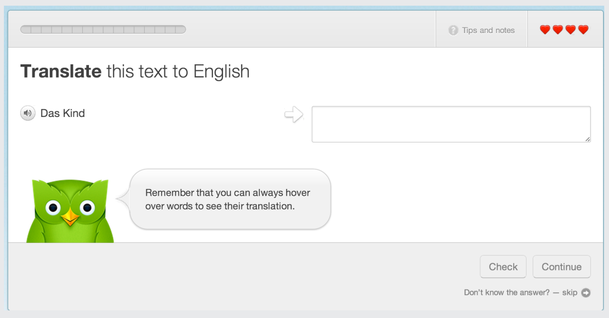
Crowd Development

Thomas D. LaToza[1], W. Ben Towne[2], André van der Hoek[1], and James D. Herbsleb[2]
[1] University of California, Irvine [2] Carnegie Mellon University

Translate the web… by learning a language

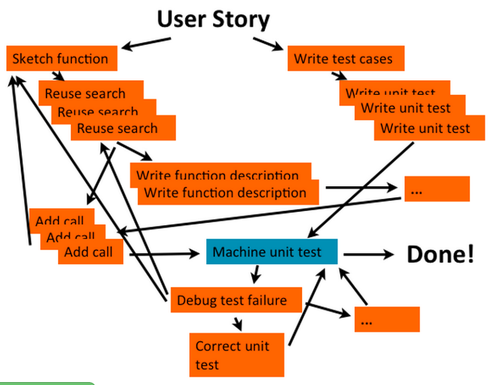


Non-expert players solved a decade long heard research problem in 10 days.

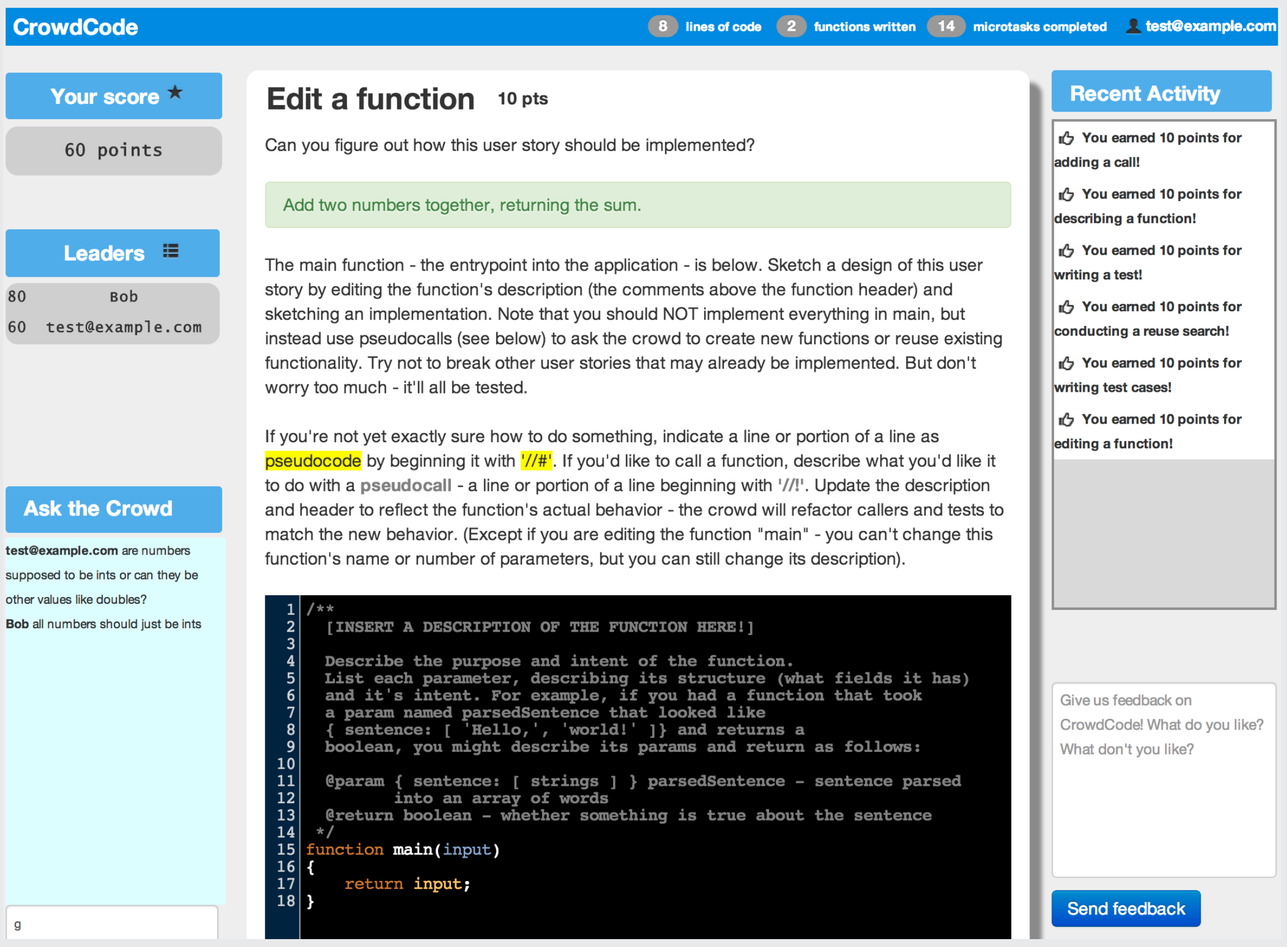
CrowdCode

Build software with a crowd!

CrowdCode organizes work into **microtasks**, small, self-describing bits like writing psuedocode or brainstorming test cases. After you finish a microtask, CrowdCode figures out what to do next, generating and distributing microtasks to the **crowd**. So you might write a description for function one, debug a test failure for another, and then edit the pseudocode the crowd wrote for function one to add a call. As you complete microtasks, you earn **points**, and can see how you're doing on the leaderboard.

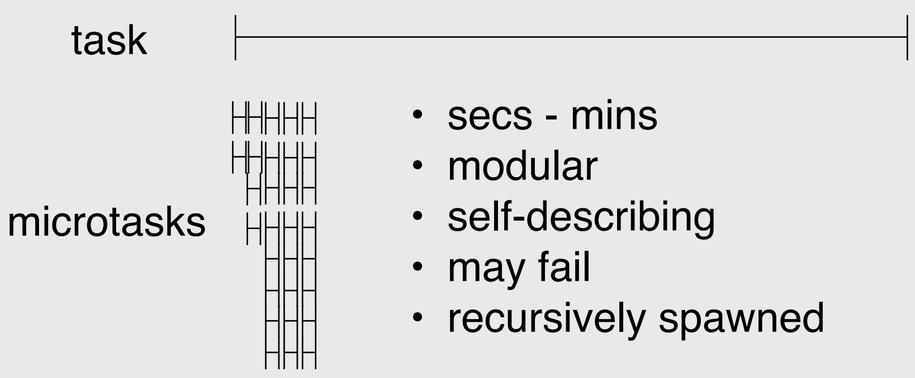


Let's get started!



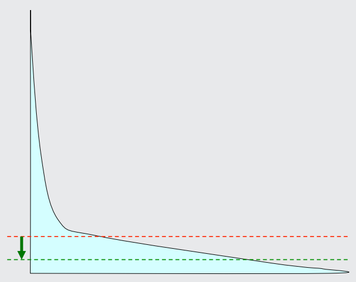
What if a large application could be built in a day?

Increasing **parallelism** reduces time to market



- secs - mins
- modular
- self-describing
- may fail
- recursively spawned

What if you could navigate to a site and start contributing?



capture the "**long tail**" of contribution by lowering joining costs

- just **start** working
- **stop** when you're bored
- **skip** what you don't want to do

What if grandma or grandpa's hobby was writing software?

reducing range of knowledge required enables **specialization** through editors or in expertise

- some artifacts edited by EUP w/ EUP tools
- expert architect contributes for a few microtasks
- workers become expert in sort routines

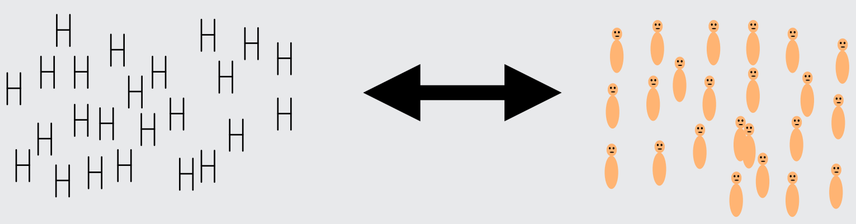
What if software development felt like a game?

optimal challenge + **social** might make building software more fun & educational

- earn as many **points** as you can
- **level** up to harder work
- cash in points for **prizes**
- watch how your **friends** are doing
- compete in pair programming

Code with Friends

How can work be effectively allocated to a crowd?



worker characteristics, **task** characteristics

- experience w/ relevant artifacts
- skill in type of programming
- reputation / trust
- overall knowledge of system

best available match vs. wait for better match

How can a crowd create a design with conceptual integrity?

push & pull information over the **dependency** graph
iterative critique - many solutions, **critique**, recombine, iterate
collective decision making - **StackOverflow** for a project

How do you create high-quality software with transient, potentially malicious workers of varying expertise?

redundancy, reviews, reputation

apportion **value created** back to contributors

- function reused
- function passes all its tests
- user story completed quickly

SDCL Software Design and Collaboration Laboratory
UCIrvine