cs304
Software Engineering

TAN, Shin Hwei

陈馨慧

Southern University of Science and Technology

Slides adapted from cs427 (UIUC) and cs304(SUSTech)

Administrative Info

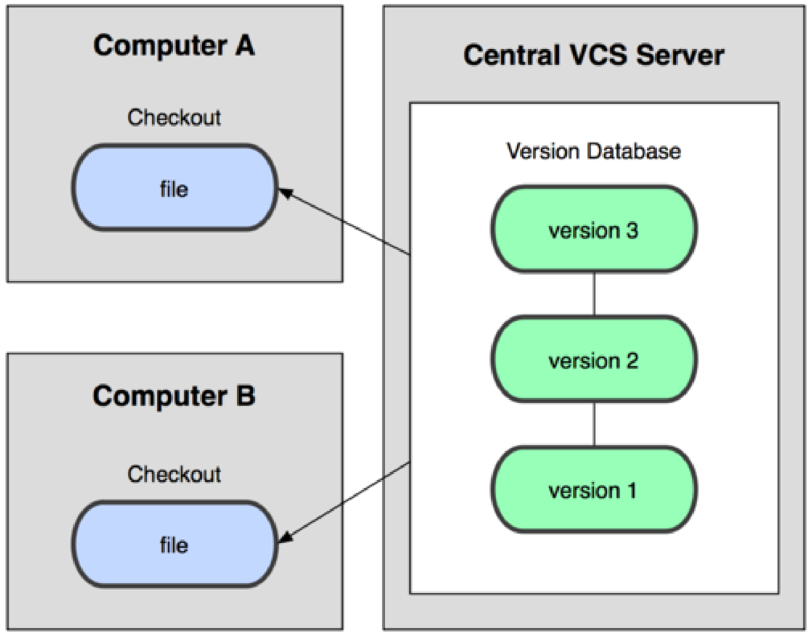
- **MP0 uploaded**, due 7 March 2022 at 11.59pm
  - Simple assignment
  - Should be able to finish in 1-2 hours within lab today
- **Project Proposal uploaded**, due 11 March 2022 at 11.59pm
  - Included frequently asked questions and corresponding answers

    Question: Can we have a repository for the class project where we can use for the entire semester?

    Answer: Yes, we provide a main repository for you to commit your code for the project for the entire semester. There is no deadline for this repository so you can use it for the entire semester even after finishing the class

Recap: SVN vs. Git

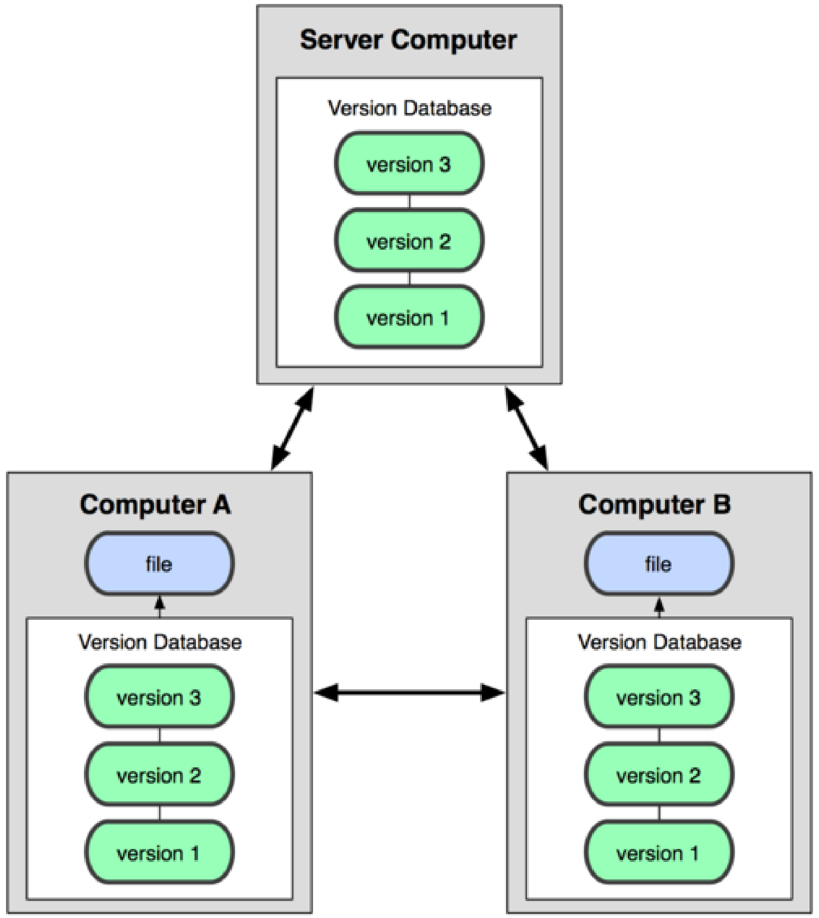
Centralized Model



(CVS, Subversion, Perforce)

Result: Many operations are local

Distributed Model



Result: Many operations are local
(Git, Mercurial)

Continue from previous week

Status accounting

- Reporting the status of components and change requests
  - Which components have changed this week?
  - Which components did Bob change?
  - Which components have the most changes?
  - Which change requests are more than a month old and of priority 3 or greater?

Audit and review

- How do we know that the build script is OK?
- How do we know that only authorized people can change the database interface?
- Can we actually run the version from September 8, 2015?

Change control

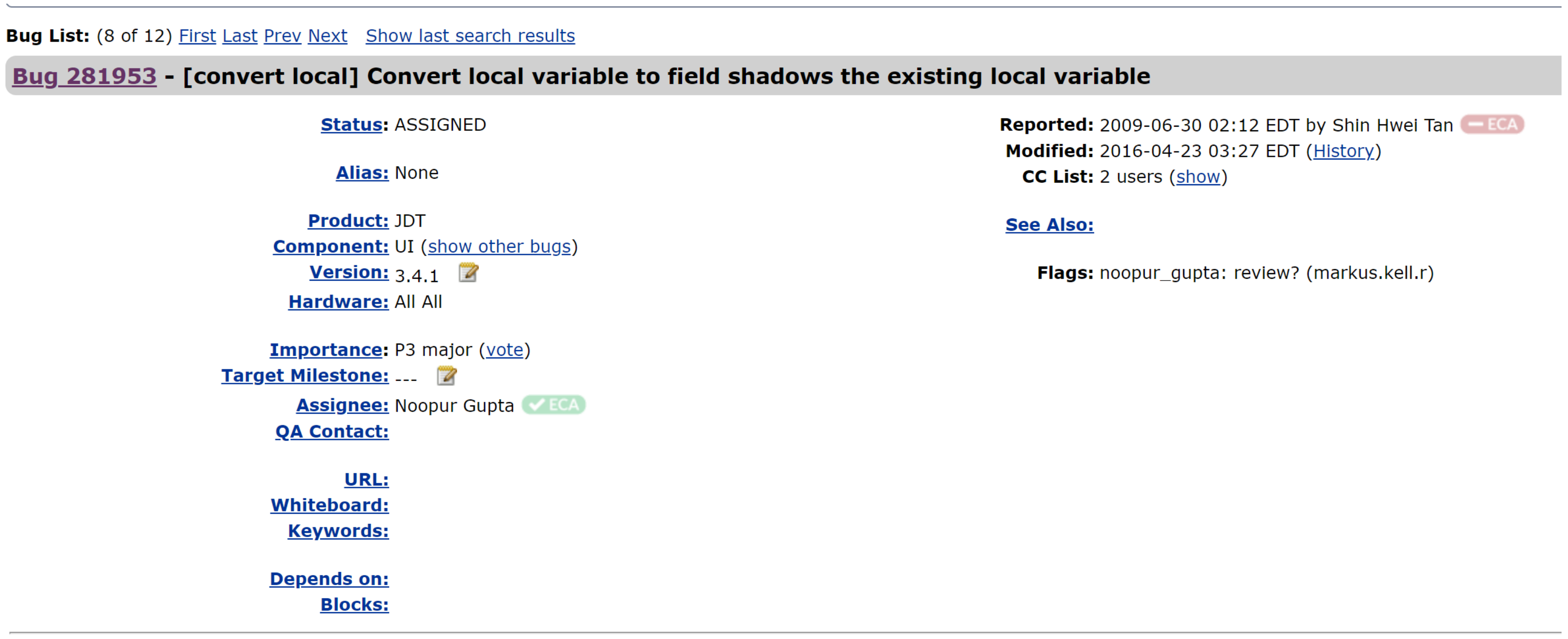
- Change request/engineering change order
  - New feature
  - Bug report
- Change control authority - decides which changes should be carried out
- Should link code changes to change requests

Example: Bugzilla

- Originally developed for Mozilla
  http://bugzilla.mozilla.org
  https://developer.mozilla.org/en/Bug_writing_guidelines
- Bugzilla is a database for bugs. It lets people report bugs and assigns these bugs to the appropriate developers. Developers can use Bugzilla to keep a to-do list as well as to prioritize, schedule and track dependencies

Bugzilla

- Each bug report has ID, name of reporter, description of bug, component, developer, dependency, attachments
- Status: unconfirmed, new, resolved, verified, closed
- Severity: blocker, critical, major, normal, minor, trivial, enhancement



Bug List: (8 of 12) First Last Prev Next   Show last search results

Bug 281953 - [convert local] Convert local variable to field shadows the existing local variable

Status:	ASSIGNED	**Reported:**	2009-06-30 02:12 EDT by Shin Hwei Tan ECA
		Modified:	2016-04-23 03:27 EDT (History)
		CC List:	2 users (show)
Alias:	None		
Product:	JDT	**See Also:**	
Component:	UI (show other bugs)		
Version:	3.4.1	**Flags:**	noopur_gupta: review? (markus.kell.r)
Hardware:	All All		
Importance:	P3 major (vote)		
Target Milestone:	---		
Assignee:	Noopur Gupta ✓ ECA		
QA Contact:			
URL:			
Whiteboard:			
Keywords:			
Depends on:			
Blocks:			

Resolving a bug

FIXED:

INVALID: not a bug, or not a bug in Mozilla

WONTFIX : a bug that will never be fixed

LATER: a bug that won’t be fixed now

REMIND: maybe now, maybe later

DUPLICATE

WORKSFORME

Topics to ponder

- What is the dynamics of people using a change control system?
  - Developers
  - Testers
  - Users
  - Managers

- What about security issues in Bugzilla?

To make SCM work requires

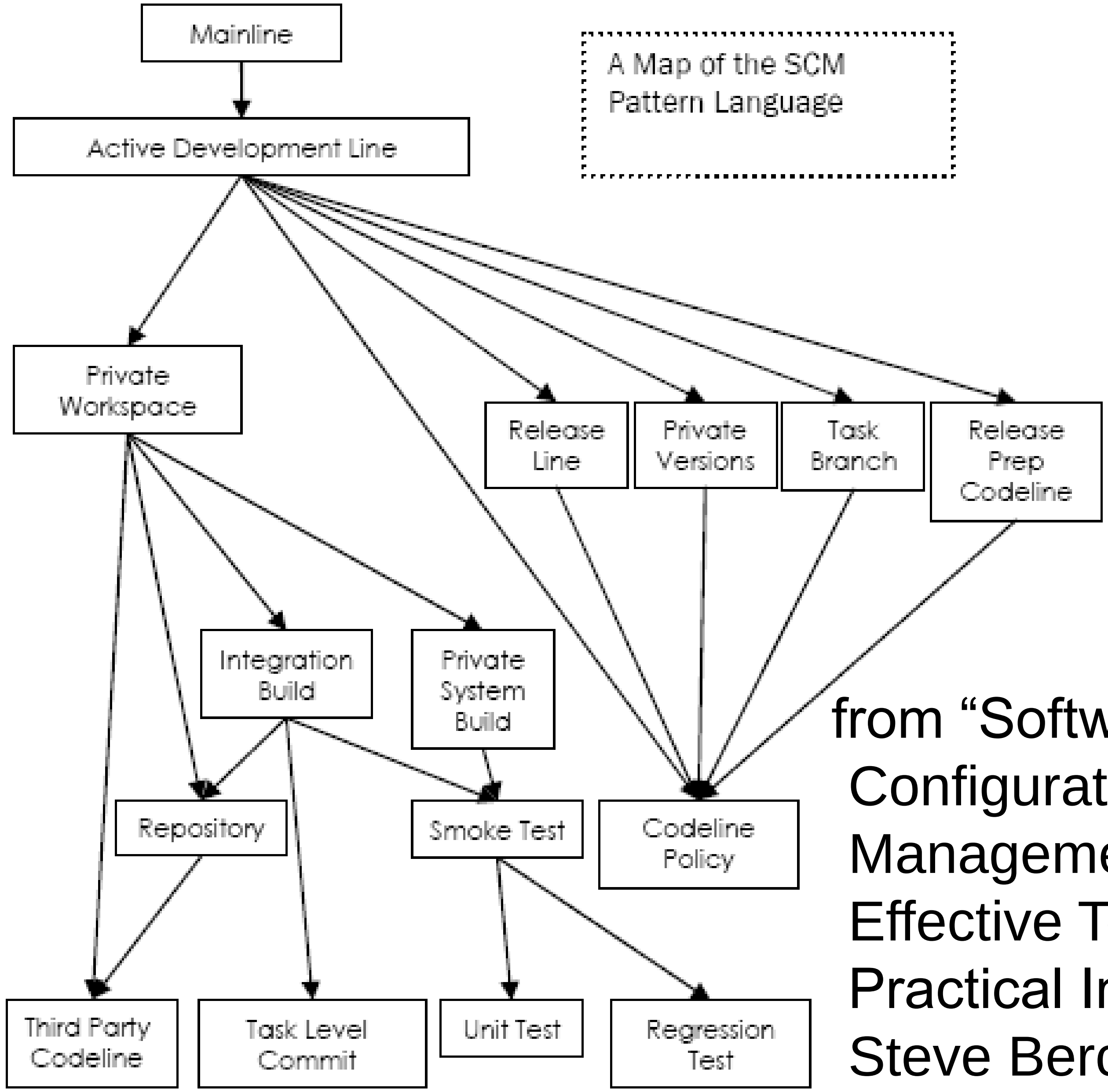
- Bureaucracy
- Discipline/training
- Tools
  - Version control – cvs, subversion, git…
  - Change control – bugzilla, mantis, jira…
  - Building – make, ant, mvn…
  - Releasing – Maven Central and Nexus…

SCM Manager

- Complex tools need expert to manage them
- SCM expert will
  - Maintain tools
  - Maintain configuration files, make branches
  - Do the merging
  - Create policies on version control, change control

Alternatives

- Toolsmith supports SCM tools
- Architect defines configuration files
- Developers merge their code back into mainline
- Managers and technical leads define policies for version control and change control



from “Software Configuration Management Patterns: Effective Teamwork, Practical Integration” by Steve Berczuk with Brad Appleton

Various tests

- Smoke test
  - Ensure that the system still runs after you make a change
- Unit test
  - Ensure that a module is not broken after you make a change
- Regression test
  - Ensure that existing code doesn’t get worse as you make other improvements

Developer issues

- Private Workspace
  - Prevent integration issues from distracting you, and from your changes causing others problems, by developing in a Private Workspace.
- Private System Build
  - Avoid breaking the build by doing a Private System Build before committing changes to the Repository

Codeline Policy

- Active Development Line
- Release Line
  - Holds bug fixes for a release
- Private Versions
- Task Branch
  - Hide a disruptive task from the rest of the team
- Release Prep Codeline

Summary of SCM

- Four aspects
  - Change control
  - Version control
  - Building
  - Releasing
- Supported by tools
- Requires expertise and oversight
- More important on large projects

EXTREME PROGRAMMING

XP
SLIDES FROM CS427

THIS WEEK’S GOALS

- What is traditional waterfall process?
- What is eXtreme Programming (XP)?
- How extremely does XP differ?
- Fundamental XP principles, activities and notions
- When (not) to use XP?

SOFTWARE DEVELOPMENT PROCESSES

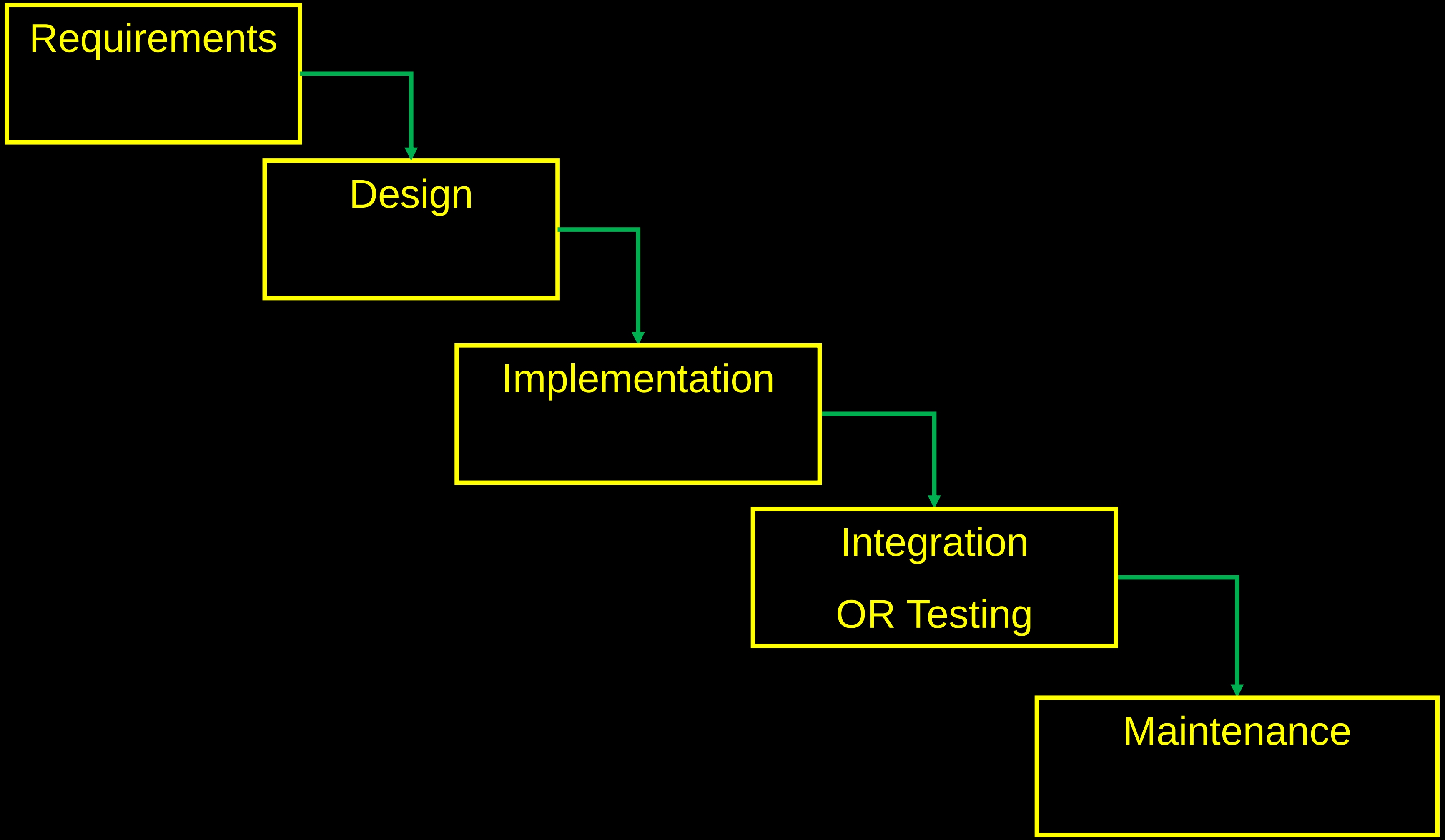
- Many ways to develop software
  - Plan-driven vs. agile
  - Centralized vs. distributed
  - High math vs. low math
  - Close .vs little customer interaction
  - Much testing vs. little testing
  - Organize by architecture vs. features
  - …

WATERFALL PROCESS ACTIVITIES

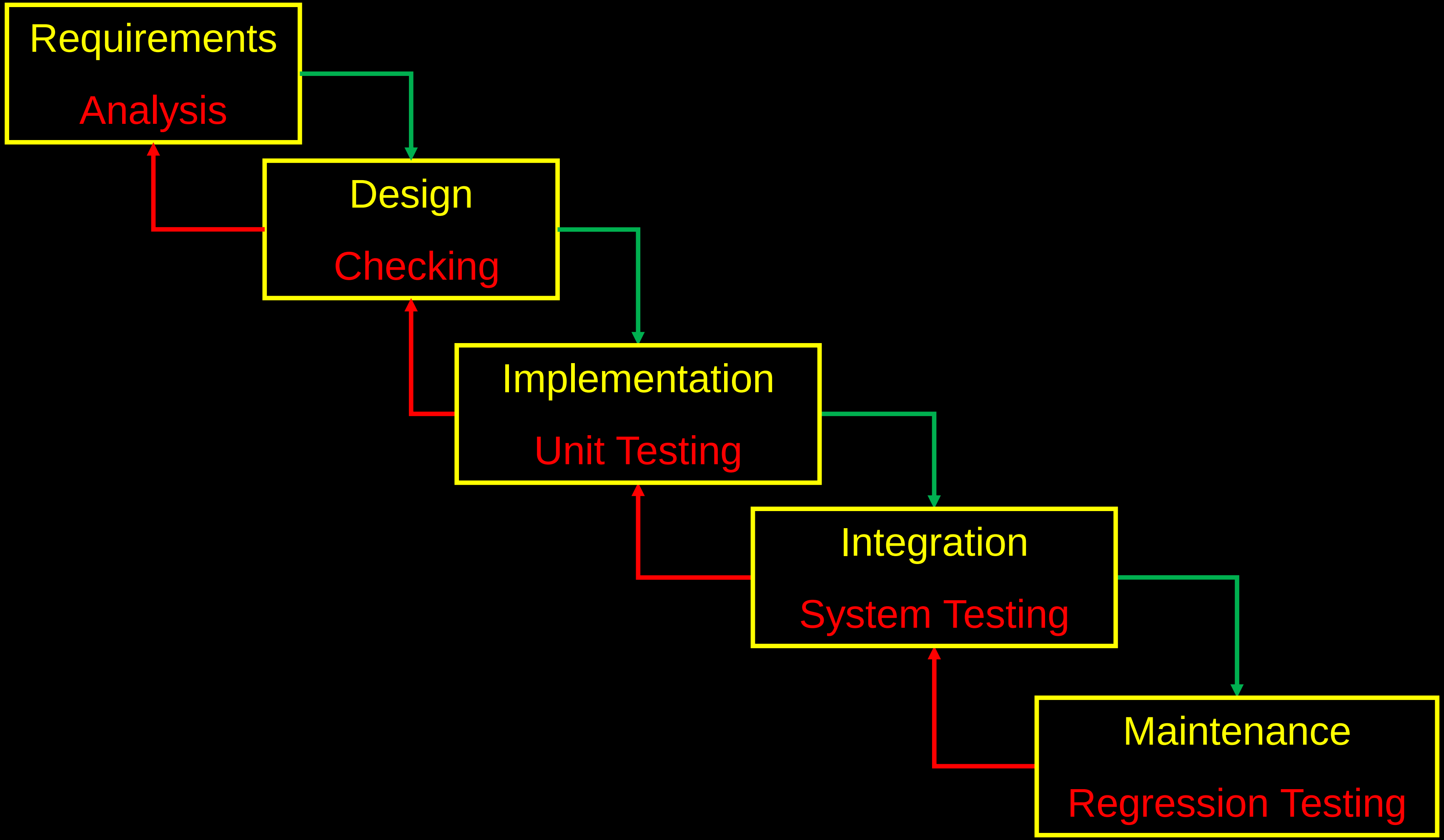
- Requirements – what software should do
- Design – structure code into modules; architecture
- Implementation – hack code
- Integration – put modules together
- Testing – check if code works
- Maintenance – keep making changes

Often merged together as Verification

THEORETICAL WATERFALL MODEL



REAL-LIFE WATERFALL MODEL

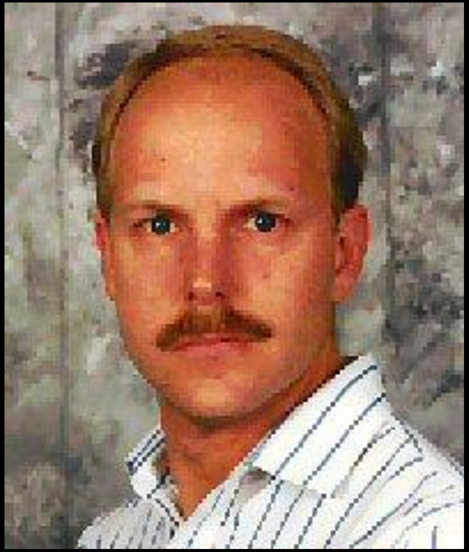


EXTREME PROGRAMMING XP

- Radically different from the rigid waterfall process
  - Replace it with a collaborative and iterative design process
- Main ideas
  - Don't write much documentation
    - Working code and tests are the main written product
  - Implement features one by one
  - Release code frequently
  - Work closely with the customer
  - Communicate a lot with team members

WHO IS THE CREATER OF XP?

CREATOR OF XP
KENT BECK



CODING PEASANTS(码农)?

- Kent beck analogize goat farming and software development.

XP: SOME KEY PRACTICES

- Planning game for requirements
- Test-driven development for design and testing
- Refactoring for design
- Pair programming for development
- Continuous integration for integration

XP IS AN ITERATIVE PROCESS

- Iteration = two week cycle (1-3 weeks)
- Plan each iteration in an iteration meeting that is held at the start of the iteration
- Iteration is going to implement set of user stories
- Divide work into tasks small enough to finish in a day
- Each day, programmers work in pairs to finish tasks

GROUP DISCUSSION: PROS / CONS IN TEAM WORK VS. SOLO WORK

- Requirement: get into a group of three neighbor students
- Go to this link: https://classroom.github.com/a/ZAXgY4km
- Open an issue in the link with title "Pros/Cons in team work vs solo work"
- Share and discuss respective answers based on your current/past team/solo work experiences either at school or outside of school

6 minutes

- I will call for volunteer groups and sometimes randomly pick groups

PAIR PROGRAMMING

Pair programming is a simple, straightforward concept. Two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code, and test. It allows two people to produce a higher quality of code than that produced by the summation of their solitary efforts.

Driver: types or writes

Navigator: observer (looking for tactical & strategic defects)

- Periodically switch roles of driver and navigator
  - Possibly every 30 minutes or less
- Pair coding, design, debugging, testing, etc.

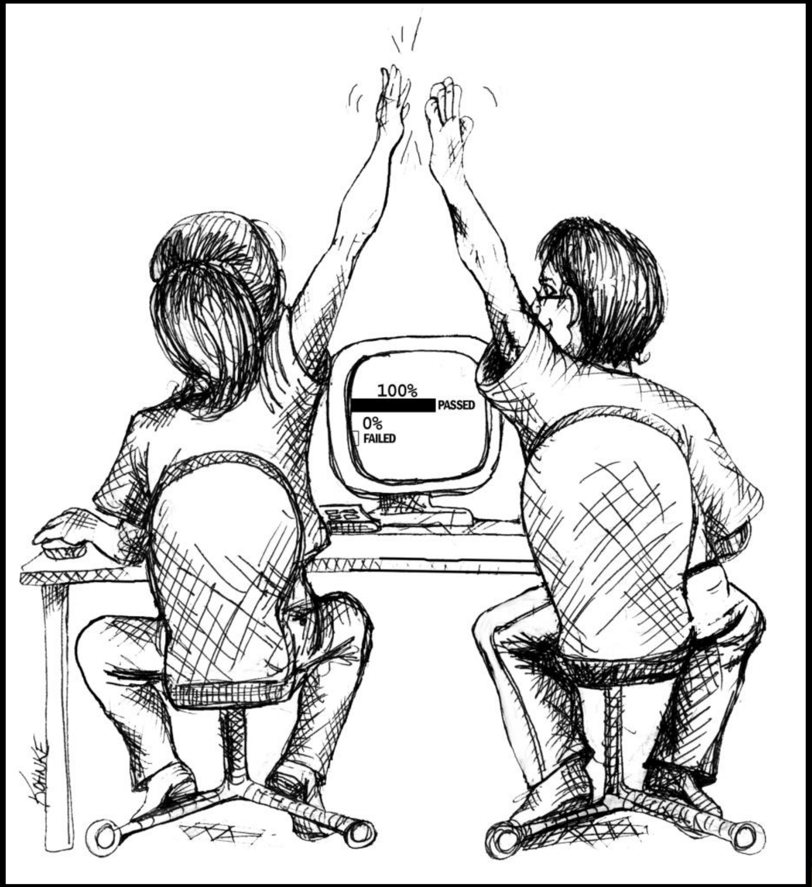
PAIRS (SHOULD) ROTATE

WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAM-MING.
scottadams@aol.com
www.dilbert.com
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
© 2002 United Feature Syndicate, Inc.
1/9/03
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.

PAIR PROGRAMMING

THIS IS NOT PAIR PROGRAMMING

THE BENEFITS OF PAIR PROGRAMMING



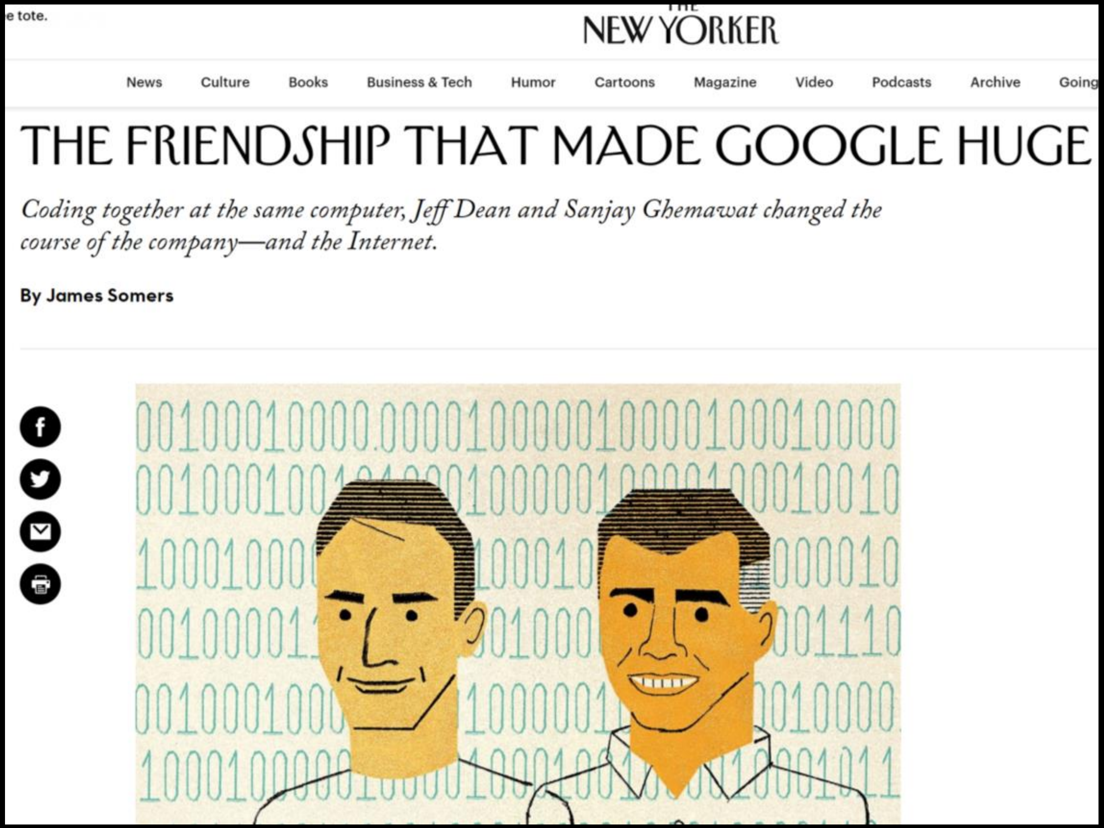
PAIR PROGRAMMING RESEARCH FINDINGS

- Strong anecdotal evidence from industry
  - "We can produce near defect-free code in less than half the time."
- Empirical Study
  - Pairs produced higher quality code
    - 15% fewer defects
  - Pairs completed their tasks in about half the time
    - 58% of elapsed time
  - Pair programmers are happier programmers
    - Pairs enjoy their work more (92%)
    - Pairs feel more confident in their work products (96%)

PAIR PROGRAMMING EXPECTED BENEFITS

- Higher product quality
- Improved cycle time
- Increased programmer satisfaction
- Enhanced learning
- Pair rotation
  - Ease staff training and transition
  - Knowledge management/Reduced product risk
  - Enhanced team building

EXAMPLE OF GOOD PAIR PROGRAMMING


e tote.

THE NEW YORKER

News Culture Books Business & Tech Humor Cartoons Magazine Video Podcasts Archive Going

THE FRIENDSHIP THAT MADE GOOGLE HUGE

Coding together at the same computer, Jeff Dean and Sanjay Ghemawat changed the course of the company—and the Internet.

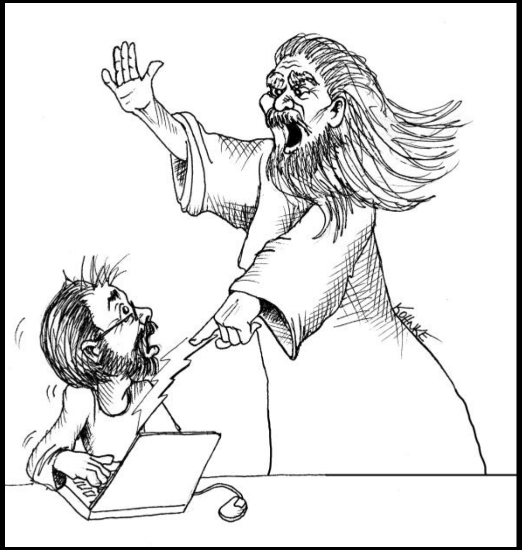
By James Somers

- Jeff Dean and Sanjay Ghemawat, two of Google's only Senior Fellows
- "I don't know why more people don't do it," Ghemawat explains.
- As Dean points out, all you need to do is "find someone that you're gonna pair-program with who's compatible with your way of thinking, so that the two of you together are a complementary force."

From: https://www.newyorker.com/magazine/2018/12/10/the-friendship-that-made-google-huge

ISSUES: PARTNER WORK

Expert paired with an Expert



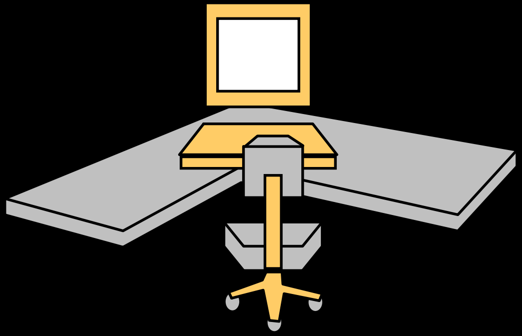
Expert paired with a Novice

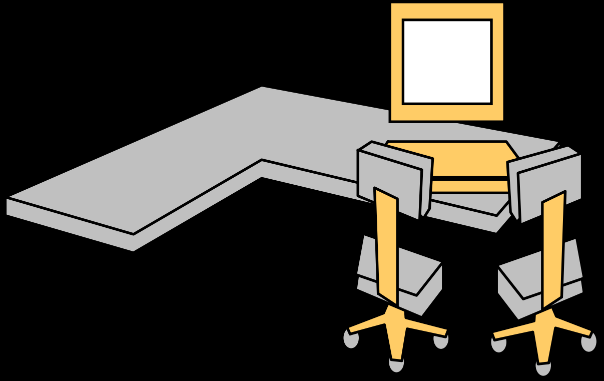
Novices paired together

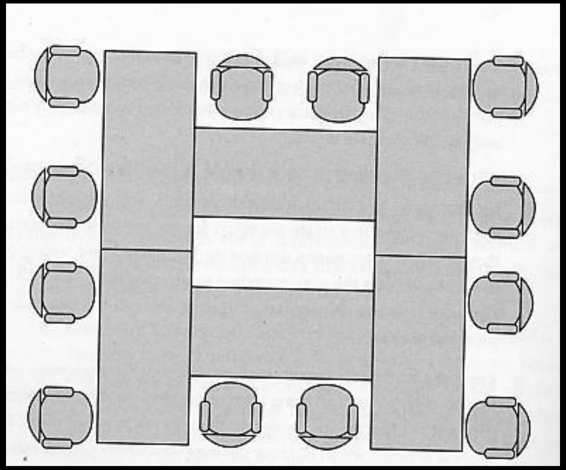
Professional Driver Problem

Culture

ISSUES: WORKPLACE LAYOUT


Bad


Better


Best

ISSUES: PROCESS

- Pair programming is an old concept; since the beginning of programming (1950s)
- Used extensively in eXtreme Programming
- But also in other development processes

- Can be added to any process !

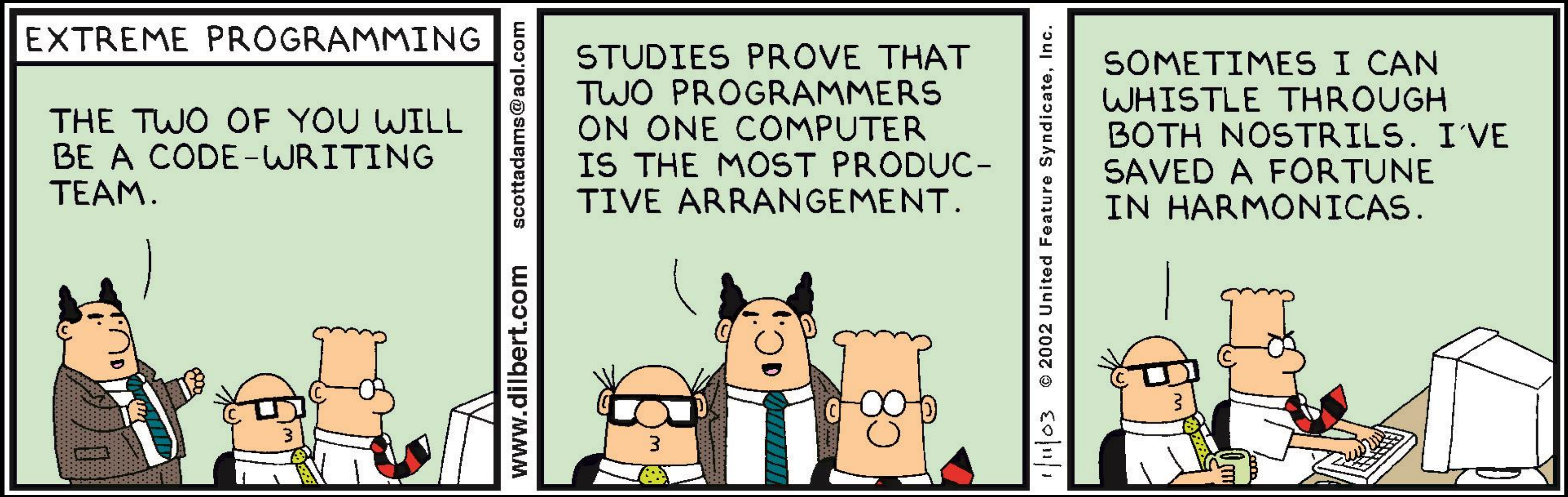
HOW/WHY DOES PAIR PROGRAMMING WORK?

- Pair Pressure
  - Keep each other on task and focused
  - Don’t want to let partner down
  - “Embarrassed” to not follow the prescribed process
- Pair Negotiation
  - Distributed Cognition: “*Searching Through Larger Spaces of Alternatives*”
    - Have shared goals and plans
    - Bring different prior experiences to the task
    - Different access to task relevant information
    - Must negotiate a common shared of action
- Pair Courage
  - “if it looks right to me and it looks right to you – guess what? It’s probably right!”

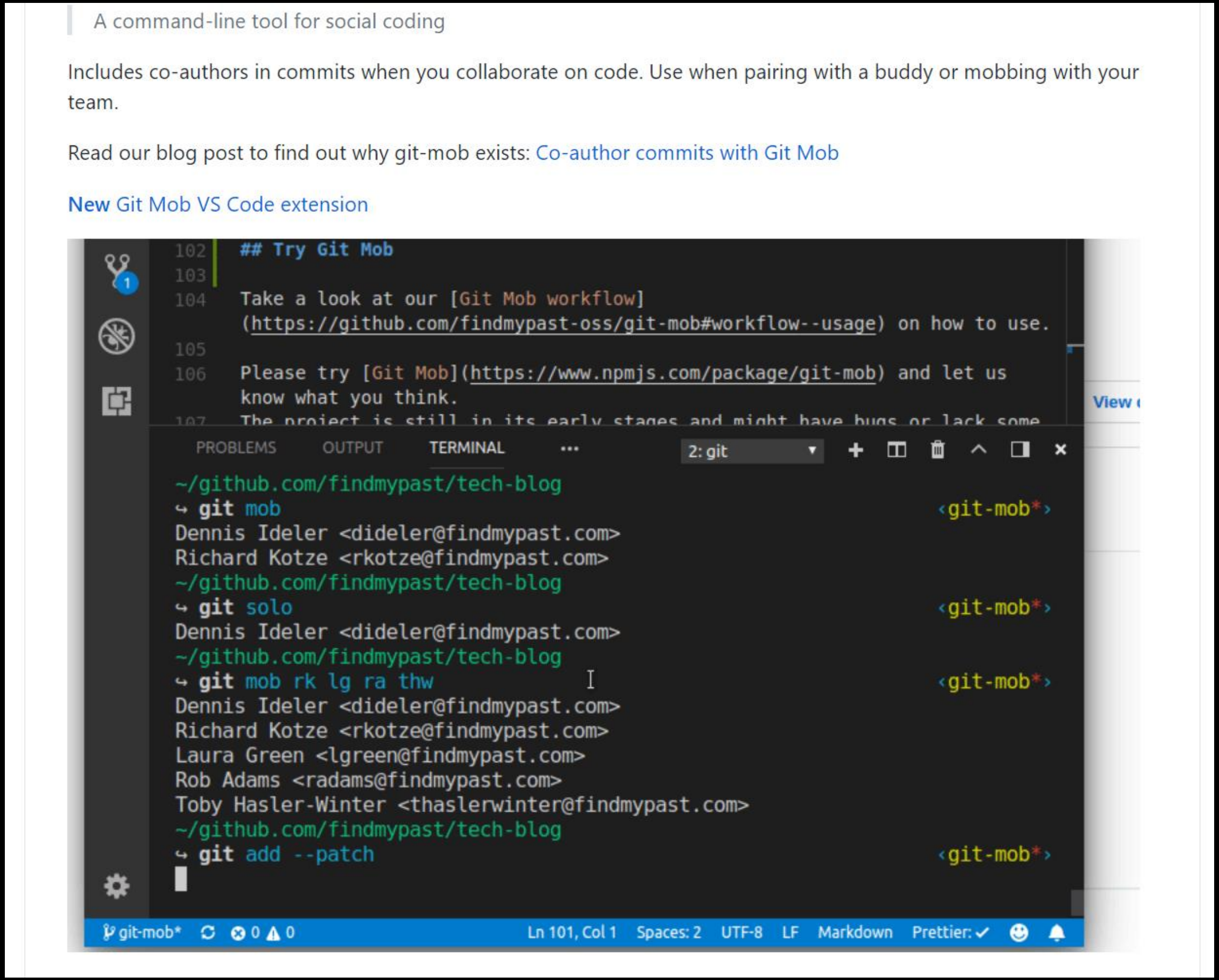
HOW/WHY DOES PAIR PROGRAMMING WORK?

- Pair Reviews
  - “Four eyeballs are better than two”
  - Continuous design and code reviews
  - Removes programmers’ distaste for reviews
    - 80% of all (solo) programmers don’t do them regularly or at all
- Pair Debugging
  - Explaining the problems to another person → “Never mind; I see what’s wrong. Sorry to bother you.”
- Pair-Learning
  - Continuous reviews → learn from partners techniques, knowledge of language, domain, etc.
  - Take turns being the teacher and the student minute by minute

"PAIRS" MEANS WORKING TOGETHER


EXTREME PROGRAMMING
THE TWO OF YOU WILL BE A CODE-WRITING TEAM.
scottadams@aol.com
www.dilbert.com
STUDIES PROVE THAT TWO PROGRAMMERS ON ONE COMPUTER IS THE MOST PRODUC-TIVE ARRANGEMENT.
© 2002 United Feature Syndicate, Inc.
1/11/03
SOMETIMES I CAN WHISTLE THROUGH BOTH NOSTRILS. I'VE SAVED A FORTUNE IN HARMONICAS.

PAIR PROGRAMMING IN GIT



A command-line tool for social coding

Includes co-authors in commits when you collaborate on code. Use when pairing with a buddy or mobbing with your team.

Read our blog post to find out why git-mob exists: Co-author commits with Git Mob

New Git Mob VS Code extension

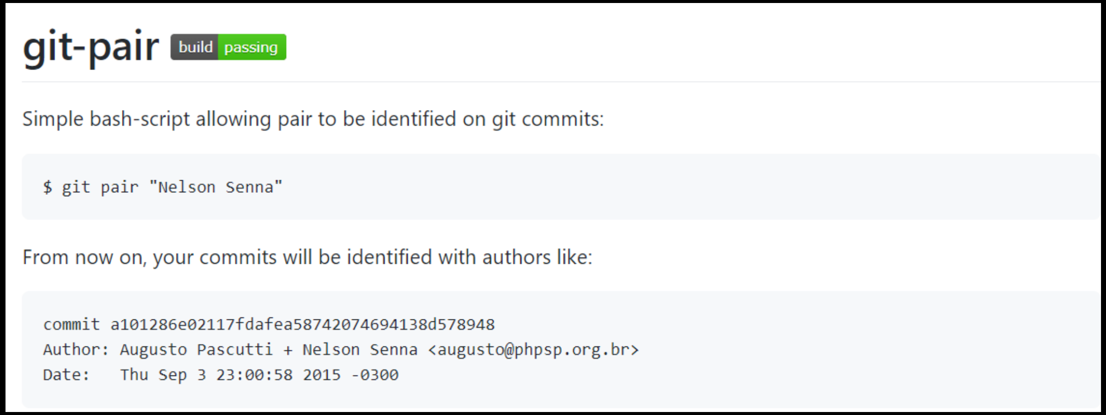
```
## Try Git Mob

Take a look at our [Git Mob workflow]
(https://github.com/findmypast-oss/git-mob#workflow--usage) on how to use.

Please try [Git Mob](https://www.npmjs.com/package/git-mob) and let us
know what you think.
The project is still in its early stages and might have bugs or lack some
```

```
~/github.com/findmypast/tech-blog
↪ git mob                                                   ‹git-mob*›
Dennis Ideler <dideler@findmypast.com>
Richard Kotze <rkotze@findmypast.com>
~/github.com/findmypast/tech-blog
↪ git solo                                                  ‹git-mob*›
Dennis Ideler <dideler@findmypast.com>
~/github.com/findmypast/tech-blog
↪ git mob rk lg ra thw                                      ‹git-mob*›
Dennis Ideler <dideler@findmypast.com>
Richard Kotze <rkotze@findmypast.com>
Laura Green <lgreen@findmypast.com>
Rob Adams <radams@findmypast.com>
Toby Hasler-Winter <thaslerwinter@findmypast.com>
~/github.com/findmypast/tech-blog
↪ git add --patch                                           ‹git-mob*›
```

- If you will code in pairs for the class project, try configuring git by acknowledging your pair programmer
- https://github.com/findmypast-oss/git-mob
- https://github.com/augustohp/git-pair



git-pair

Simple bash-script allowing pair to be identified on git commits:

```
$ git pair "Nelson Senna"
```

From now on, your commits will be identified with authors like:

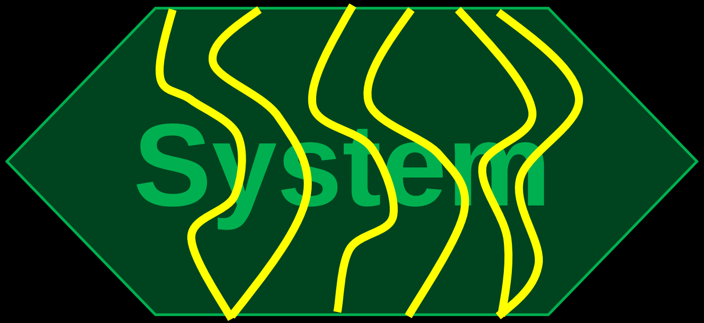
```
commit a101286e02117fdafea58742074694138d578948
Author: Augusto Pascutti + Nelson Senna <augusto@phpsp.org.br>
Date:   Thu Sep 3 23:00:58 2015 -0300
```

RECALL:
XP IS AN ITERATIVE PROCESS

- Iteration = two week cycle (1-3 weeks)
- Plan each iteration in an iteration meeting that is held at the start of the iteration
- Iteration is going to implement set of user stories
- Divide work into tasks small enough to finish in a day
- Each day, programmers work in pairs to finish tasks

WHAT ARE USER STORIES?

- A user story represents
  - A feature customers want in the software
- A user story is the smallest amount of information (a step) necessary to allow the customer to define (and steer) a path through the system



- Written by customers (through communication with developers), and not by developers (!)
- Typically written on index cards

WRITING USER STORIES

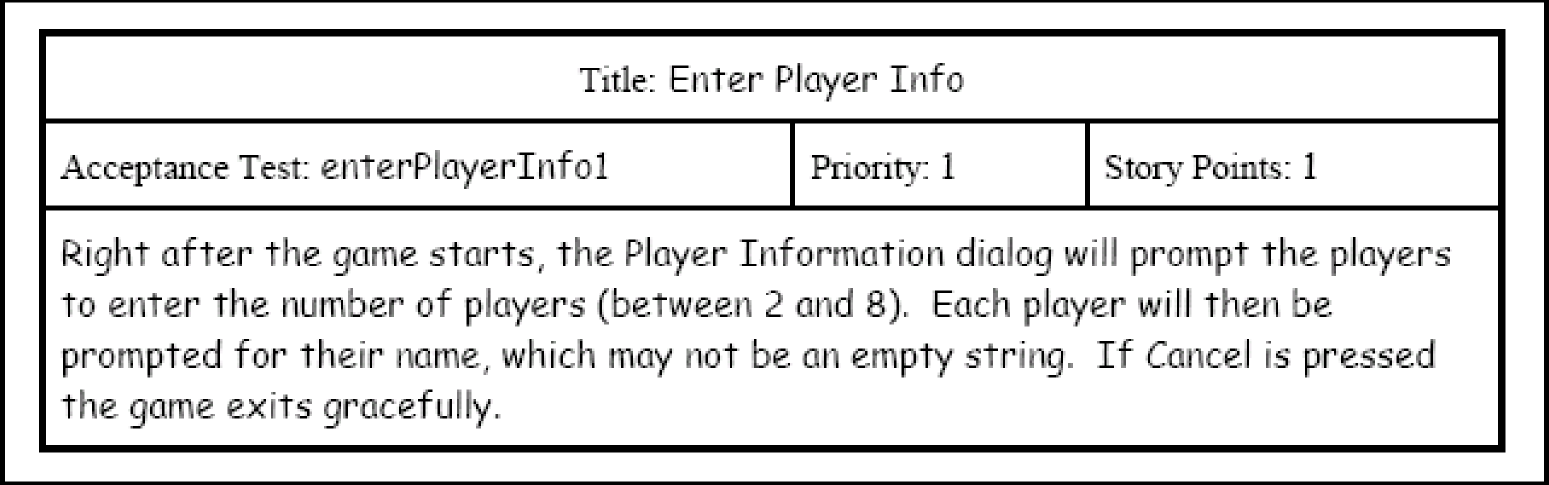
Materials

- A stack of blank index cards
- Pens or pencils
- Rubber bands ☺

- Start with a goal of the system
  - e.g., “Applicant submits a loan application.”
- Think about the steps that the user takes as he/she does the activity
- Write no more than one step on each card

FORMAT OF A USER STORY

- Title: 2-3 words
- Acceptance test (unique identifier)
- Priority: 1-2-3 (1 most important)
- Story points (can mean #days of ideal development time, i.e., no distractions or working on other things)
- Description: 1-2 sentences (a single step towards achieving the goal)



Title: Enter Player Info		
Acceptance Test: enterPlayerInfo1	Priority: 1	Story Points: 1
Right after the game starts, the Player Information dialog will prompt the players to enter the number of players (between 2 and 8). Each player will then be prompted for their name, which may not be an empty string. If Cancel is pressed the game exits gracefully.		

EXAMPLE ACCEPTANCE TEST FOR A USER STORY

Create Receipt

Keep a running receipt with a short description of each scanned item and its price

`createReceipt1`

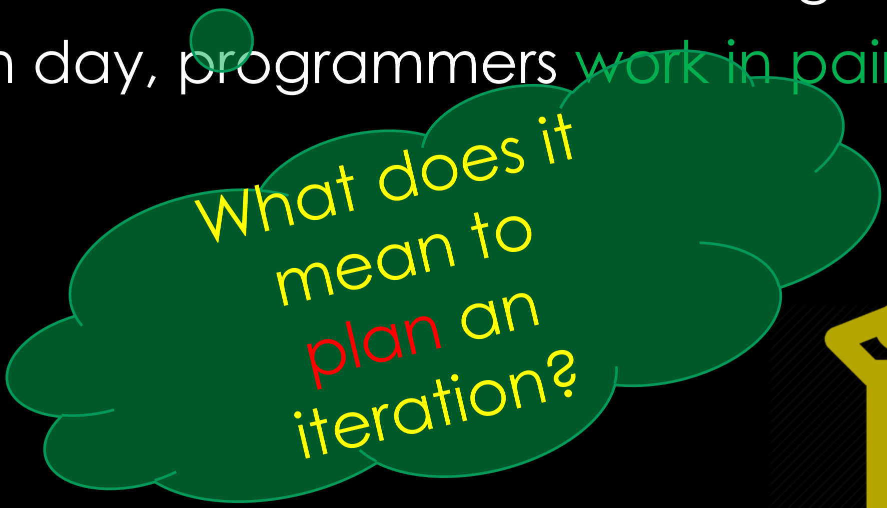
Setup: The cashier has a new customer

Operation: The cashier scans three cans of beans at $0.99, two pounds of spinach at $0.59/lb, and a toothbrush at $1.99

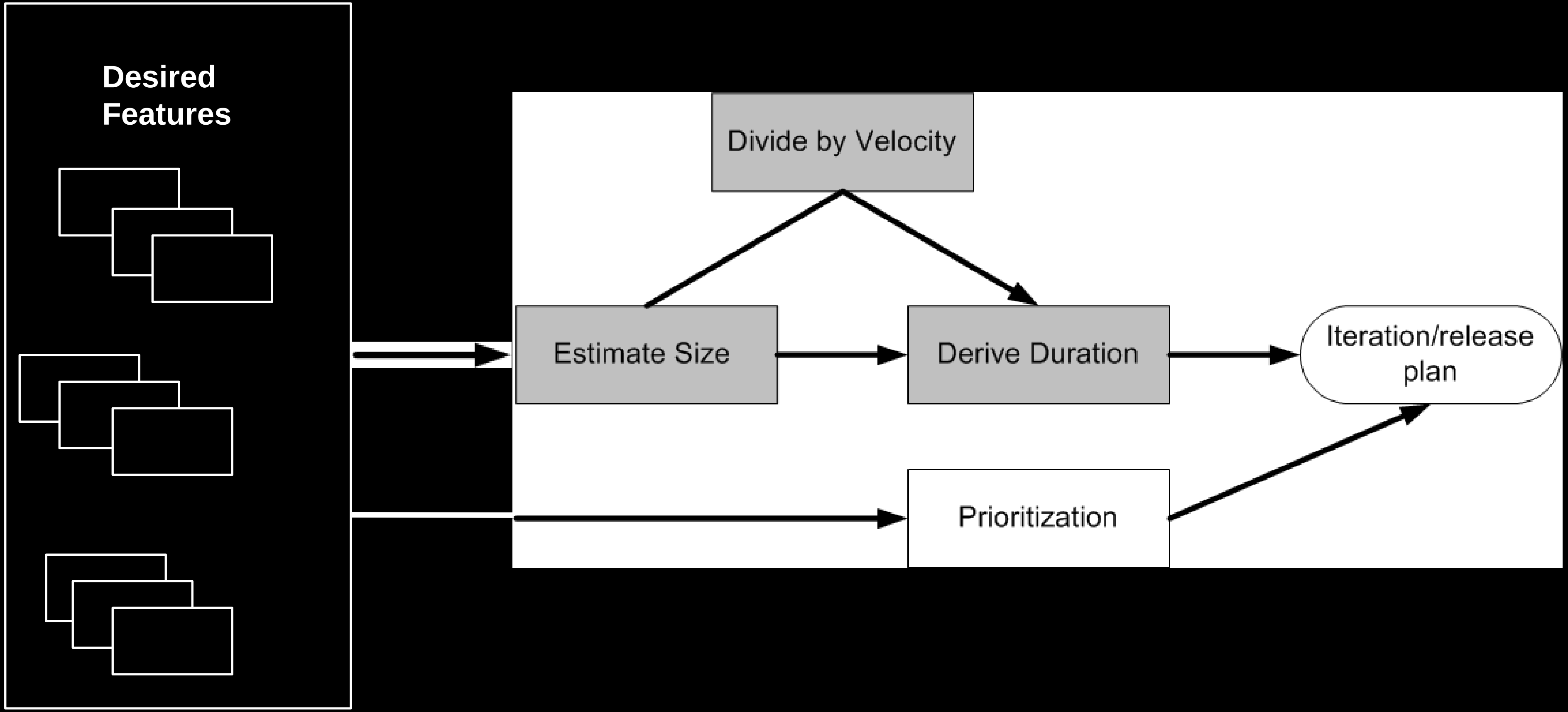
Verify: The receipt has all of the scanned items and their correctly listed prices

RECALL: XP IS AN ITERATIVE PROCESS

- Iteration = two week cycle (1-3 weeks)
- Plan each iteration in an iteration meeting that is held at the start of the iteration
- Iteration is going to implement set of user stories
- Divide work into tasks small enough to finish in a day
- Each day, programmers work in pairs to finish tasks



COMING UP WITH THE PLAN



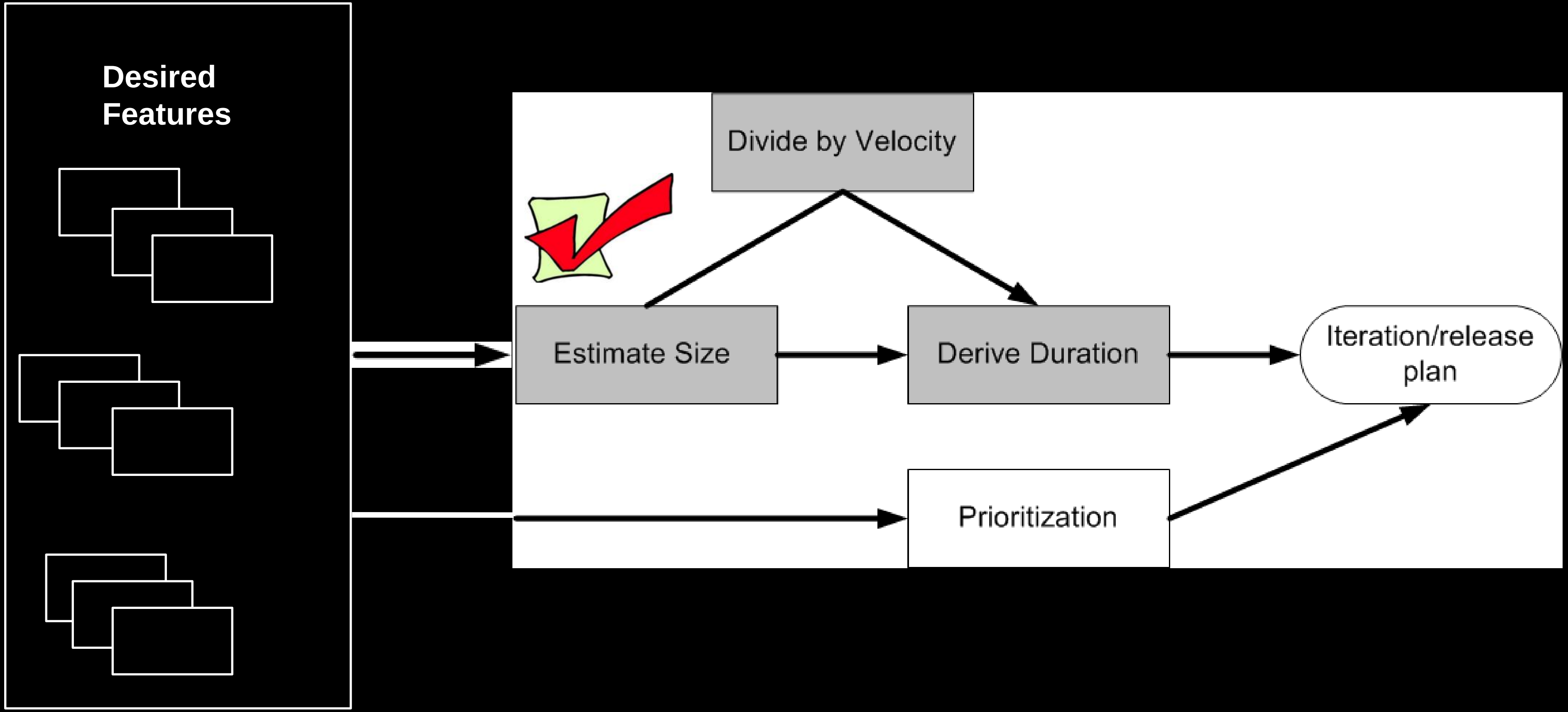
CONCEPTS

- Story point: unit of measure for expressing the overall size of a user story, feature, or other piece of work. The raw value of a story point is unimportant. What matters are the relative values.
  - Related to how hard it is and how much of it there is
  - Not related to amount of time or the number of people
  - Unitless, but numerically-meaningful
- Ideal time: the amount of time "something" takes when stripped of all peripheral activities
  - Example: American football game = 60 minutes
- Elapsed time: the amount of time that passes on the clock to do "something"
  - Example: American football game = 3 hours
- Velocity: measure of a team's rate of progress

PRIORITIES

- High
  - “Give us these stories to provide a minimal working system.”
- Medium
  - “We need these stories to complete this system.”
- Low
  - “Bells and whistles? Which stories can come later?”

COMING UP WITH THE PLAN



ESTIMATING STORY POINTS

- Choose a medium-size story and assign it a "5"
- Estimate other stories relative to that
  - Twice as big
  - Half as big
  - Almost but not quite as big
  - A little bit bigger
- Only values:
  - 0, 1, 2, 3, 5, 8, 13, 20, 40, 100

Near term iteration "stories"

A few iterations away "epic"

ESTIMATING IDEAL DAYS

- Ideal days vs. elapsed time in software development
  - Supporting current release
  - Sick time
  - Meetings
  - Demonstrations
  - Personal issues
  - Phone calls
  - . . . .
- When estimating ideal days, assume:
  - The story being estimated is the only thing you'll work on
  - Everything you need will be on hand when you start
  - There will be no interruptions

IDEAL DAYS VS. STORY POINTS

- Favoring story points:
  - Help drive cross-functional behavior
  - Do not decay (change based on experience)
  - Are a pure measure of size (focus on feature, not person)
  - Estimation is typically faster in the long run
  - My ideal days are not your ideal days (focus on person and their speed ☹)
- Favoring ideal days:
  - Easier to explain outside of team
  - Estimation is typically faster at first

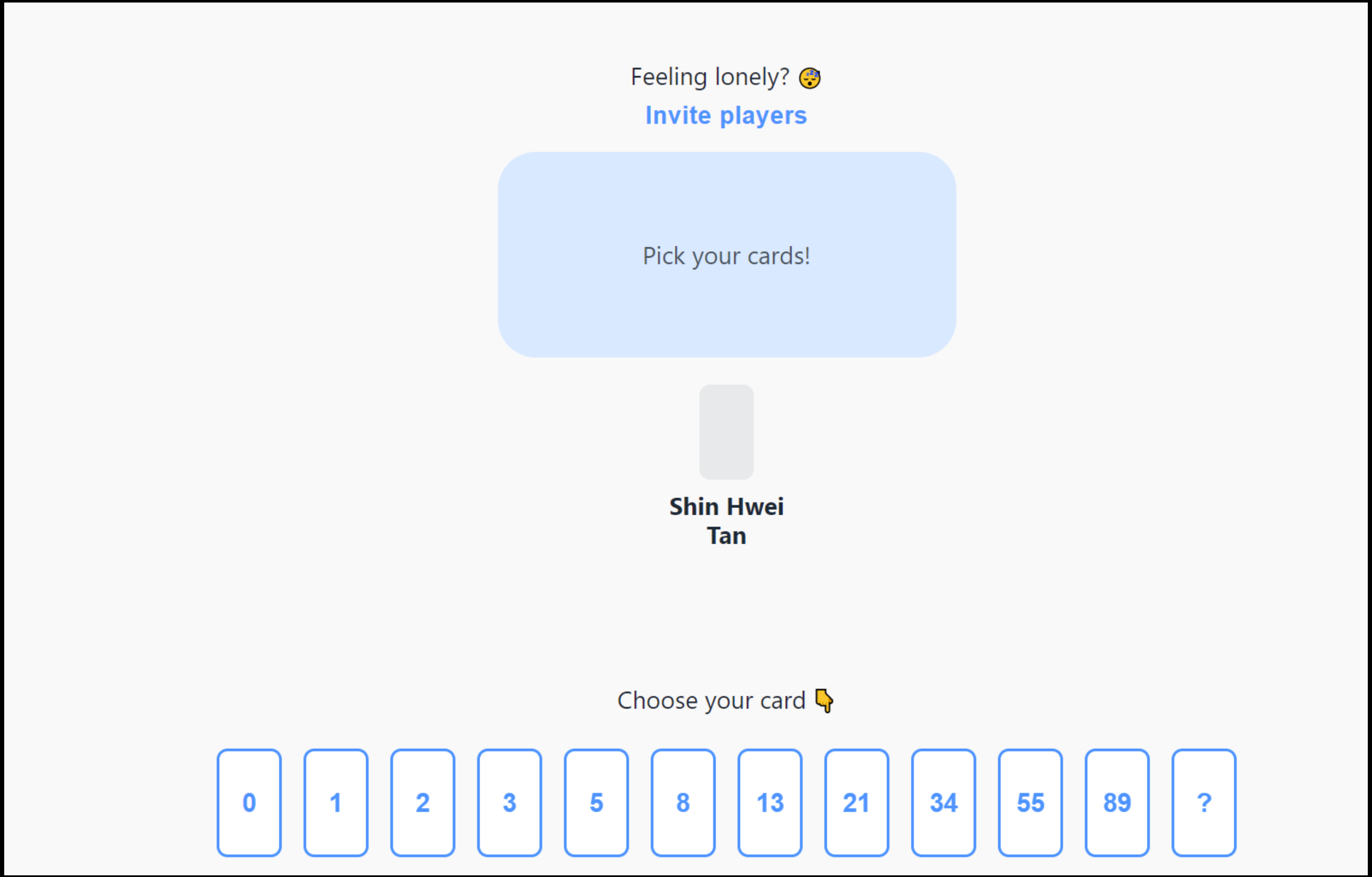
DERIVING AN ESTIMATE FOR A USER STORY

- Expert opinion
  - Rely on gut feel based on (extensive) experience
  - Disadvantage for agile: need to consider all aspects of developing the user story, so one expert will likely not be enough
- Analogy
  - Relative to (several) other user stories
  - Triangulation: little bigger than that “3” and a little smaller than that “8”
- Disaggregation
  - Break up into smaller, easier-to-estimate pieces/tasks.
  - Need to make sure you don’t miss any tasks.
  - Sanity check: does the sum of all the parts make sense?
- Planning poker
  - Combines expert opinion, analogy, disaggregation

WHAT ARE THIS SEQUENCE CALLED? 1, 1, 2, 3, 5, 8, 13 AND 21.

1. Odd numbers
2. $X_{i+1}=X_i+2$
3. Fibonacci Sequence

PLANNING POKER



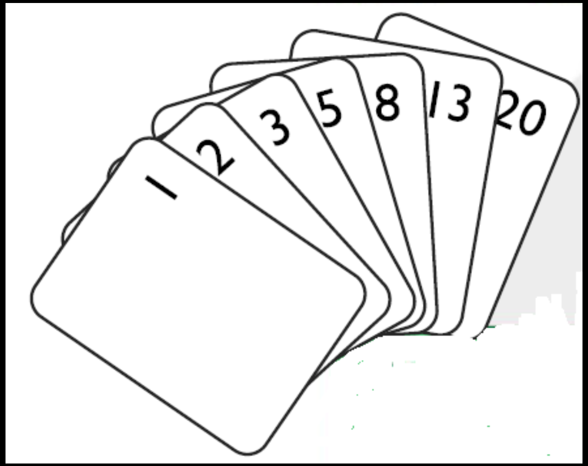
Feeling lonely? 😴

Invite players

Pick your cards!

Shin Hwei Tan

Choose your card 👇

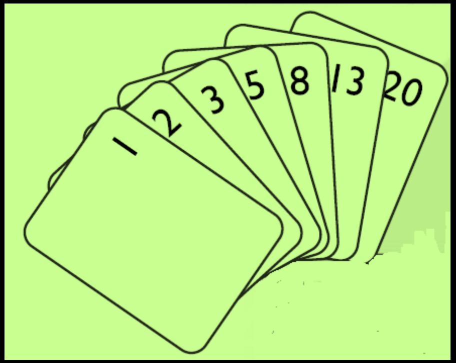


RULES OF THE GAME

- Let’s watch this video to find out what planning poker is all about:
- http://youtu.be/0FbnCWWg_NY

PLAYING PLANNING POKER

- Include all players on the development team (but less than 10 people overall):
  - Programmers
  - Testers
  - Database engineers
  - Requirements analysts
  - User interaction designers . . .
- Moderator (usually the product owner or analyst) reads the description and answers any questions
- Each estimator privately selects a card with their estimate
- All cards simultaneously turned over
- Re-estimate
- Repeat until converge



GROUP DISCUSSION: PLAY PLANNING POKER

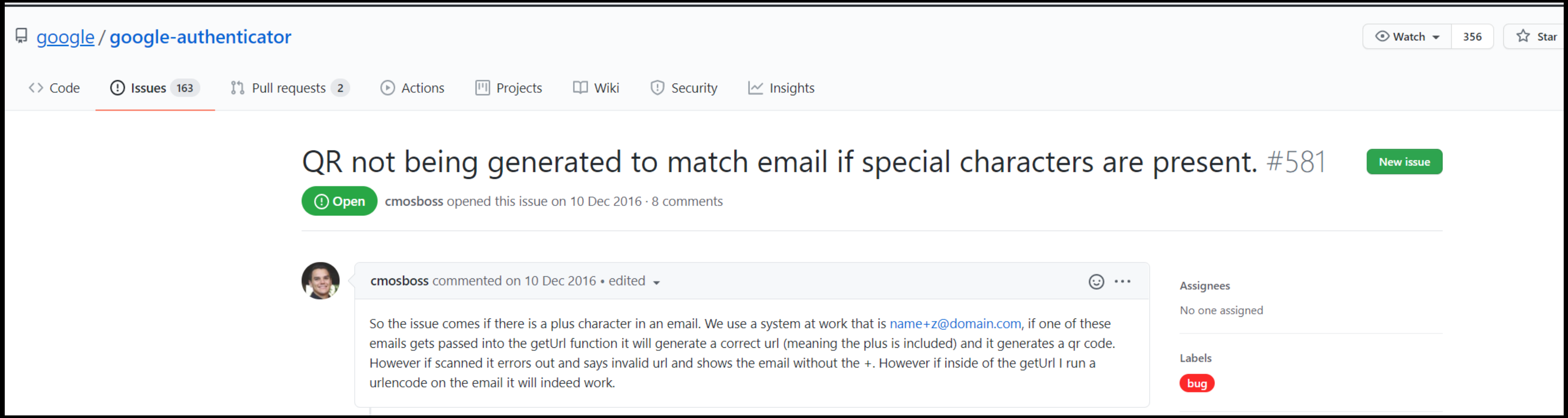
- One student in the group serves as the moderator by providing a set of tasks for an upcoming period (e.g., upcoming week, month, semester) for the rest of the group to estimate cost.
- Each student uses small pieces of papers to write 1, 2, 3, **5**, 8, 13, 20, 40, 100 on each of them.
- Carry out to play planning poker

- https://planningpokeronline.com/9zdZSs2q4k1whlknoY3R

https://en.wikipedia.org/wiki/Planning_poker

GROUP DISCUSSION: PLAY PLANNING POKER

User stories:
- Add an Espresso test
- Handle special character



QR not being generated to match email if special characters are present. #581

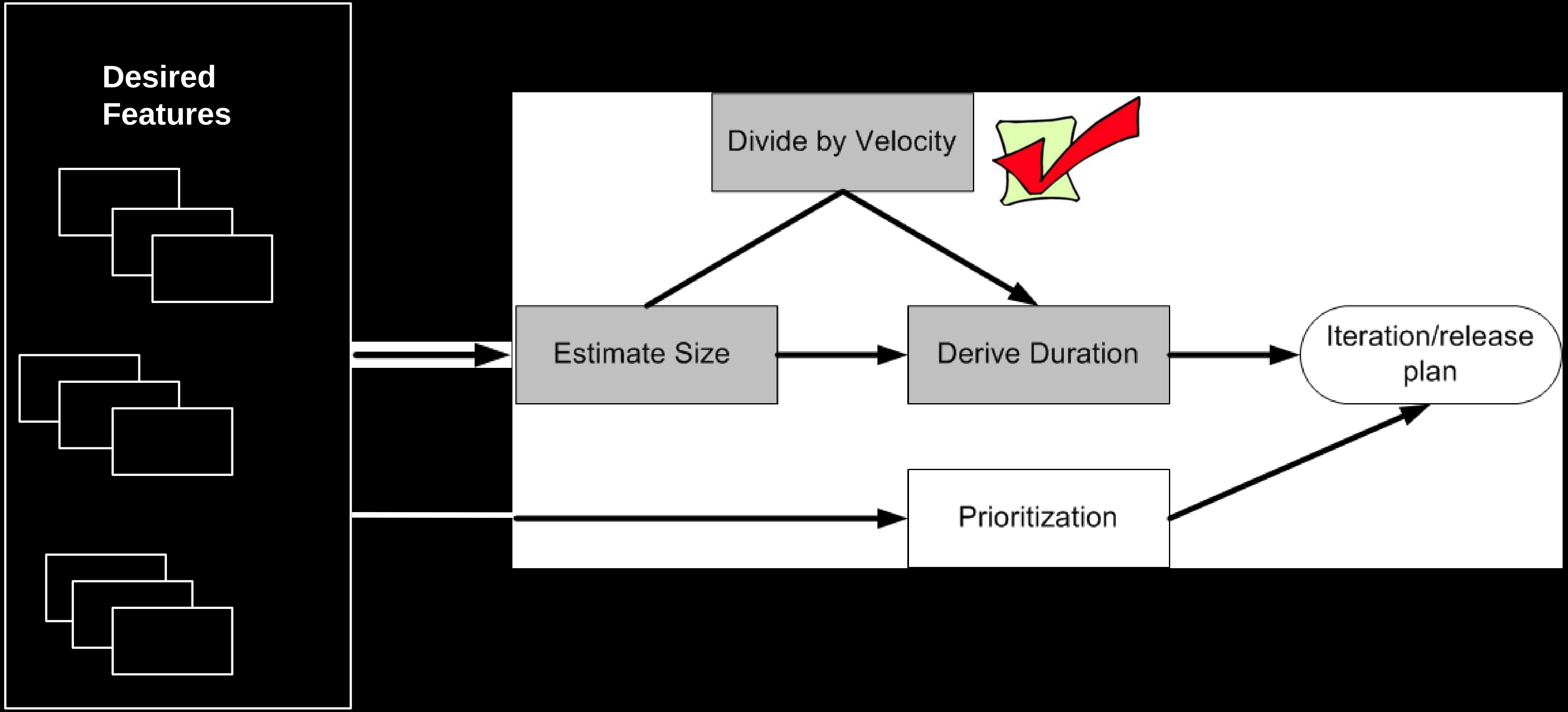
Open cmosboss opened this issue on 10 Dec 2016 · 8 comments

cmosboss commented on 10 Dec 2016 • edited

So the issue comes if there is a plus character in an email. We use a system at work that is name+z@domain.com, if one of these emails gets passed into the getUrl function it will generate a correct url (meaning the plus is included) and it generates a qr code. However if scanned it errors out and says invalid url and shows the email without the +. However if inside of the getUrl I run a urlencode on the email it will indeed work.

https://github.com/google/google-authenticator/issues/581

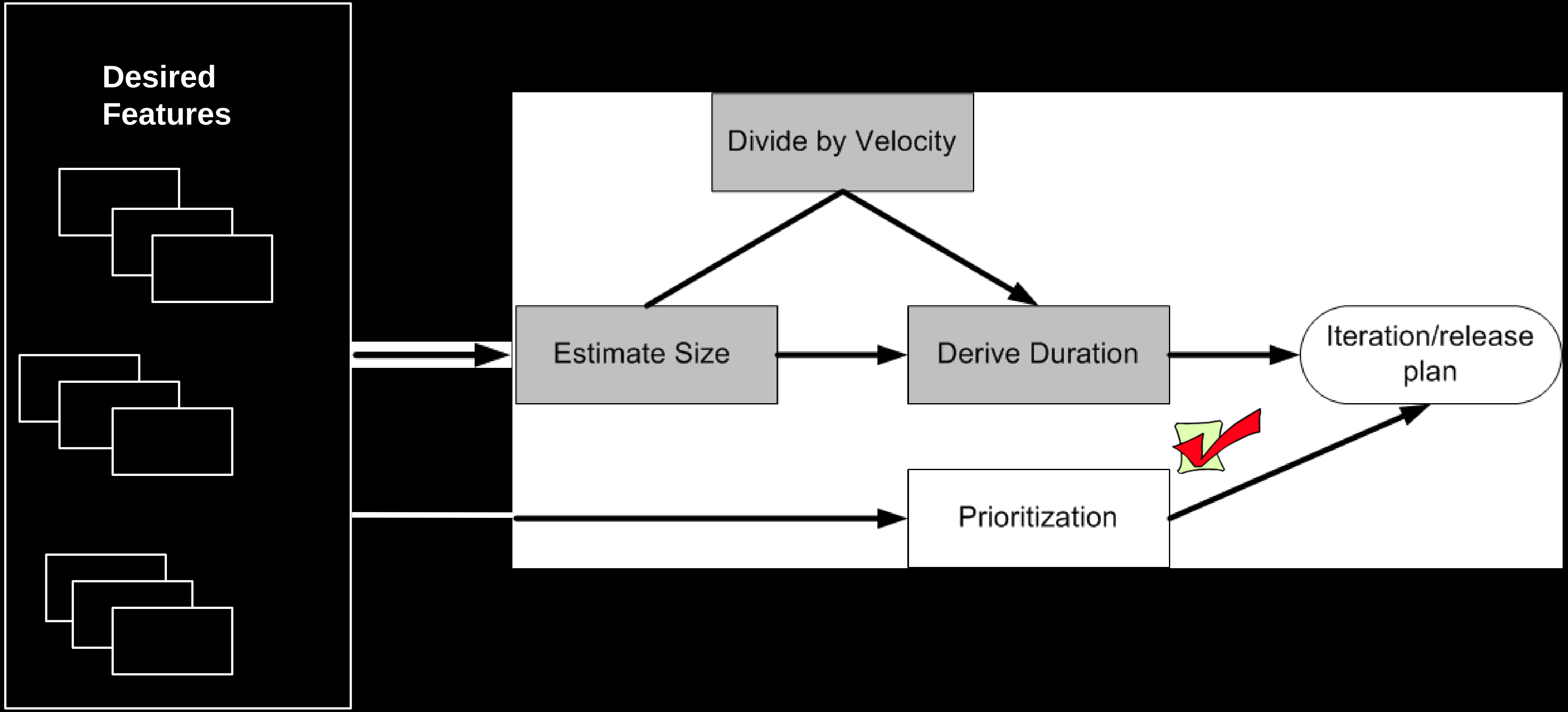
COMING UP WITH THE PLAN



VELOCITY

- Velocity is a measure of a team's rate of progress.
- Velocity is calculated by summing the number of story points assigned to each user story that the team <u>completed</u> during the operation.
- We assume that the team will produce in future iterations at the rate of their past average velocity.
    - "Yesterday's weather"

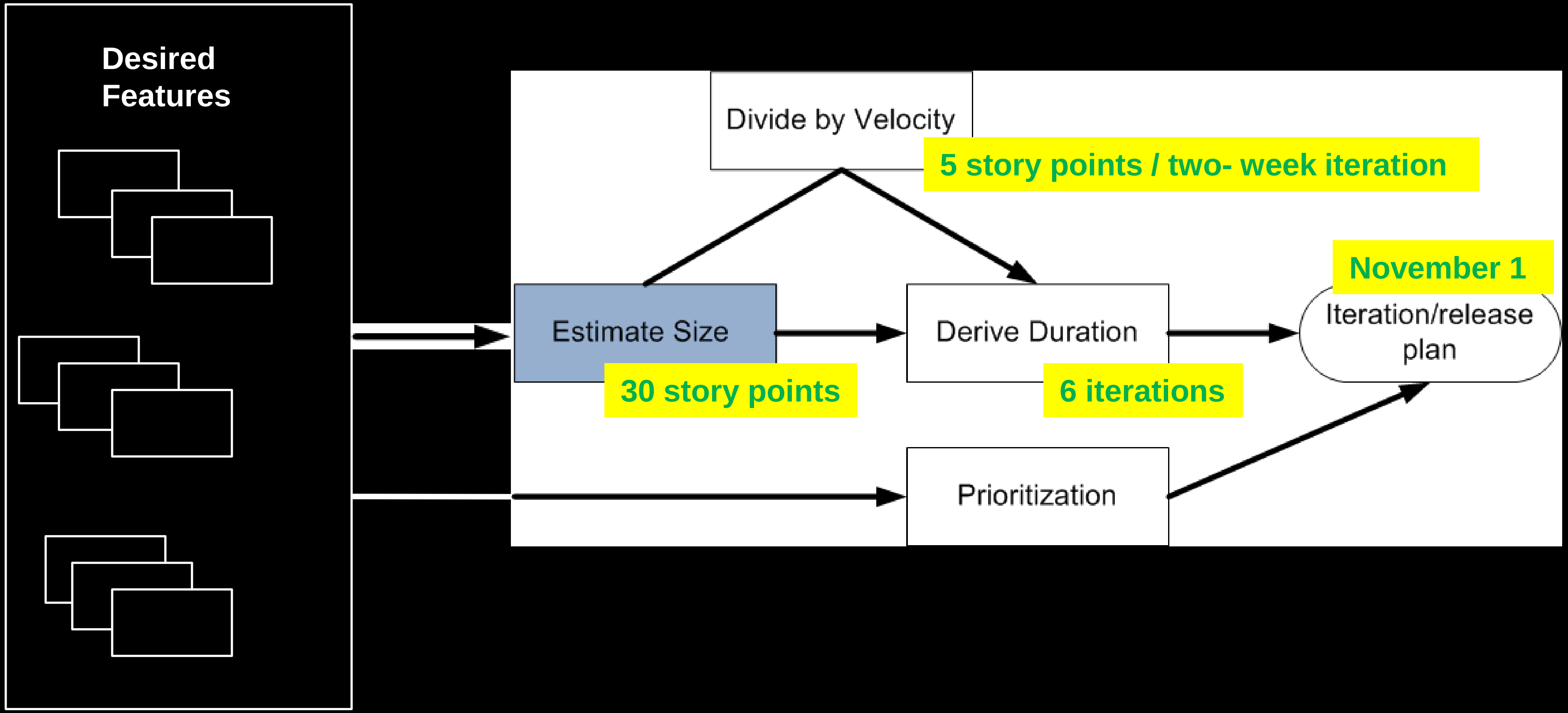
COMING UP WITH THE PLAN



PRIORITIZATION

- Driven by customer, in conjunction with developer
- Choose features to fill up velocity of iteration, based on:
  - Desirability of feature to a broad base of customers/users
  - Desirability of feature to a small number of important customers/users
  - The cohesiveness of the story in relation to other stories Example:
    - "Zoom in" a high priority feature
    - "Zoom out" not a high priority feature
      - But it becomes one relative to "Zoom in"

COMING UP WITH THE PLAN



PLANNING GAME

- Customer writes user stories
- Programmers estimate time to do each story
- If story is too big, customer splits it
- Customer chooses stories to match project velocity
- Project velocity is amount of work done in the previous iteration(s)

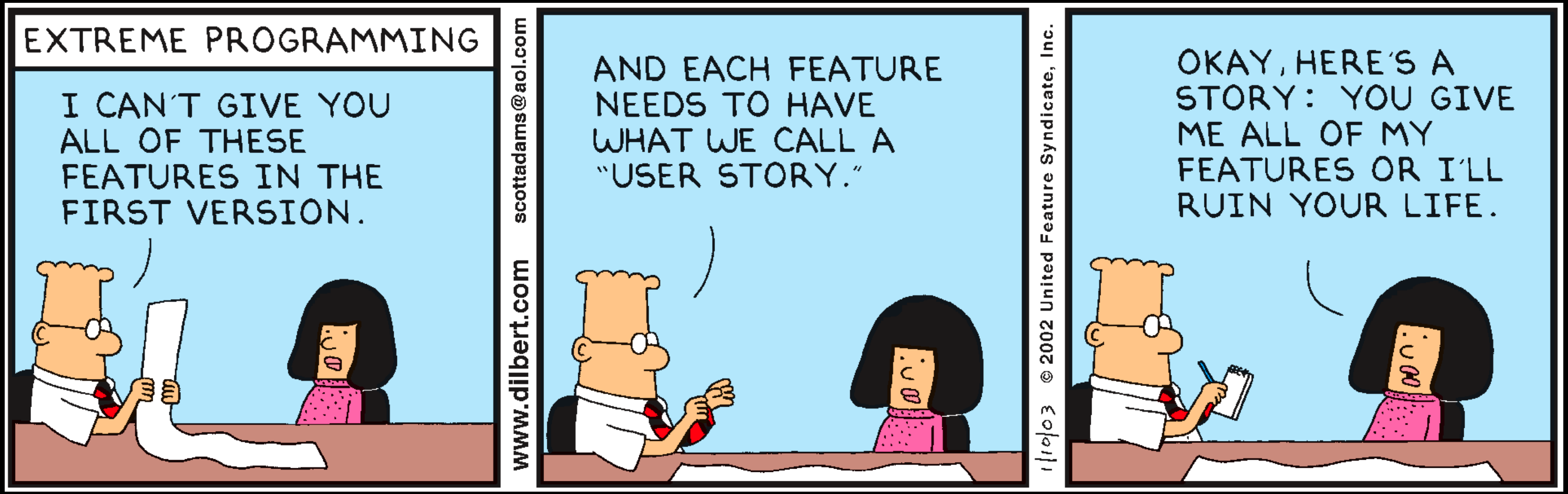
PLANNING

- Programmers only worry about one iteration at a time
- Customer can plan as many iterations as desired, but can change future iterations

SIMPLICITY

- Add one feature (user story) at a time
- Don’t worry about future stories
- Make program as simple as possible
- The simplest thing that could possibly work

NEED EDUCATED CUSTOMER


EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
www.dilbert.com scottadams@aol.com
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
© 2002 United Feature Syndicate, Inc.
1/10/03
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.

XP WORKS BEST WHEN

- Educated customer on site
- Small team
- People who like to talk
- All in one room (including customer)
  - These days the room can also be virtual (slack channel?)
- Changing requirements

UNIT TESTS AND REFACTORING

- Because code is as simple as it can be, adding a new feature tends to make it less simple
- To recover simplicity, you must refactor the code
- To refactor safely, you should have a rigorous set of unit tests

WORKING SOFTWARE

- All software has automated (unit) tests
- All tests pass, all the time
  - Never check in broken code
- How to work on a task
  - Get latest version of the code. All tests pass.
  - Write test first. It fails.
  - Write code to make test pass. Now all tests pass.
  - Refactor (clean up)
  - Check in your code

ONE KEY PRACTICE

- Write tests first, then write code
- Various names
  - Test-first programming
  - Test-driven development
- Is it testing or designing?

- Degree to which you stick to it for MP?

WHY TEST?

- Improve quality - find bugs
- Measure quality
  - Prove there are no bugs? (Is it possible?)
  - Determine if software is ready to be released
  - Determine what to work on
  - See if you made a mistake
- Learn the software
- Grade MPs ☺