



Tips and Tricks

Software Engineering II
WS 2015/16

Arian Treffer
arian.treffer@hpi.de

Prof. Plattner, Dr. Uflacker
Enterprise Platform and Integration Concepts

Agenda



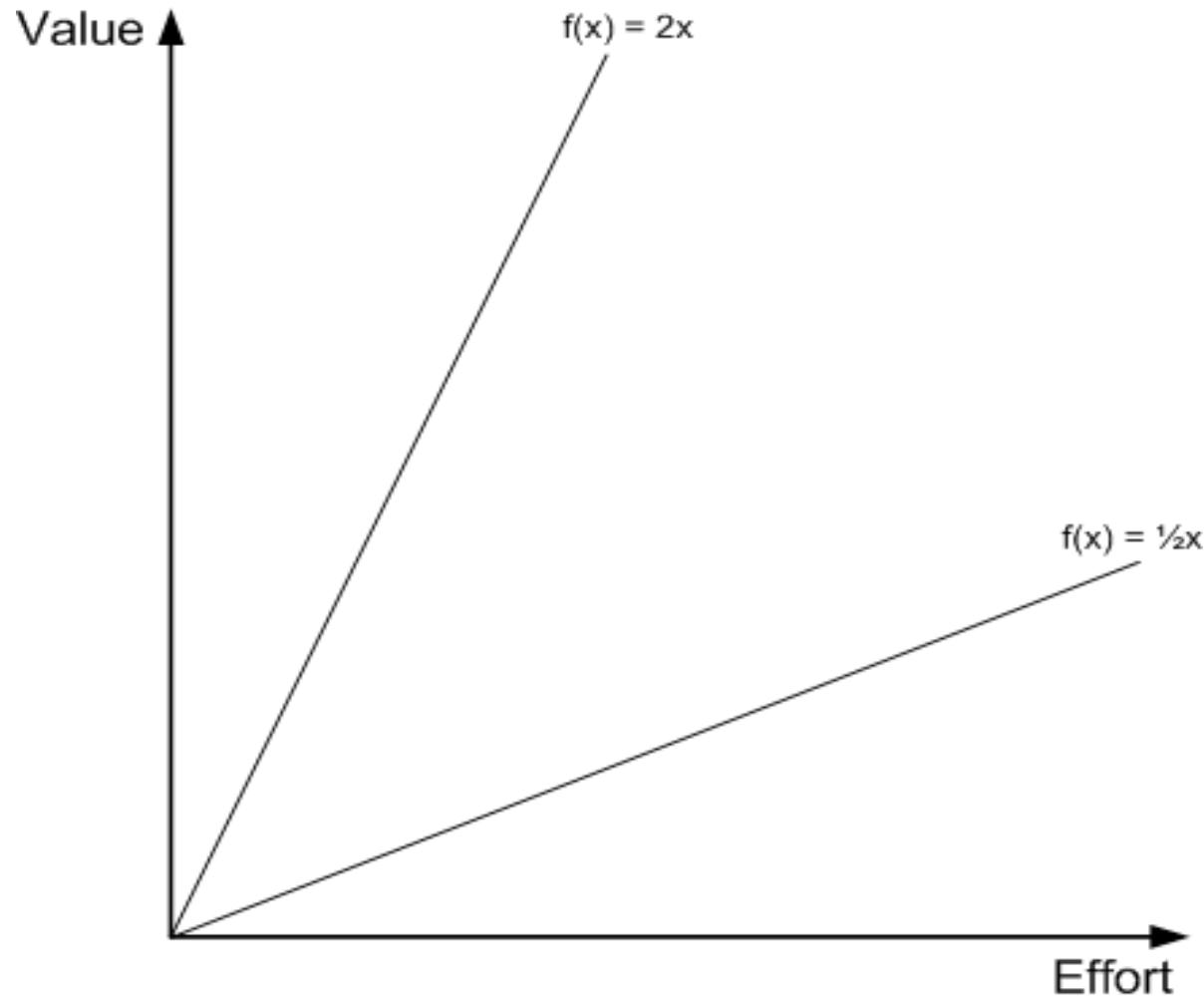
1. Value-based Requirements Analysis
2. Organizing your Project
3. Git Tricks
4. Outlook

Value-based Requirements Analysis



- Requirements are often analyzed in a value-neutral environment
[Boehm, Barry W. "Value-based software engineering: Overview and agenda." 2006]
- 80% of the value is expressed in 20% of the requirements (Pareto principle)
[Koch, 1998]
- A value-oriented approach is more appropriate
- How to do that?
 - Identify the system's success-critical stakeholders
 - Obtain their value propositions with respect to the system
 - Estimate / find out value of a requirement to the stakeholders
 - Estimate effort to implement a requirement

Value-based Requirements Analysis



Rules:

- Implement: Above $2x$
- Skip: Below $\frac{1}{2}x$
- In-between: Review

- Whole truth?
- Beware of dependencies!

Agenda



1. Value-based Requirements Analysis
2. Organizing your Project
 - Scrum Burn-Down Chart
 - Communication
 - Dealing with Dependencies
 - Estimating Large Backlogs
 - Beyond Scrum
3. Git Tricks
4. Outlook

Organizing your Project

Questions:

- Which stories are part of Sprint#1?
- Who is working on which tasks?
- Which version is a good one that can be shown to the customer?

Tools that might help:

- Put your user stories & tasks into Github's issue tracker
 - Assign issues to developers
 - Use milestones to assign user stories to sprints
 - Use issue tags, e.g. to denote responsible teams or status
 - Use "project management" tools that give an overview of GH issues, e.g. <https://waffle.io/> or <https://www.zenhub.io/>
- Tag versions that can be presented

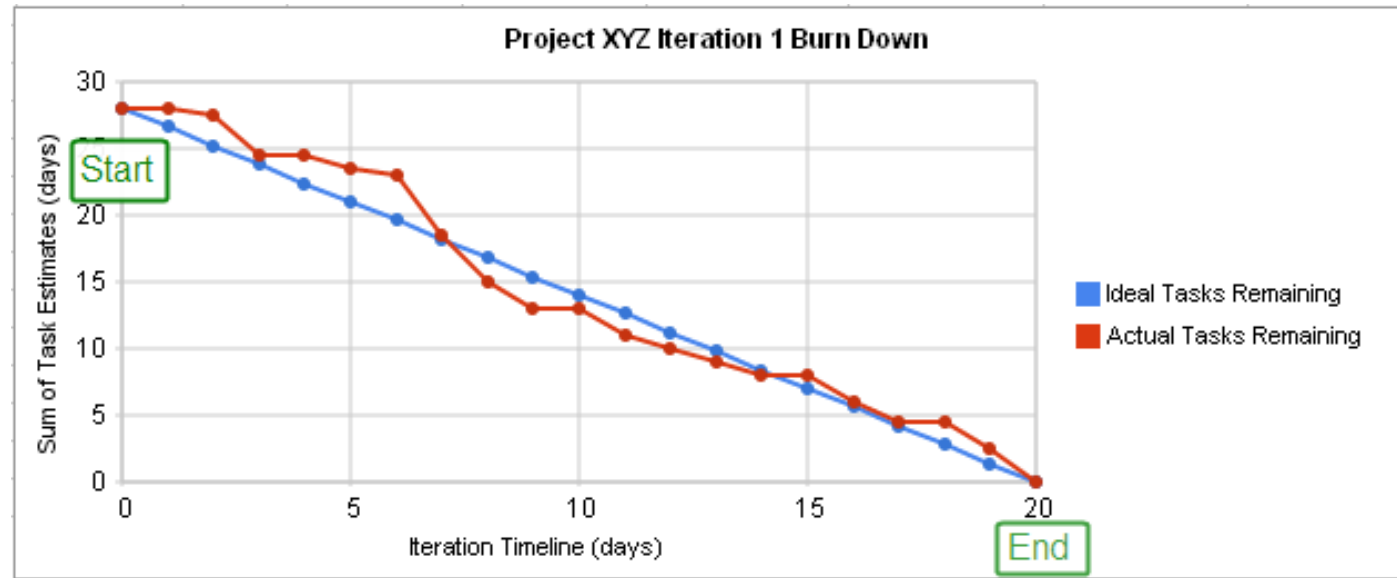
```
$ git tag -a v0.1 -m 'version after Sprint#1 without US #2'
```



Side note:

When assigning tickets to devs it's helpful if usernames are identifiable (or there is some info on the profile).
"Who is schnuckihasi25 again?"

Scrum Burn-Down Chart



- Graphical representation of work left to do versus time
- X-Axis: sprint timeline, e.g. 10 days
- Y-Axis: work that needs to be completed in sprint (time or story points)
- "Ideal" work remaining line: straight line from start to end
- Actual work remaining line
 - above ideal → behind schedule, below ideal → ahead schedule

Scrum Boards - Virtual vs. Real-Life



Definition of Done



How do I know when to stop?

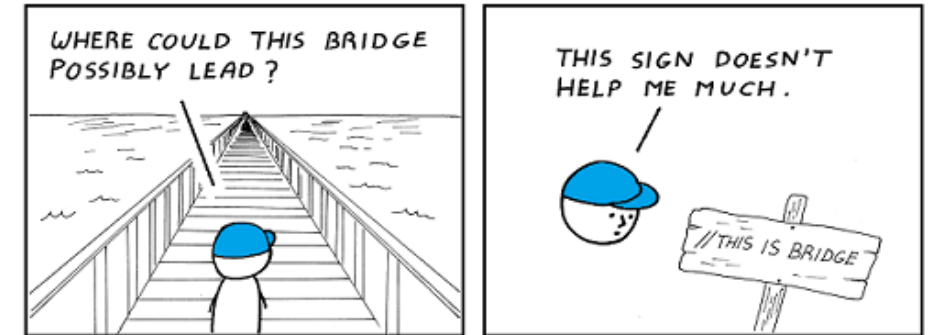
- Acceptance criteria fulfilled
- All tests are green
- Code looks good
- Objective quality goals
- Second opinion
- Internationalization
- Security
- Documentation

The Definition of Done is the team's **consensus of what it takes to complete a feature.**

Communication

Questions:

- How do we communicate in and between teams?
- How do I find out about architecture changes?
- How do I know how to use other people's code?

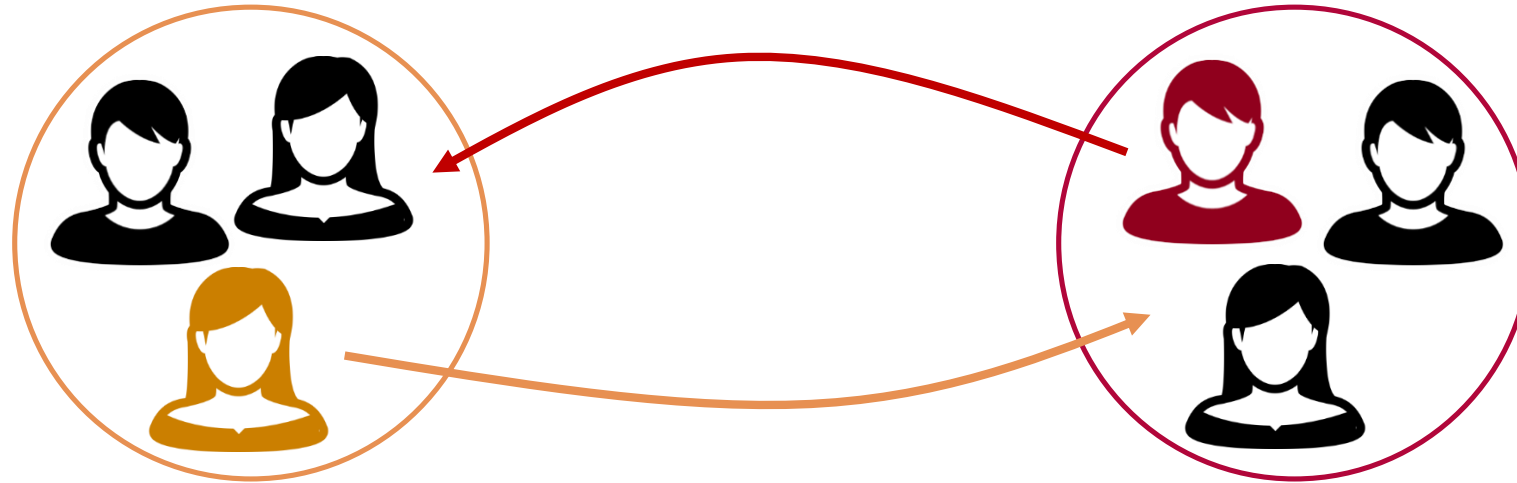


Tools that might help:

- Github wiki to (briefly!) document how to use components
- Code comments explaining the larger context, common pitfalls
- One(!) common communication channel for announcing changes, e.g. E-Mail list, IRC, IM, Slack, Google Hangouts, Facebook group

Dealing with Dependencies

Ambassadors



- Mutual Exchange of team members
 - Improves efficiency of communications
 - Allows deeper understanding of problems
 - Prevents coordination problems early in the process
- Ambassadors should be fully integrated team members
- Especially useful for API development, design, etc.

Estimating Large Backlogs (1/2)



Bucket Estimation (Jukka Lindström) [Scrumcenter, 2009]

- Create physical buckets based on examples (2-3 per bucket)
- Assign items to buckets one by one through
 - Comparing & discussing
 - Planning Poker

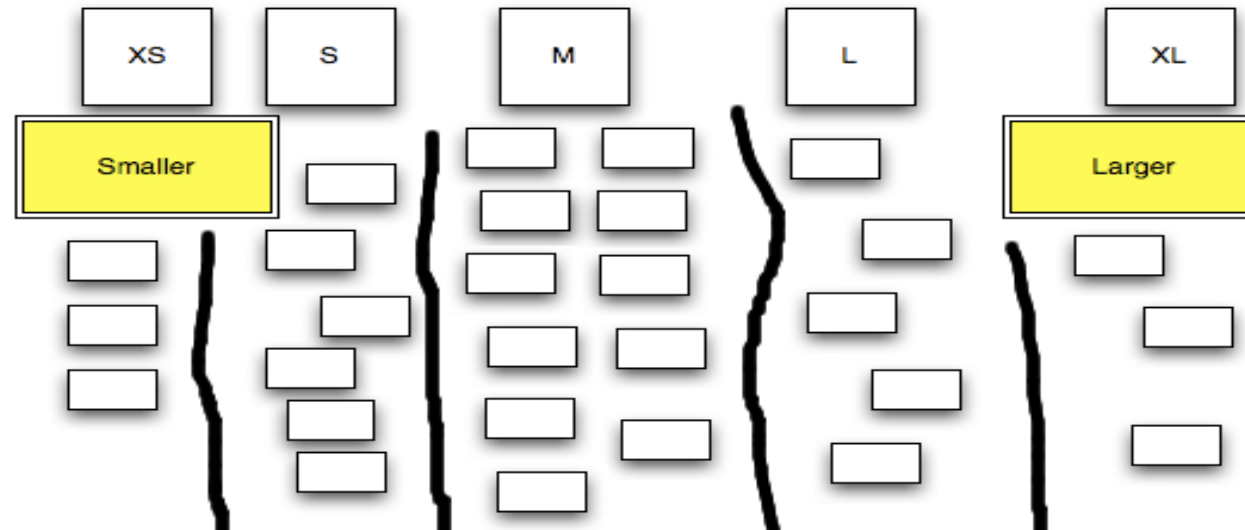


Estimating Large Backlogs (2/2)



Affinity Estimation (Lowell Lindstrom) [Scrumcenter, 2009]

- Read each story to the entire team
- Arrange stories horizontally based on size (no talking!)
- Place Fibonacci numbers above the list
- Move each story to the preferred number



Scrum critique:

- Scrum and agile are by no means universally accepted as "the way" to do software engineering ("Agile Hangover")
- Michael O. Church - *Why "Agile" and especially Scrum are terrible (2015)*
<https://michaelochurch.wordpress.com/2015/06/06/why-agile-and-especially-scrum-are-terrible/>
 - *Business-driven engineering* — Scrum increases the feedback frequency while giving engineers no real power
 - *Terminal juniority* — Architecture and R&D and product development aren't part of the programmer's job
 - *It's stupidly, dangerously short-term* — engineers are rewarded or punished solely based on the completion, or not, of the current two-week "sprint"

Agenda



1. Value-based Requirements Analysis
2. Organizing your Project
3. Git Tricks
4. Outlook

Git Tricks – amend, interactive staging



Change commit message of previous commit
(Careful, don't do this if you already pushed the commit)

```
$ git commit --amend -m "new message"
```

Forgot to commit files?

```
$ git add [missing files]
$ git commit --amend #uses the previous commit's message
```

Undo the amending

```
$ git reset --soft HEAD@{1}
$ git commit -C HEAD@{1}
```

Interactive staging (also allows committing only parts of files)

```
$ git add -i
$ git add --patch [file]
```

A yellow sticky note with two red pushpins is pinned to the right side of the slide. It contains an opinion about interactive staging.

Opinion:
Interactive staging
(`git add -i`)
is probably the most
powerful git feature
you're not using yet.

Git Tricks – reflog, diff, stash



Log of all recent actions

```
$ git reflog
```

What did I work on recently?

Show differences that are not staged yet

```
$ git diff
```

Shows differences between staging and the last file version

```
$ git diff --staged
```

Temporarily store/retrieve all modified tracked files

```
$ git stash
```

```
$ git stash pop
```

List all stashed changesets

```
$ git stash list
```



Tip:

git stash is often helpful if you don't want to directly commit your changes, but need to checkout another branch/commit.

Git Tricks – log, blame, rebase



Shorter version of the git log

```
$ git log --abbrev-commit --pretty=oneline
```

Show pretty graph of git history

```
$ git log --graph --decorate --pretty=oneline --abbrev-commit
```

Show changesets in the log

```
$ git log -p
```

Show what revision and author last modified each line

```
$ git blame --date=short [file]
```

History is becoming cluttered with merge commits

```
$ git rebase <branch>
```



Warning:

Do not rebase commits that others have worked with!

"people will hate you, and you'll be scorned by friends and family."

<https://git-scm.com/book/en/v1/Git-Branching-Rebasing#The-Perils-of-Rebasing>

Git Rebase – setup

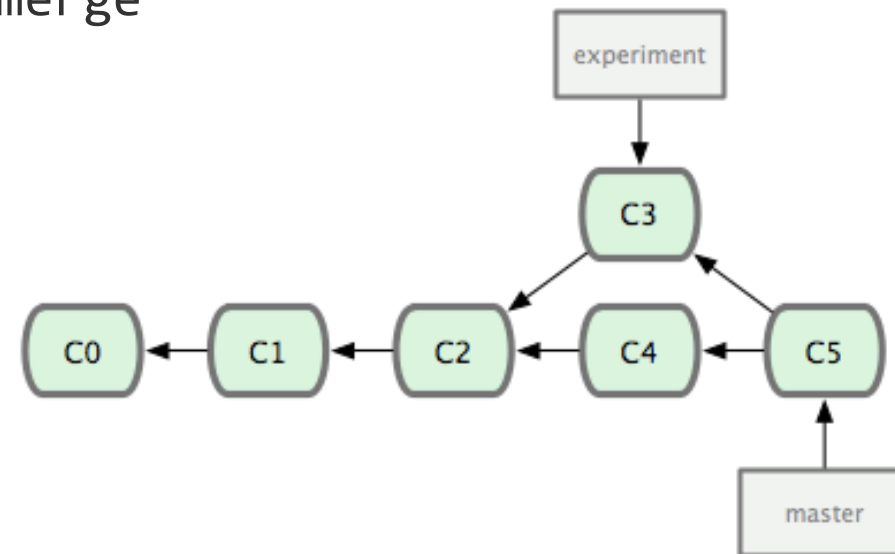
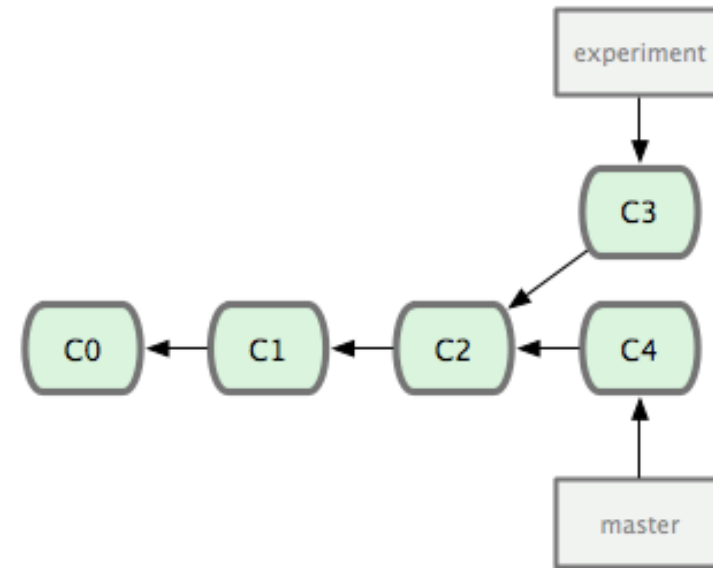


- Created "experiment" branch to try something out

```
$ git checkout -b "experiment"  
$ git commit -a -m "C3"
```

- Easiest way to integrate the branches is merge
 - Will create merge commits

```
$ git checkout master  
$ git merge experiment
```



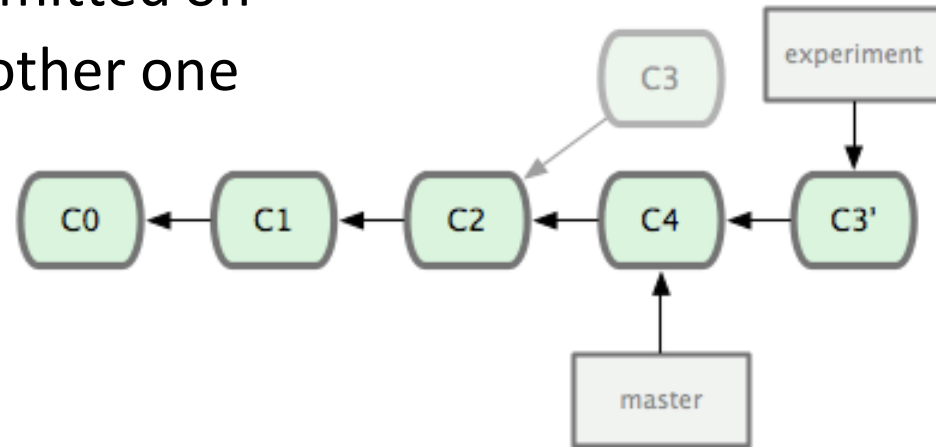
Git Rebase – execution



- git rebase

- Take all the changes that were committed on one branch and replay them on another one
- Only do this with local commits

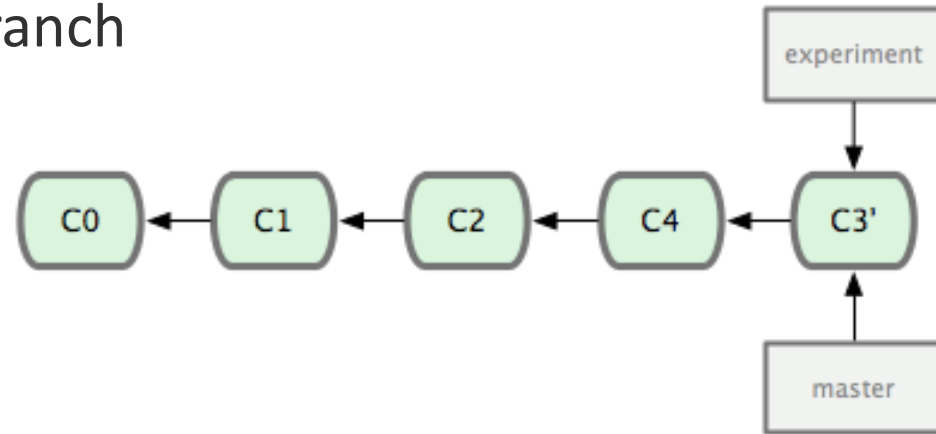
```
$ git checkout experiment  
$ git rebase master
```



- Afterwards: fast-forward the master branch

- No merge commits

```
$ git checkout master  
$ git merge experiment
```



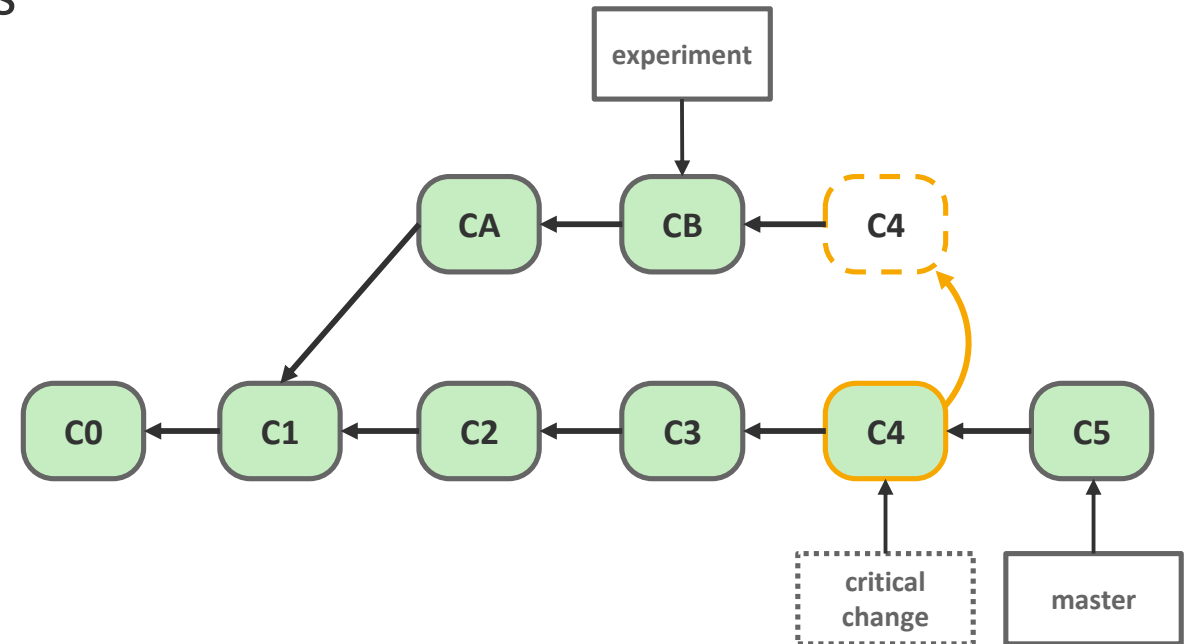
Git cherry-pick



- **Problem:** Quickly get changes from other commits without having to merge entire branches
- `git cherry-pick`
 - apply the changes introduced by existing commits

```
$ git checkout master  
$ git log --abbrev-commit --pretty=oneline  
d7ef34a C3: Implement feature  
0be778a C4: critical change introduced
```

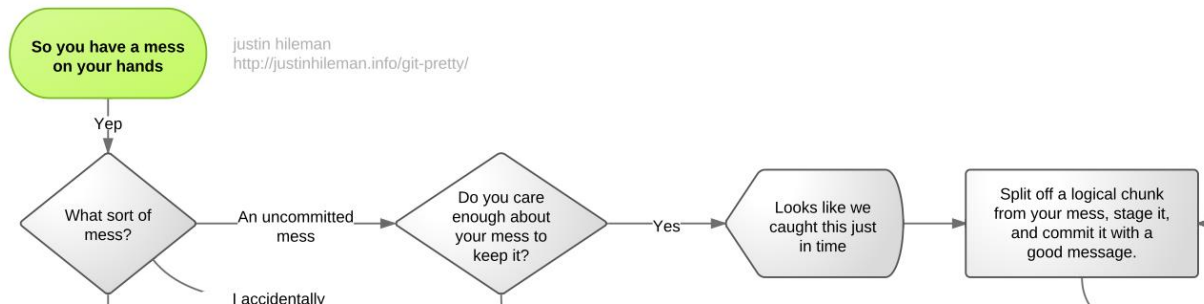
```
$ git checkout experiment  
$ git cherry-pick 0be778a
```



Git Self-help Resources



- How to undo (almost) anything with git – guide by Github
 - <https://github.com/blog/2019-how-to-undo-almost-anything-with-git> one
- Git cheat sheet – by Github
 - <https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf>
- Git FAQ – answers to common questions
 - <http://gitfaq.org/>
 - https://git.wiki.kernel.org/index.php/Git_FAQ
- Git pretty – troubleshooting flowchart
 - <http://justinhileman.info/article/git-pretty/>



Tooling suggestions



- Many GUIs for git available (<https://git-scm.com/downloads/guis>)
 - Make some complex git interactions much simpler
 - Draw pretty commit graphs, overviews of branches and merges
 - *GitX, TortoiseGit, SourceTree, Tower, SmartGit, gitg, git-cola*
- Github Integration
 - Github also provides git tools
<https://mac.github.com/>, <https://windows.github.com/>
- Git extras (<https://github.com/tj/git-extras>)
 - Common git commands bundled

Agenda



1. Value-based Requirements Analysis
2. Organizing your Project
3. Git Tricks
4. Outlook

Outlook



Dec 18

- Present your intermediate results

Jan 15 & 22

- Guest lectures