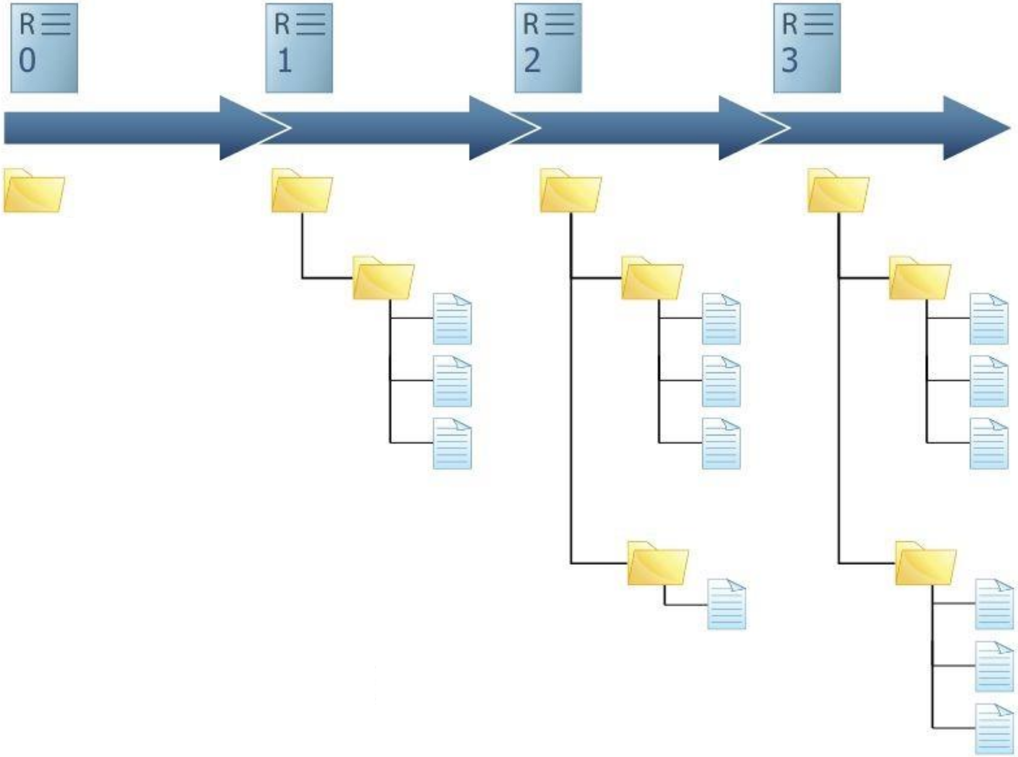
- Repository: tools that allow developers to effectively manage changes
    - Version control systems
    - Issue tracking systems

Repository Layout



Repository Layout (cont.)

- Each project has multiple revisions
  - Each revision is assigned a name
  - Revision number is incremented for every commit transaction
  - Delta (diff) information is recorded



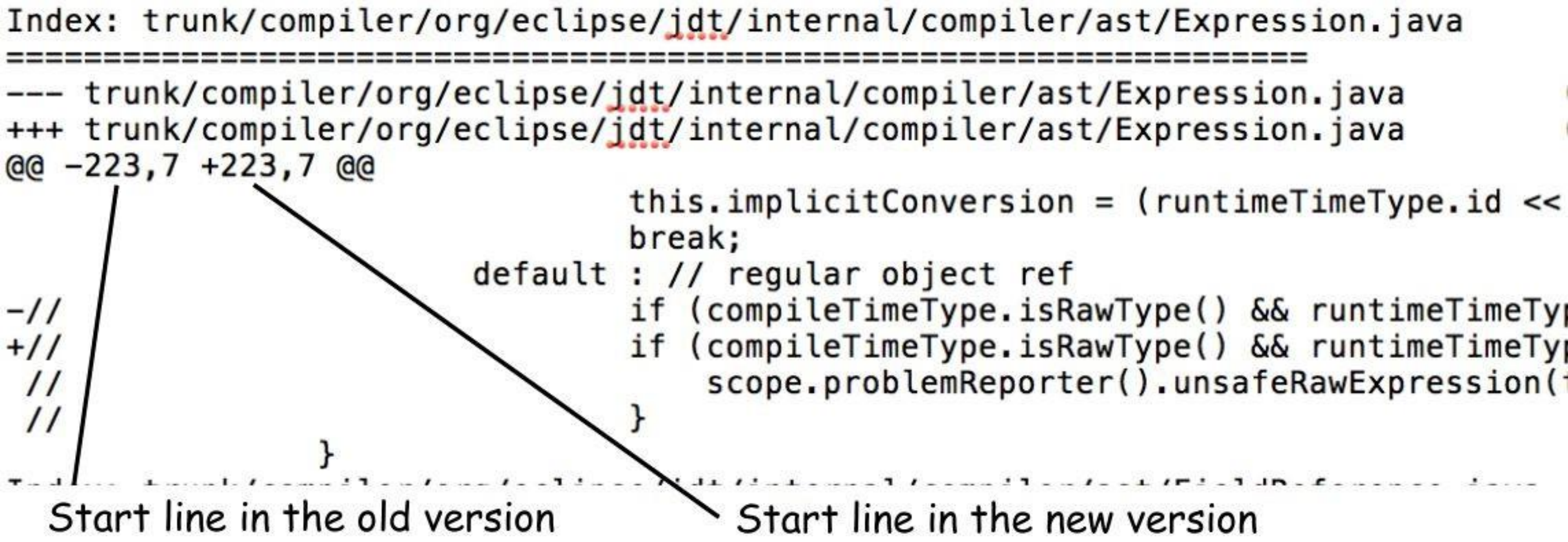
Basic Features of a Repository

- Keep the history of all changes to files and directories
  - You can add in new versions
  - You can recover any previous version
- Access control
  - Read/write permission for users
- Logging
  - Author, date, and reason for a change
- Additional features
  - *Diff, Branches, Merging*

Diff

- **Goal:** To display the differences between two revisions
  - *What has been changed?*
  - Add or delete a line of text
  - No update, or move
- A lot of features are based on diff
  - Save new versions
  - Recover a prior version
  - Patch

Diff Example



```
Index: trunk/compiler/org/eclipse/jdt/internal/compiler/ast/Expression.java
===================================================================
--- trunk/compiler/org/eclipse/jdt/internal/compiler/ast/Expression.java
+++ trunk/compiler/org/eclipse/jdt/internal/compiler/ast/Expression.java
@@ -223,7 +223,7 @@
                               this.implicitConversion = (runtimeTimeType.id <<
                               break;
                       default : // regular object ref
-//                            if (compileTimeType.isRawType() && runtimeTimeTy
+//                            if (compileTimeType.isRawType() && runtimeTimeTy
 //                                scope.problemReporter().unsafeRawExpression(
 //                            }
                }
Index: trunk/compiler/org/eclipse/jdt/internal/compiler/ast/FieldReference.java
```

Start line in the old version

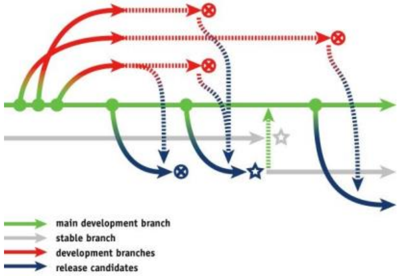
Start line in the new version

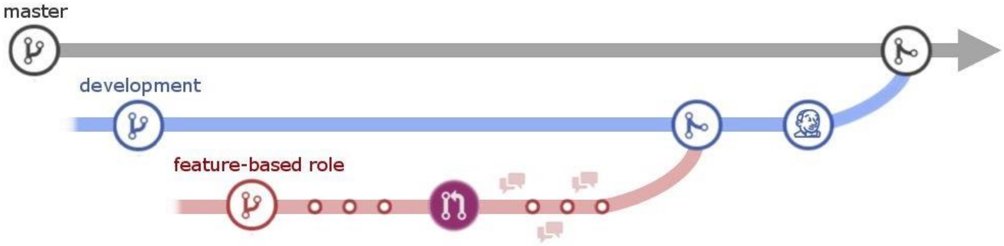
- “+”: added lines, “-”: deleted lines
- Some unchanged lines are shown to indicate program context

Branching

A ***branch*** is a mechanism for allowing concurrent changes to be made to a source repository.

- Empirical research shows branching is effective, but developers create ineffective branches. [Bird]





Branching (cont.)

Reasons to Branch:

- Separate concerns among teams/developers
- Tentative new features
- Parallel version history without interference
- Different releases
- Fix bugs
- …

Branching Anti-patterns

Merge-a-phobia — avoiding merging at all cost, usually because of a fear of the consequences.

Merge Mania — spending too much time merging code instead of developing it.

Big Bang Merge — deferring branch merging and attempting to merge all branches simultaneously.

Never-Ending Merge — continuous merging activity because there is always more to merge.

Branch-a-holic — creating too many branches.

Cascading Branches — branching but never merging back to the main line.

Volatile Branches — branching with unstable files merged into other branches.

Development Freeze — stopping all development activities while branching and merging.

Integration Wall — using branches to divide the development team members, instead of dividing work.

Spaghetti Branching — integrating changes between unrelated branches.

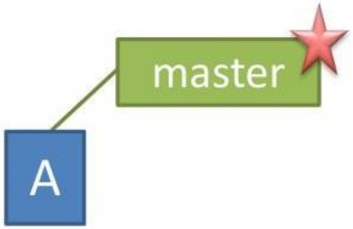
[Appleton]

Merging

After making code changes to fix a major bug on a separate branch, you need to apply similar changes to the main branch. What do you do?

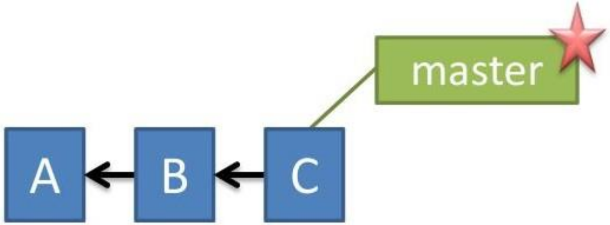
- Merge!

Example



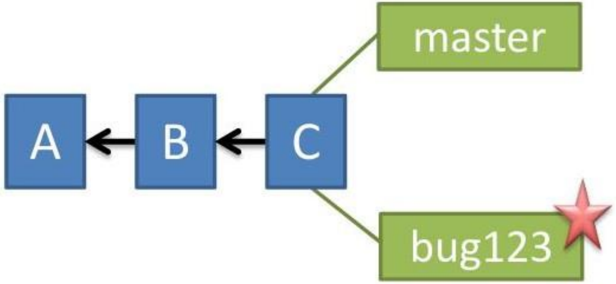
```
> git commit –m ‘my first commit’
```

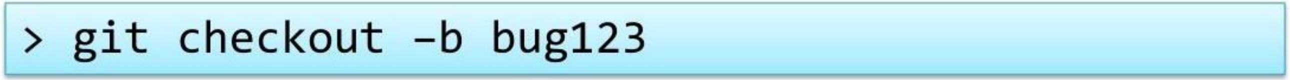
Example



```
> git commit (x2)
```

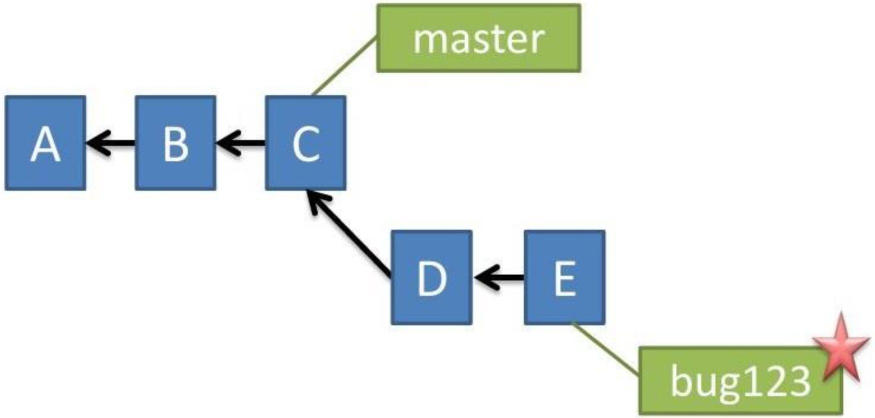
Example





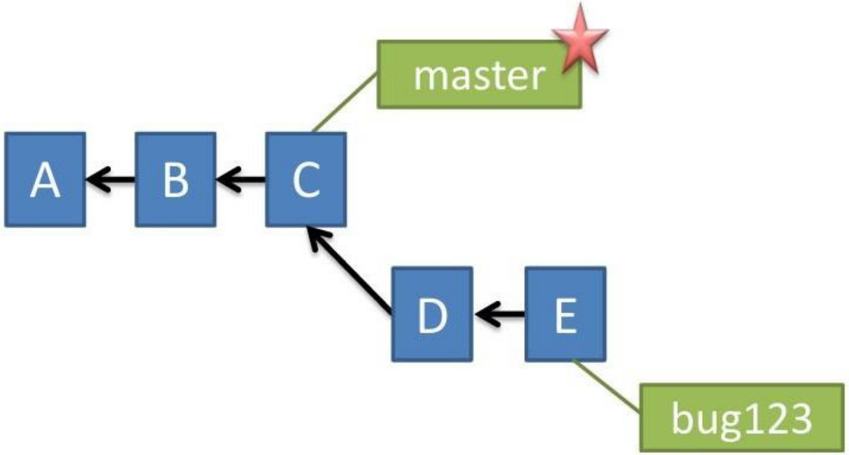
```
> git checkout -b bug123
```

Example



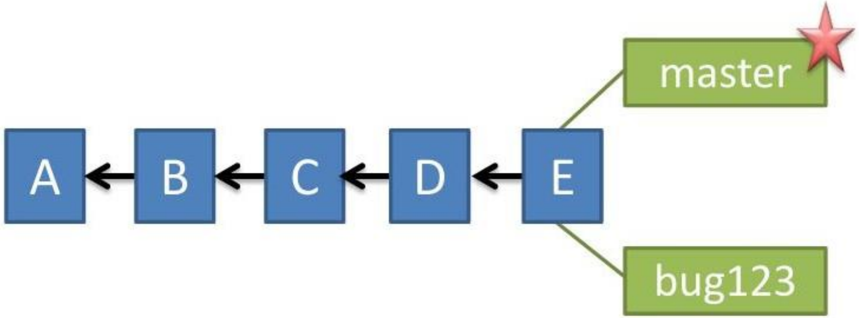
```
> git commit (x2)
```

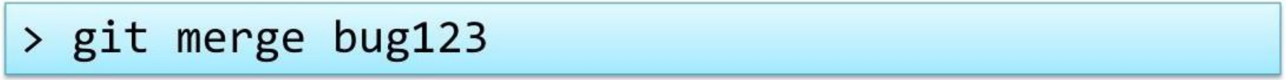
Example



```
> git checkout master
```

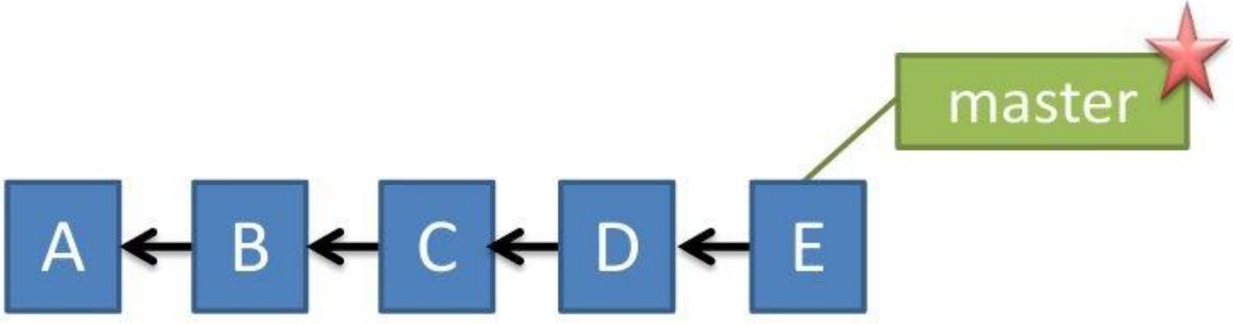
Example





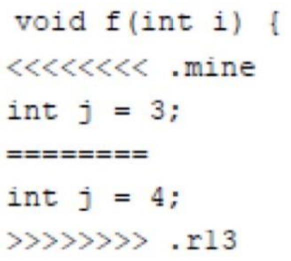
```
> git merge bug123
```

Example



Merge Conflicts

- When two developers edit the same file
  - Can be resolved manually or automatically



```
 void f(int i) {
<<<<<<<< .mine
int j = 3;
========
int j = 4;
>>>>>>>> .r13
```

- Tips:
  - Small commits
  - Commit often!
  - Write meaningful commit messages
  - Avoid commit noise

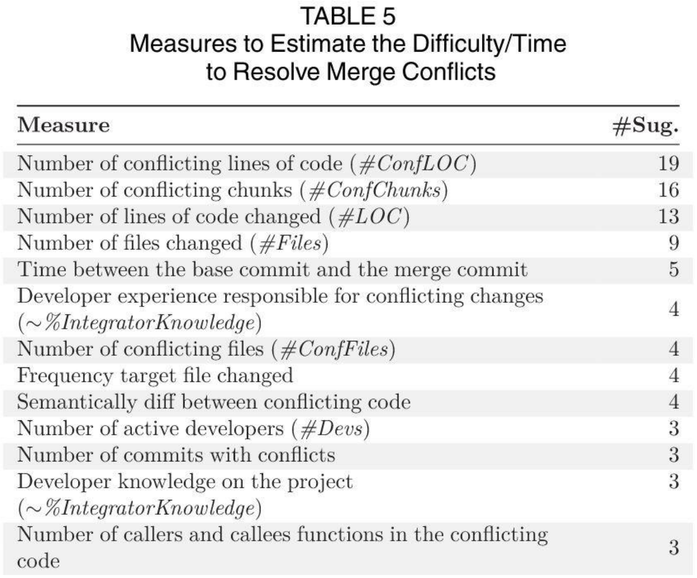


TABLE 5
Measures to Estimate the Difficulty/Time to Resolve Merge Conflicts

Measure	#Sug.
Number of conflicting lines of code (*#ConfLOC*)	19
Number of conflicting chunks (*#ConfChunks*)	16
Number of lines of code changed (*#LOC*)	13
Number of files changed (*#Files*)	9
Time between the base commit and the merge commit	5
Developer experience responsible for conflicting changes (∼*%IntegratorKnowledge*)	4
Number of conflicting files (*#ConfFiles*)	4
Frequency target file changed	4
Semantically diff between conflicting code	4
Number of active developers (*#Devs*)	3
Number of commits with conflicts	3
Developer knowledge on the project (∼*%IntegratorKnowledge*)	3
Number of callers and callees functions in the conflicting code	3

Issue Tracking Systems

- Manage and maintain list of issues as needed by the organization.
  - Create, update, and resolve reported issues by customers and developers.
- An ***issue*** is a unit of work to accomplish an improvement to the system. This includes
  - – a bug
  - – a requested feature
  - – a patch
  - – missing documentation, …

Issue Tracking (cont.)

Why do we need issue tracking systems?

- Developers need to communicate while making changes to address issues.
- Basic features:
  - Description
  - Status (Tracking ability)
  - Unique ID/title
- Prior approaches:
  1. Mailing Lists (hard to manage, unorganized)
  2. Forums (no tracking code and status)

Bugzilla

Jira

Issue Resolution

- Fixed

– A bug is fixed, a feature is added, a patch is applied

- Invalid

– Bug cannot be reproduced, features do not make sense, patch is not correct

- Duplicate

– It is a duplicate of an existing issue

– Get merged with the other issue

- Won’t fix

– The developers decide not to fix the bug or accommodate the new feature (Limited human resources, lack of essential information to reproduce a bug, lack of expertise,...)

Review: What are requirements?

Definition: Capabilities and conditions to which the system — and more broadly, the project — must conform. [Larman]

- Focusing on the **WHAT** *not* the **HOW**
- **Should always come *first* in the SDLC**

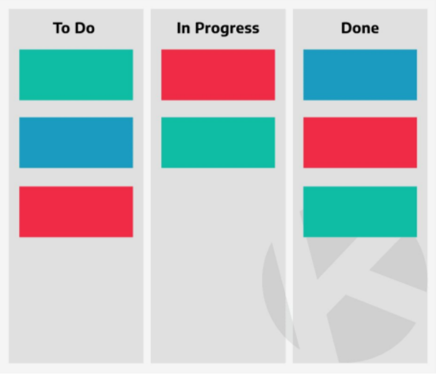
Requirements vs. Tasks

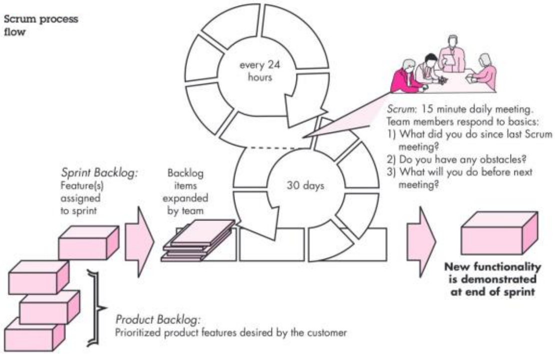
- **Requirements:** What the *system* should do

- **Tasks:** What *you* should do
  - Includes implementing the system functionality based on the requirements, and much more!

Task Management

- We’ve already seen several useful practices for task management, including:
  - Kanban and Scrum




Scrum process flow
every 24 hours
Scrum: 15 minute daily meeting. Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
30 days
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

TODO: Another scrum meeting!

- What I did.
- What I need to do next.
- What is blocking me.

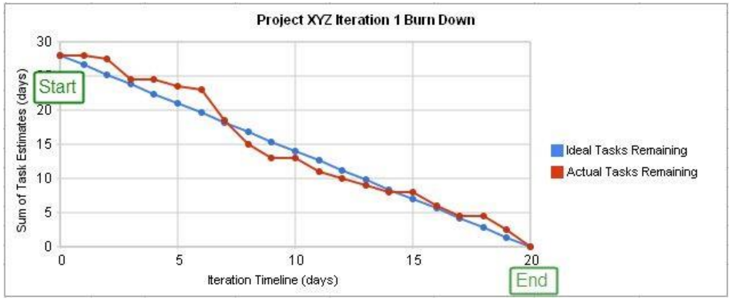
Process Metrics

Evaluating how the team is doing, developer productivity.

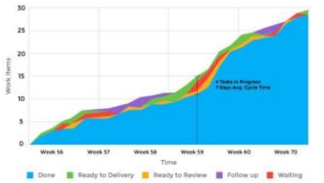
Velocity: units of work completed per iteration

Fault-slippage: number of bugs in production

Burndown: work done vs. outstanding work over time



Process Metrics (cont.)



Cumulative Flow: measures consistency of workflow

Lead time: amount of time between request and product delivery

Work Item Age: time to complete task

Blocked Time: time that a task is blocked

Value delivered/Throughput: Assign a value for every requirement (function points, $, etc.)

Project Meetings

- Remember, More meetings = Bad workdays
  - But, meetings are unavoidable
- Agile development meetings:
  - Stand-ups ✅
    - Daily ~15 min. meetings to provide updates on tasks
  - Sprint planning meetings
  - Triages
  - Sprint review meetings
  - Retrospectives

➔ Will differ based on company, development team, etc.

Does there need to be a meeting?

- To *collaborate*? That's a different kind of gathering…
- To *inform*? Only if you are expecting questions
- To *consult*? Only if people get a vote
  - Otherwise it's just informing with pretense
- To *discuss*? Yes
  - But only in small groups
  - Or with well-defined procedural rules

Meeting Tips

- Create an agenda
  - If you don’t care enough to make a list, you don’t need a meeting
  - Prioritize
  - Time
- Have clear rules for making decisions
  - Every group has a power structure
- Record minutes
  - So people who weren’t there know what happened
  - So people who were there agree what happened
  - So people can be held accountable at later meetings

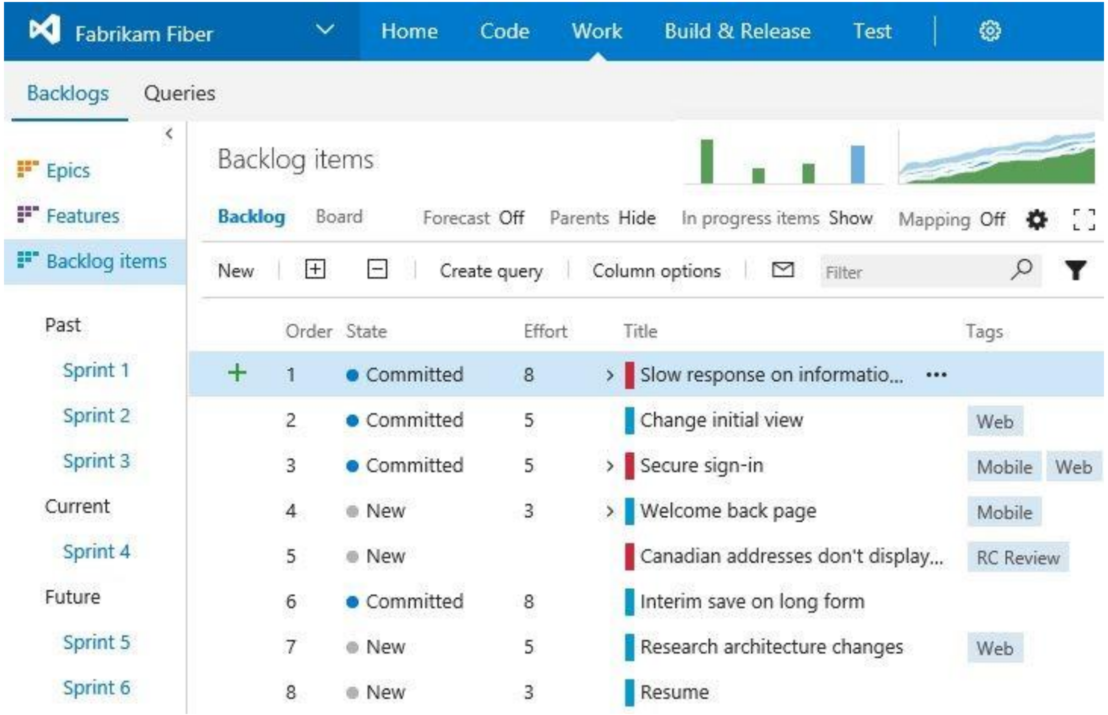
Things that could go wrong

- **Feature creep:** Expansion on the *scope* of a project (i.e. new features) while work is ongoing
  - i.e. adding new tasks to an existing sprint
  - Leads to delays, less stability, failures, etc.
- **Technical debt:** Work that piles up due to earlier design compromises, changing circumstances, bugs,...
- *Team dynamics*
  - **Team contract:** written agreement between team with expectations on how to operate
- *Failures*
  - problems with the application, unmet test cases, etc.

Sprint Planning

- Meeting to review backlog and determine tasks to be completed in the upcoming sprint
  - Entire team (Scrum master, manager, developers, etc.)
- Divide work between team members
  - How do you divide work between software engineers?[Wilson]
    - **Feature decomposition**- by feature aspects
    - **Modular decomposition**- by module (i.e. parts of code)
    - **Functional decomposition**- by set of tasks or skill
    - **Rotating decomposition**- by function, but swapping periodically
    - **Chaotic decomposition**- random
- Project Estimation

Sprint Planning: Backlog



Backlog items

Forecast Off | Parents Hide | In progress items Show | Mapping Off

Order	State	Effort	Title	Tags
1	● Committed	8	Slow response on informatio...	
2	● Committed	5	Change initial view	Web
3	● Committed	5	Secure sign-in	Mobile Web
4	● New	3	Welcome back page	Mobile
5	● New		Canadian addresses don't display...	RC Review
6	● Committed	8	Interim save on long form	
7	● New	5	Research architecture changes	Web
8	● New	3	Resume	

https://www.visualstudio.com/en-us/docs/work/backlogs/create-your-backlog

Sprint Planning: Project Estimation

- Software projects require…
  - Size estimation
  - Effort estimation
  - Time estimation
  - Cost estimation
- But, humans are mostly bad at estimation…
- **Estimation techniques**
  - Constructive Cost Model (COCOMO) [Boehm]
  - Planning Poker 🂡
    - *Function points:* estimated measure of time, effort, etc. to complete a work task

Ex) How many miles are between the Earth and the moon?

238,900 mi 🌎 🌘

Triage Meetings

- Review of bugs and defects
  - Analyze reproducibility
  - Prioritize fixes
  - NOT coming up with a solution
- Depending on team, triages can be regularly scheduled, integrated with sprint planning meetings, or only held in case of emergency.

- Medical usage: the prioritization of patient care based on the urgency and severity of wounds, illness, etc.

Sprint Review Meeting

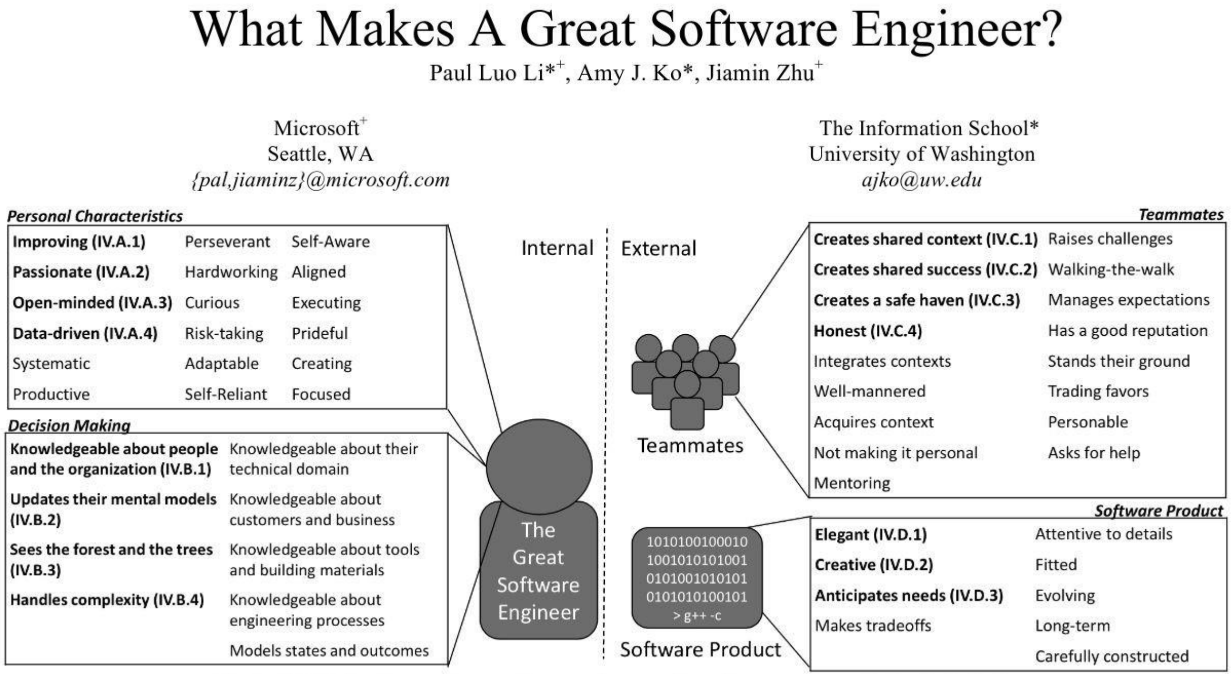
- Development team presents work that was completed during the sprint.
    - Often includes a demo of completed features, tasks
    - Collect immediate feedback from stakeholders
- **Anti-patterns** [Wolpers]
    - Powerpoint presentation
    - No clients, stakeholders, or customers in attendance
    - Requirements analysis and specification
    - *Wizard of Oz*’ing
    - Extended scrum or planning meeting
    - Retrospective

Retrospectives

- Review of the sprint to reflect on activities and processes, brainstorm for next sprint.
    - Opportunity for Scrum team to inspect itself **over the last sprint only!**
    - Iterative improvement applied to the team

<u>Retrospective questions:</u>

1. **What went well?**
2. **What didn’t go well?**
3. **What should we do differently next time?**



What Makes A Great Software Engineer?

Paul Luo Li*+, Amy J. Ko*, Jiamin Zhu+

Microsoft+
Seattle, WA
{pal,jiaminz}@microsoft.com

The Information School*
University of Washington
ajko@uw.edu

Personal Characteristics

Improving (IV.A.1) Perseverant Self-Aware
Passionate (IV.A.2) Hardworking Aligned
Open-minded (IV.A.3) Curious Executing
Data-driven (IV.A.4) Risk-taking Prideful
Systematic Adaptable Creating
Productive Self-Reliant Focused

Decision Making

Knowledgeable about people and the organization (IV.B.1) Knowledgeable about their technical domain
Updates their mental models (IV.B.2) Knowledgeable about customers and business
Sees the forest and the trees (IV.B.3) Knowledgeable about tools and building materials
Handles complexity (IV.B.4) Knowledgeable about engineering processes
Models states and outcomes

Teammates

Creates shared context (IV.C.1) Raises challenges
Creates shared success (IV.C.2) Walking-the-walk
Creates a safe haven (IV.C.3) Manages expectations
Honest (IV.C.4) Has a good reputation
Integrates contexts Stands their ground
Well-mannered Trading favors
Acquires context Personable
Not making it personal Asks for help
Mentoring

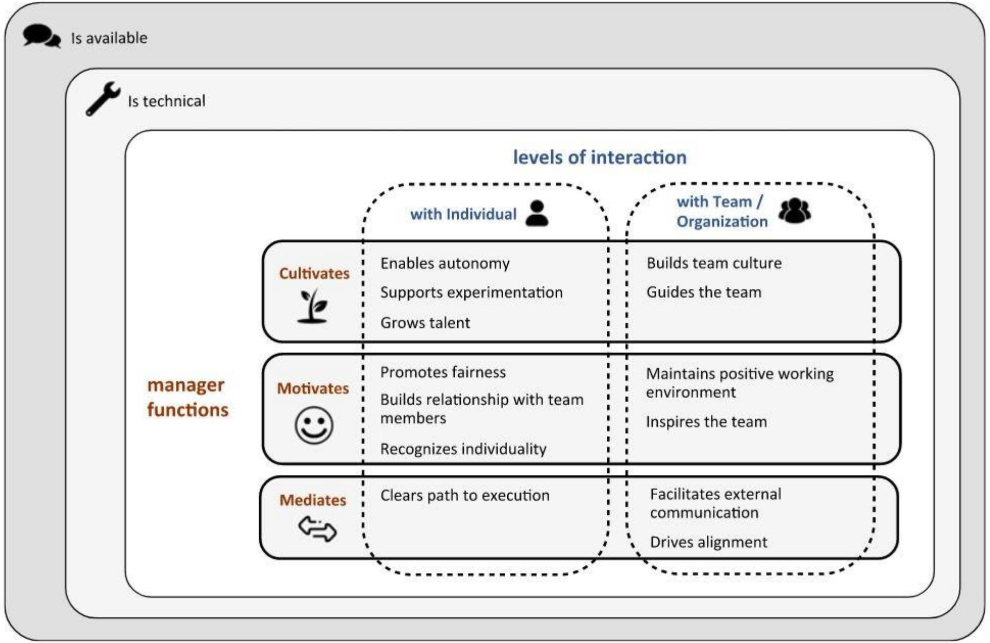
Software Product

Elegant (IV.D.1) Attentive to details
Creative (IV.D.2) Fitted
Anticipates needs (IV.D.3) Evolving
Makes tradeoffs Long-term
Carefully constructed

Managers: *"Walk the walk"*, create an environment and culture to foster these attributes with your development team.

What Makes a Great Manager of Software Engineers?

Eirini Kalliamvakou, *Student Member, IEEE*, Christian Bird, *Member, IEEE*, Thomas Zimmermann, *Member, IEEE*, Andrew Begel, *Member, IEEE*, Robert DeLine, *Member, IEEE*, and Daniel M. German



Is available

Is technical

levels of interaction

manager functions	with Individual	with Team / Organization
Cultivates	Enables autonomy Supports experimentation Grows talent	Builds team culture Guides the team
Motivates	Promotes fairness Builds relationship with team members Recognizes individuality	Maintains positive working environment Inspires the team
Mediates	Clears path to execution	Facilitates external communication Drives alignment

Next Class

- **Discussion Presentation on SE Process**
  - Please sign up for a talk if you haven’t already!
- **Spring break next week**
  - No class!
- **HW2 due (3/11) by 11:59pm**

References

- Dr. Chris Brown, Na Meng, Barbara Ryder. Virginia Tech
- Sarah Heckman, Chris Parnin. NC State
- Greg Wilson. “*Building Software Together*”
- Maggie Norby Adams. “*A brief overview of planning poker*”. 2021
- Paul Li, et al. “What Makes a Great Software Engineer?”. 2015
- Eirini Kalliamvakou, et al. “What Makes a Great Manager of Software Engineers?”