EPFL

CS-311: Collaborative Software Development

Prof. George Candea
School of Computer & Communication Sciences

Outline

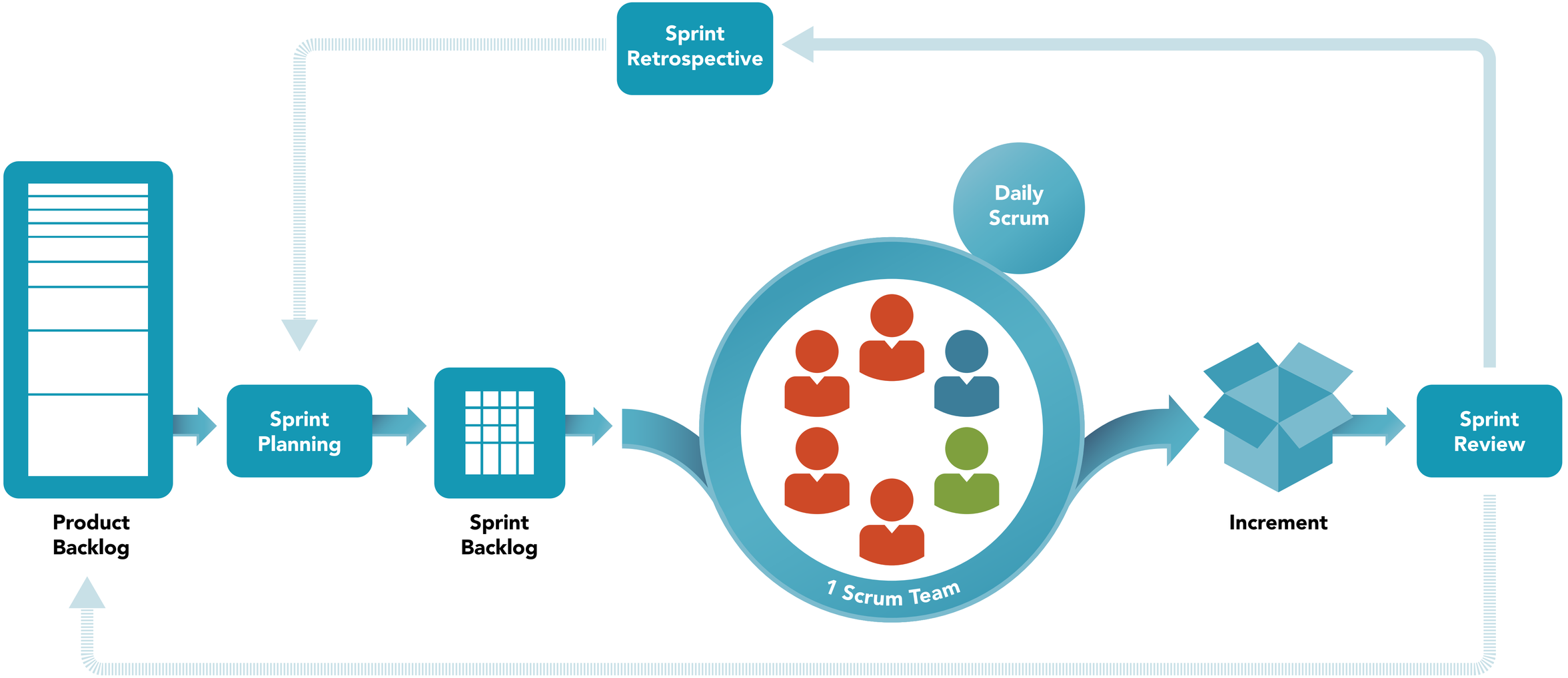
- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

Development Process: Scrum

Scrum Method

- Basic structure = ***Sprint***
  - *1-4 weeks (rigidly fixed length)*
  - *one Sprint after another*
  - *working product at the end of each sprint*
- Cross-functional development teams of 3-10 people
- Meet daily

Scrum Workflow



https://www.scrum.org/resources/scrum-framework-poster

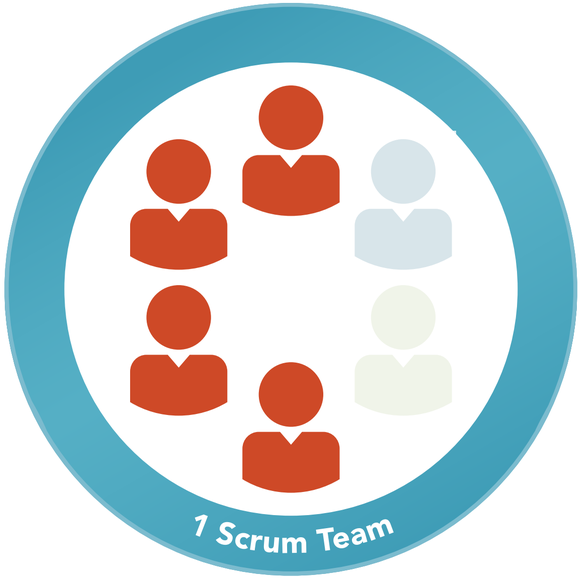
- **Players**
- **Workflow**

Scrum Team

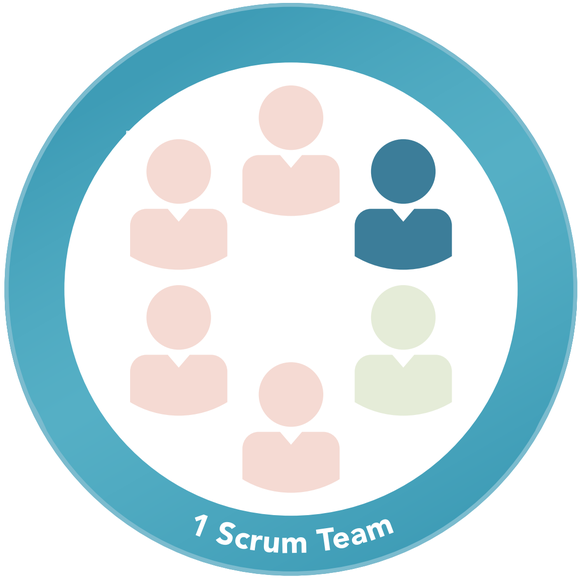
- N Developers +
  1 Product Owner (PO) +
  1 Scrum Master (SM)
- Cohesive unit of ≤10 professionals
  - *small enough to be nimble, big enough to do significant work in one Sprint*
  - *no hierarchy, no sub-teams, self-managing, autonomous → accountable*
  - *for big products, can have multiple Scrum teams with same Goal, Backlog, PO*
- Scrum team does "everything"
  - *dev code, test, maintain, research, stakeholder collaboration, etc.*

Scrum Team: Developers

- Write the code, build the product, maintain, devops, etc.
- Cross-functional
  - *developers, UX designers, testers, ...*
  - *≤ 8 developers on a Scrum team*
- Accountable for
  - *creating the Sprint Backlog and adapting the plan during the Sprint*
  - *keep focus on Sprint Goal, produce Increment*
  - *hold each other to high professional standards*

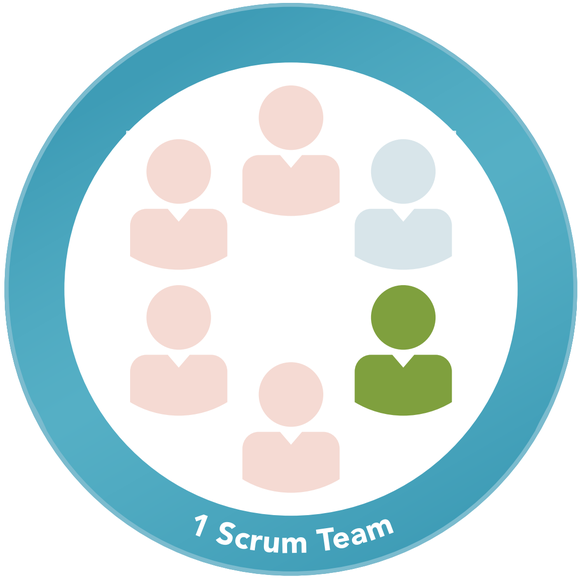


Scrum Team: Product Owner



- Represents stakeholders' interests
  - *sort of like a traditional product manager*
  - *translates stakeholders' needs into a priority list*
  - *thinks about user stories, gathers requirements, etc.*
- PO = one single person
- Accountable for
  - *maximizing business value of Team's product*
  - *managing the Product Backlog:*
    creating PB items, communicating them, prioritizing them, etc.

Scrum Team: Scrum Master



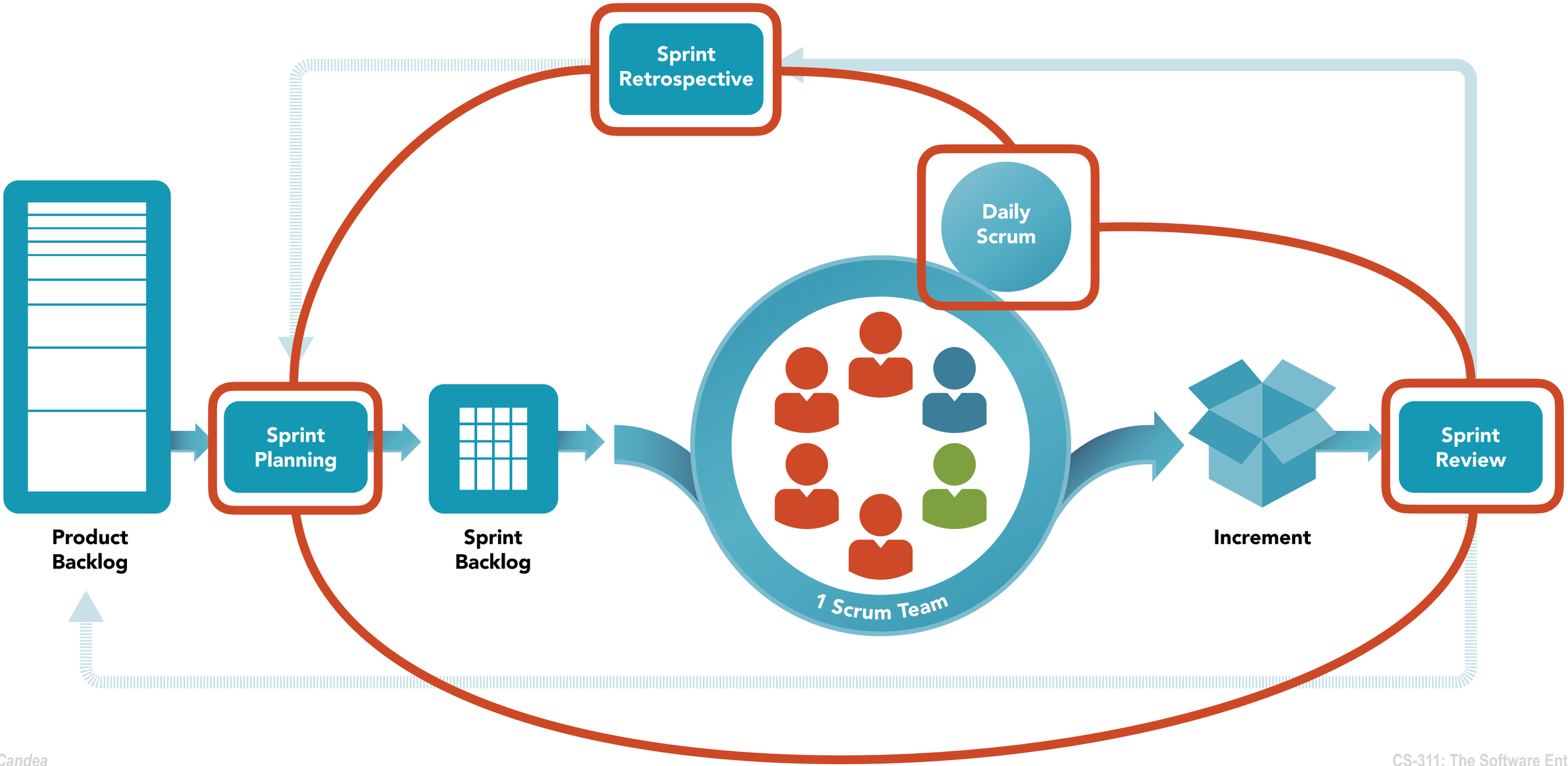
- Accountable for team's success and effectiveness
  - *a facilitator and enabler, not a manager*
- Help developers
  - *coach on how to self-manage and cross-function*
  - *focus on Sprint Goal + protect devs from outside interference*
  - *help remove impediments*
  - *make sure Sprint events take place, are productive, and properly timeboxed*
- Help PO
  - *manage PB, collaborate w/ stakeholders, plan product, ...*

- Players
- **Workflow**

Sprint

- Pick Sprint duration (1-4 weeks) and stick with it for the entire project
  - *think of each Sprint as a mini-project*
  - *helps team improve their estimates*
- Sprint is never extended
  - *if goals not meet, team must own up to it*
  - *takes some experience to determine how long tasks will take*
    → over time, team gets better at it

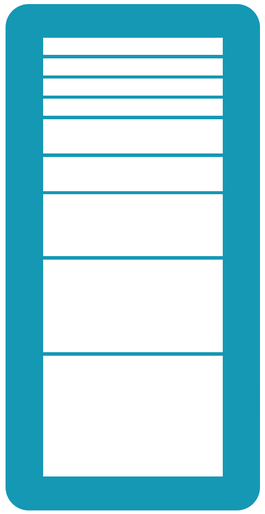
Scrum Events



https://www.scrum.org/resources/scrum-framework-poster

Product Backlog (PB)

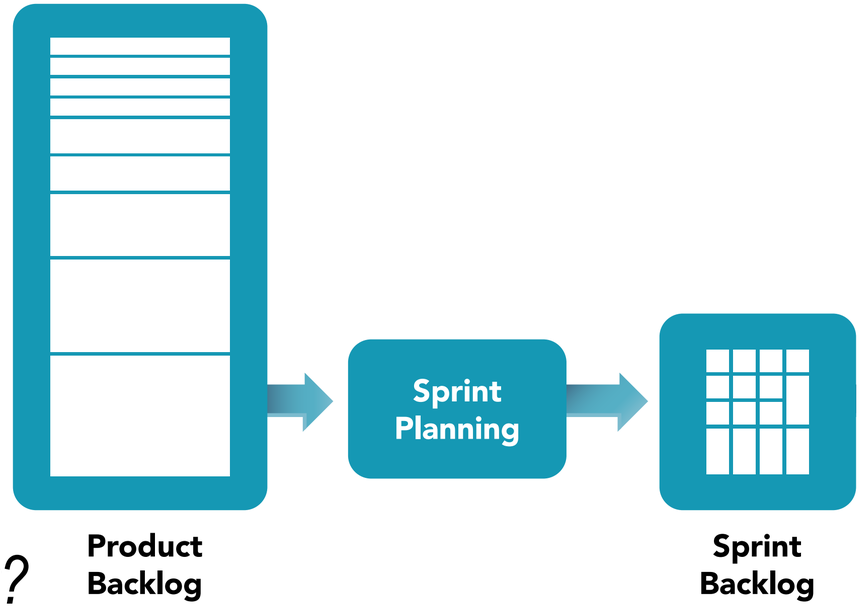
- Product Goal = product vision (by PO)
  - *how will it provide value and to whom*
- Rest of PB:
  - *list of user stories, research tasks, bug fixes, etc.*
  - *prioritized by value to stakeholders*
- Evolves over time (change is a given!)
- Team provides time estimates in person-hours/days
  - *PO uses time estimates as input in prioritization of PB*



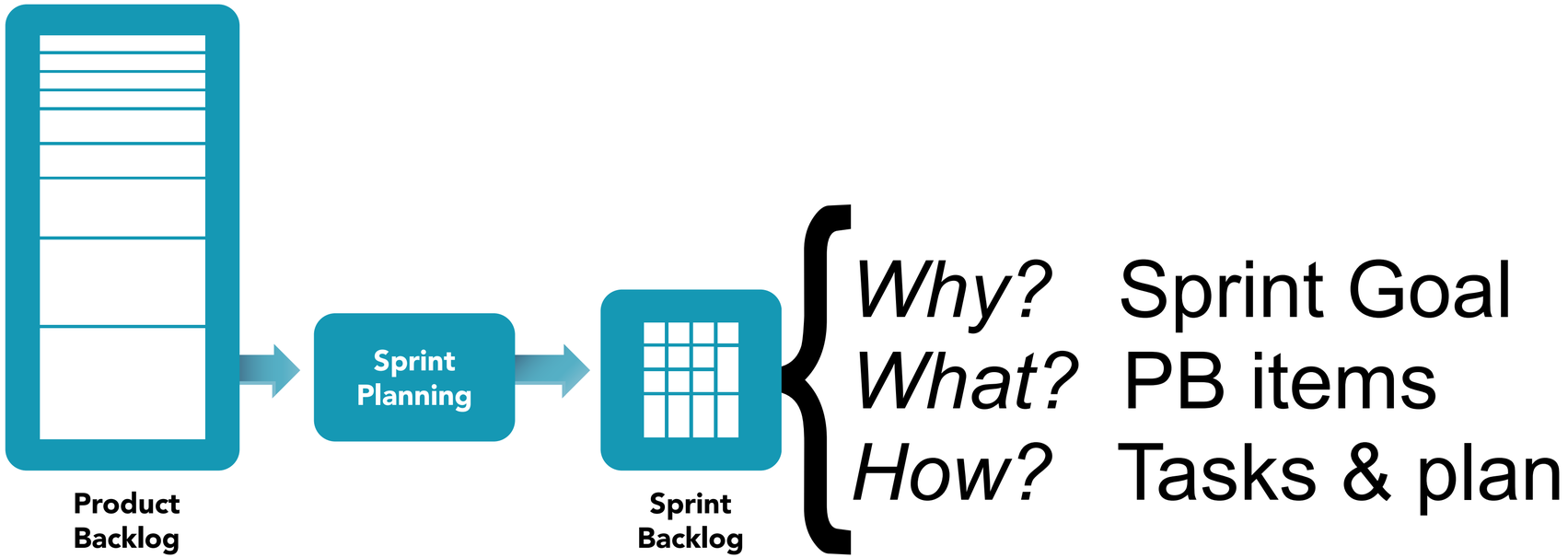
Product Backlog

Scrum Events: Sprint Planning

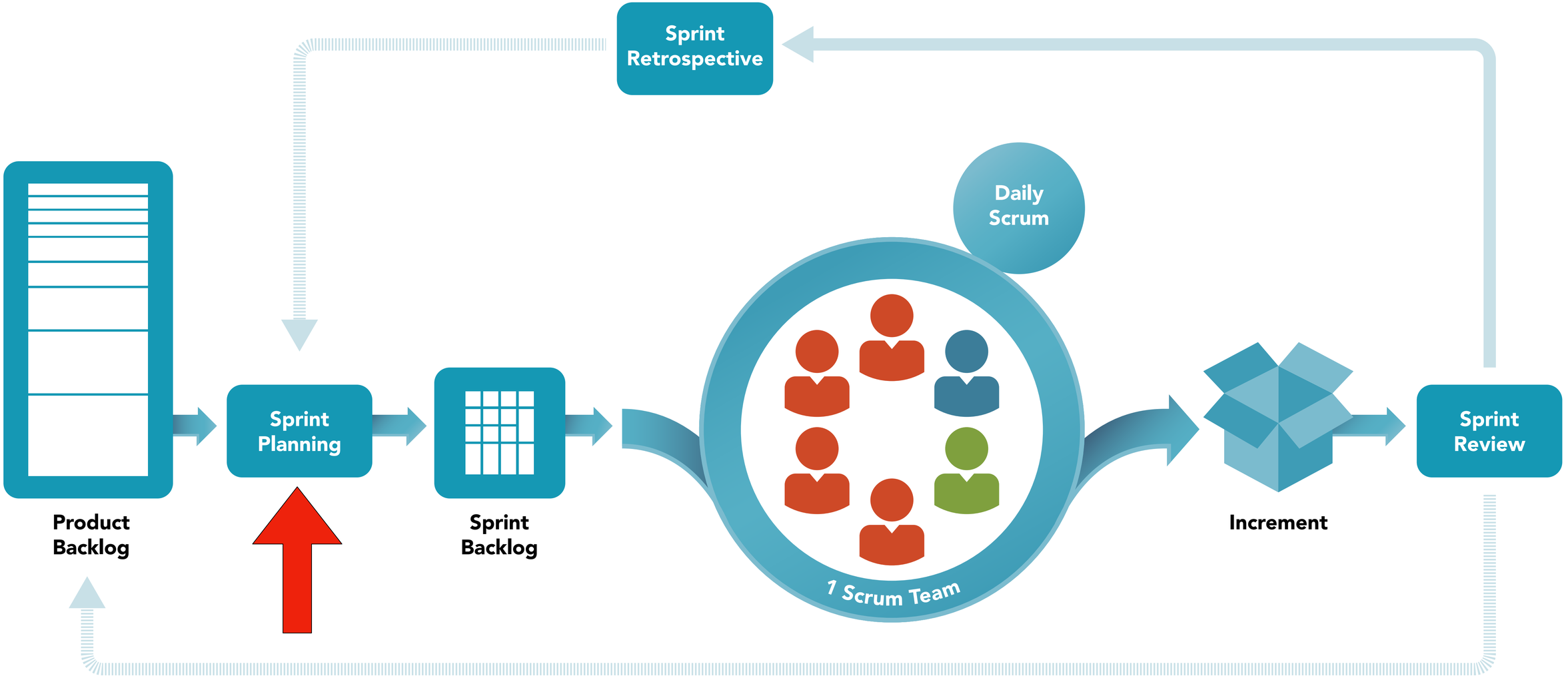
- Attendees: Scrum team (devs + PO + SM)
  - *can invite others to provide advice (e.g., tech experts)*
- Three topics
  - *Why is this Sprint valuable? → leads to Sprint Goal*
  - *Which items at the top of the PB can be done this Sprint?*
  - *How to create an Increment: turn each claimed PB item into tasks*
- Gives team insight into the thinking of the customer
- Once Team has committed, no more changes to PB



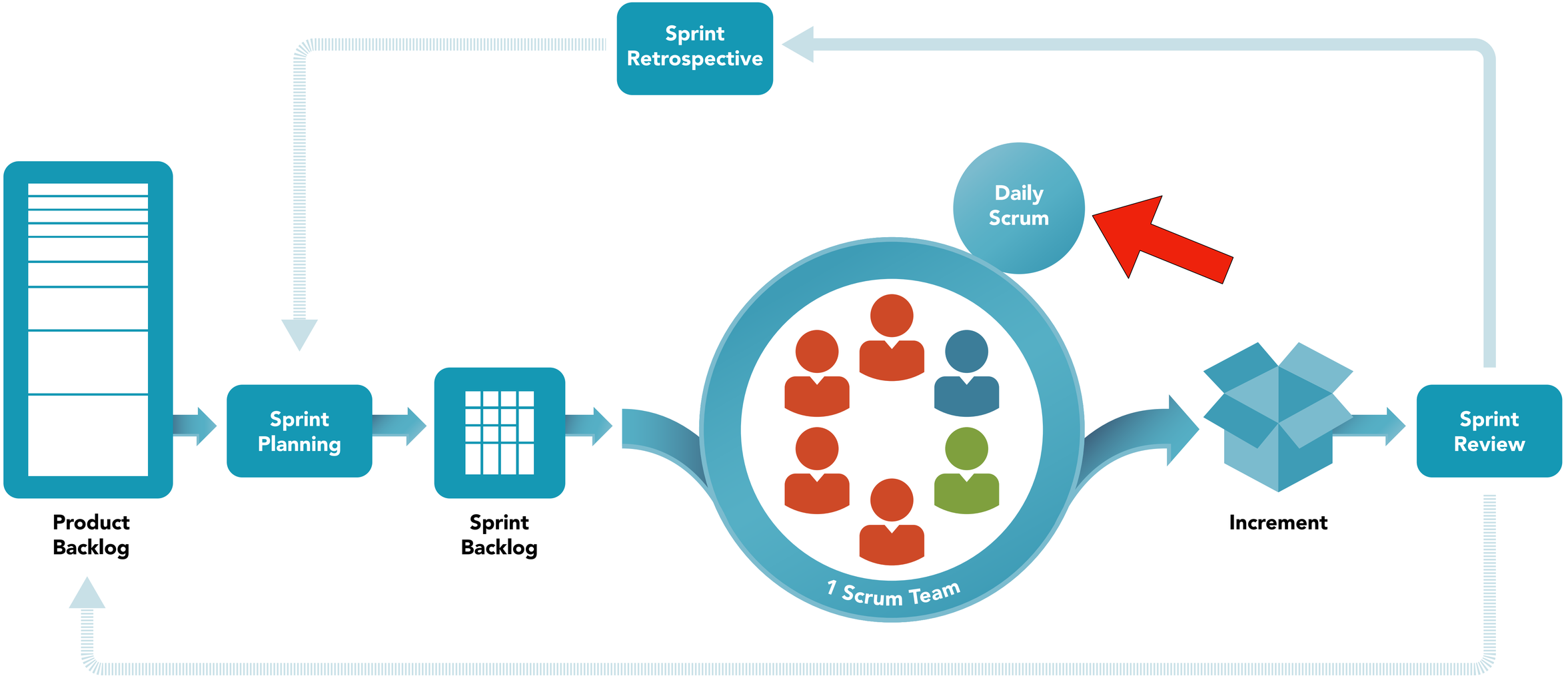
The Sprint Backlog



Scrum Events

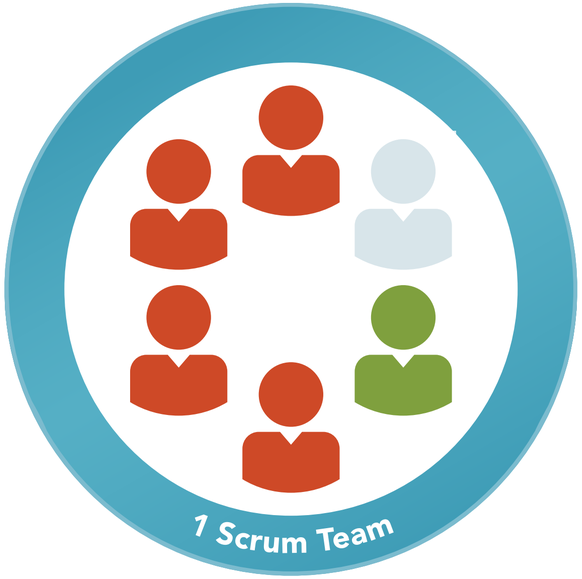


Scrum Events

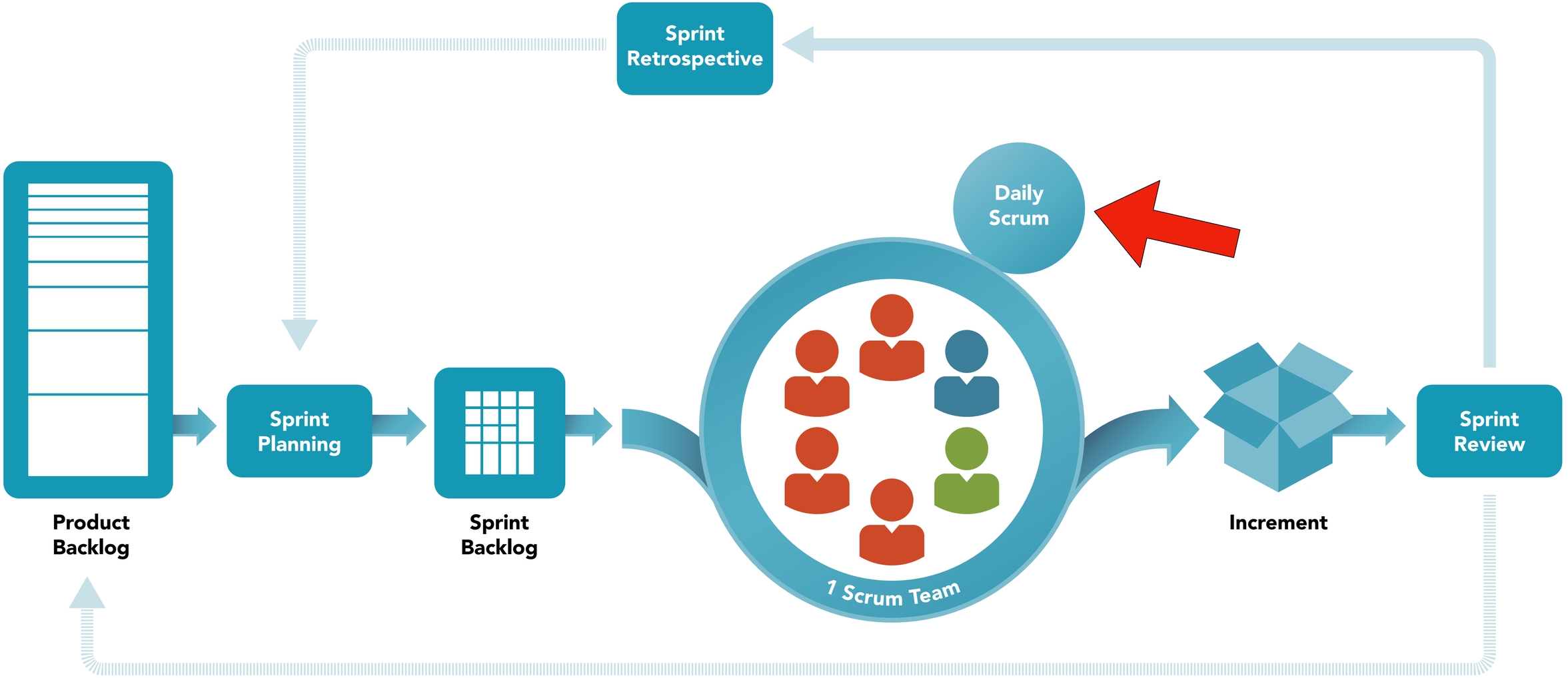


Scrum Events: Daily Scrum

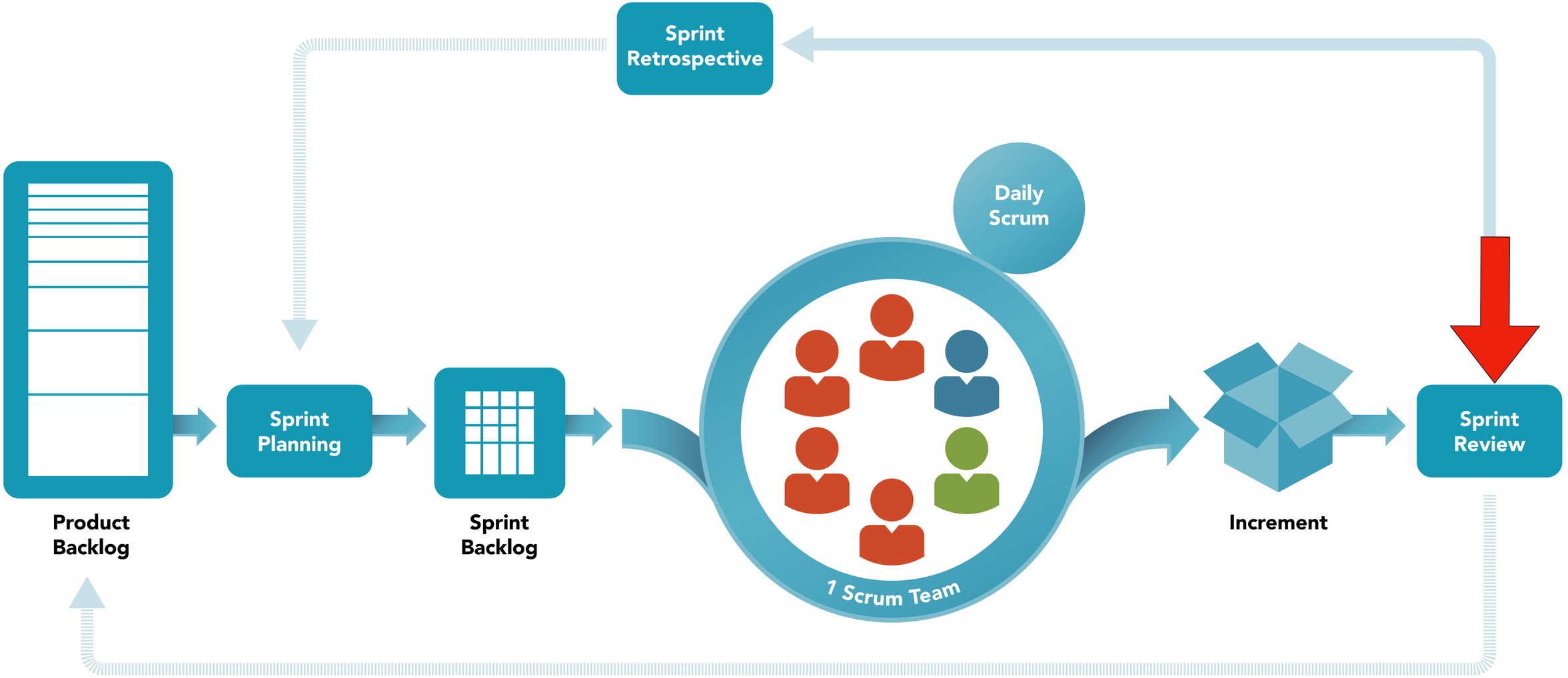
- Attendees: Developers + Scrum Master
  - *at same time, same place every day of the Sprint*
  - *meeting lasts <= 15 minutes, everyone stands*
- Each developer reports 3 items (no discussion!):
  - *What (s)he has done since last meeting*
  - *What (s)he will do until next meeting*
  - *Any blocks or impediments*
- After meeting
  - *Scrum Master resolves impediments and updates progress metrics*



Scrum Events



Scrum Events



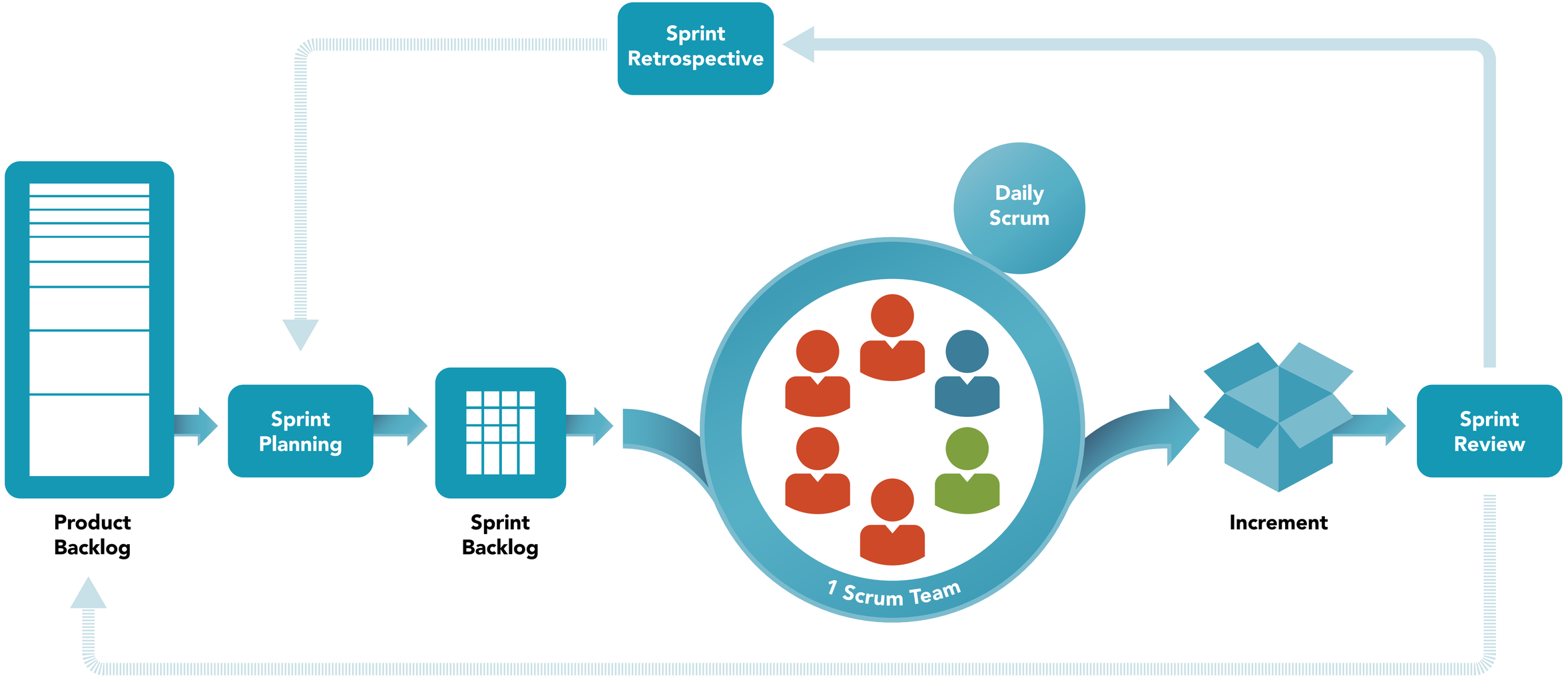
Scrum Events: Sprint Review

- Attendees: Scrum Team + Stakeholders
- Demo the Sprint Increment
  - *stakeholders inspect the outcome of the Sprint*
  - *determine future adaptations*

Scrum Events: Sprint Retrospective

- Attendees: Scrum Team
  - *may be facilitated by a neutral outside (e.g., other SM)*
- Identify and plan ways to increase quality and effectiveness
  - *what went wrong? what worked well?*
  - *how can we improve?*
- PO updates PB based on Review & Retrospective

Scrum Workflow



https://www.scrum.org/resources/scrum-framework-poster

https://scrumguides.org/scrum-guide.html

Resources

- https://agilepainrelief.com/blog/scrum-by-example.html
- Wrong way of doing things...
  - *https://www.youtube.com/watch?v=I1yWusiaLCM (Requirements & Specifications)*
  - *https://www.youtube.com/watch?v=oLmDe8pAc6I (Stand-Ups)*
  - *https://www.youtube.com/watch?v=FJezcyKno5k (Retrospectives)*
- *User stories for SwEnt*
  - *https://www.youtube.com/watch?v=apOvF9NVguA (Creating User Stories)*
- *Tutorial (with Jira, but ignore that) — good preparation for SwEnt*
  - *Part 1: https://www.atlassian.com/agile/tutorials/how-to-do-scrum-with-jira-software*
  - *Part 2: https://www.atlassian.com/agile/tutorials/how-to-do-advanced-scrum-practices-with-jira-software*

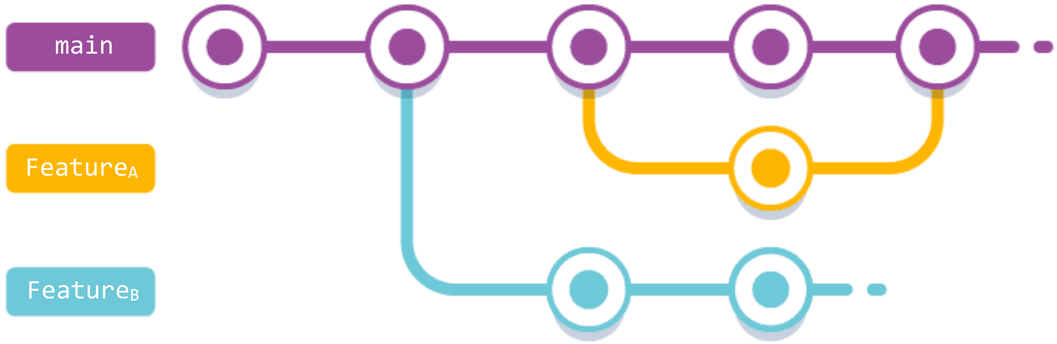
Outline

- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

Collaboration Workflows

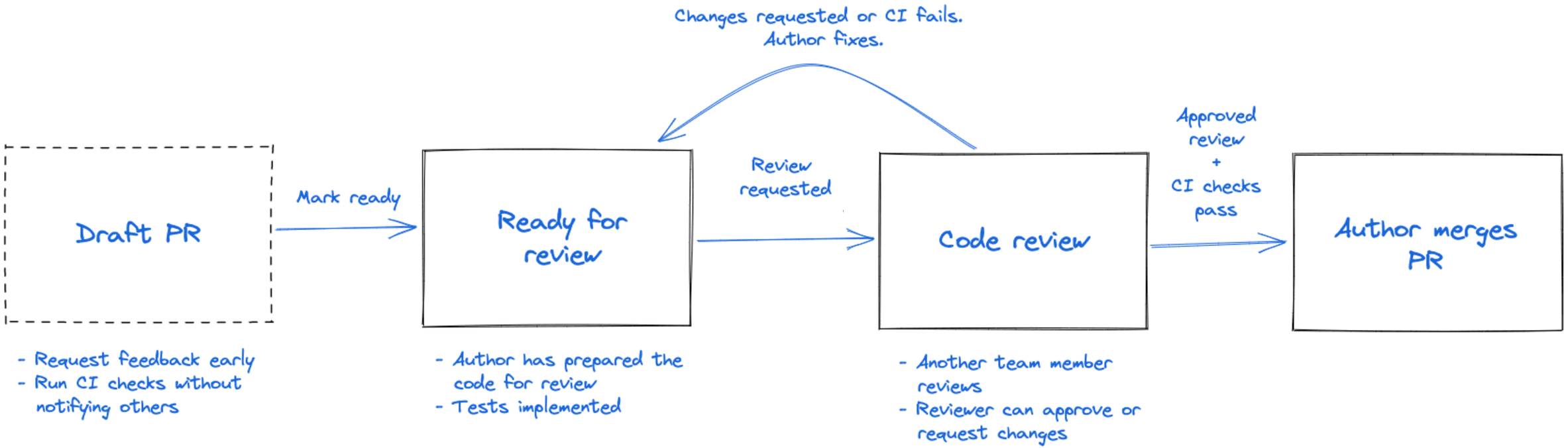
Basic "Feature Branch" Workflow

- Create new branch for each feature/bug
- Commit all related code to this branch
- Regular commits + branch sync-up
- Test → Code review
- Resolve merge conflicts
- Changes approved → merge branch to main
  - *fast-forward merge*
  - *merge commmit*
  - *rebase & merge*
  - *squash merge*
- Delete the feature branch



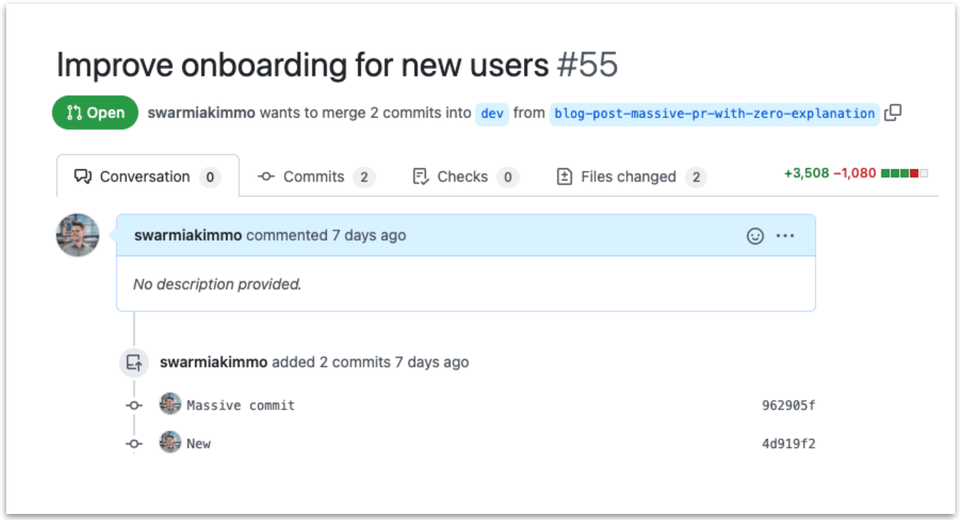
Workflow Overview

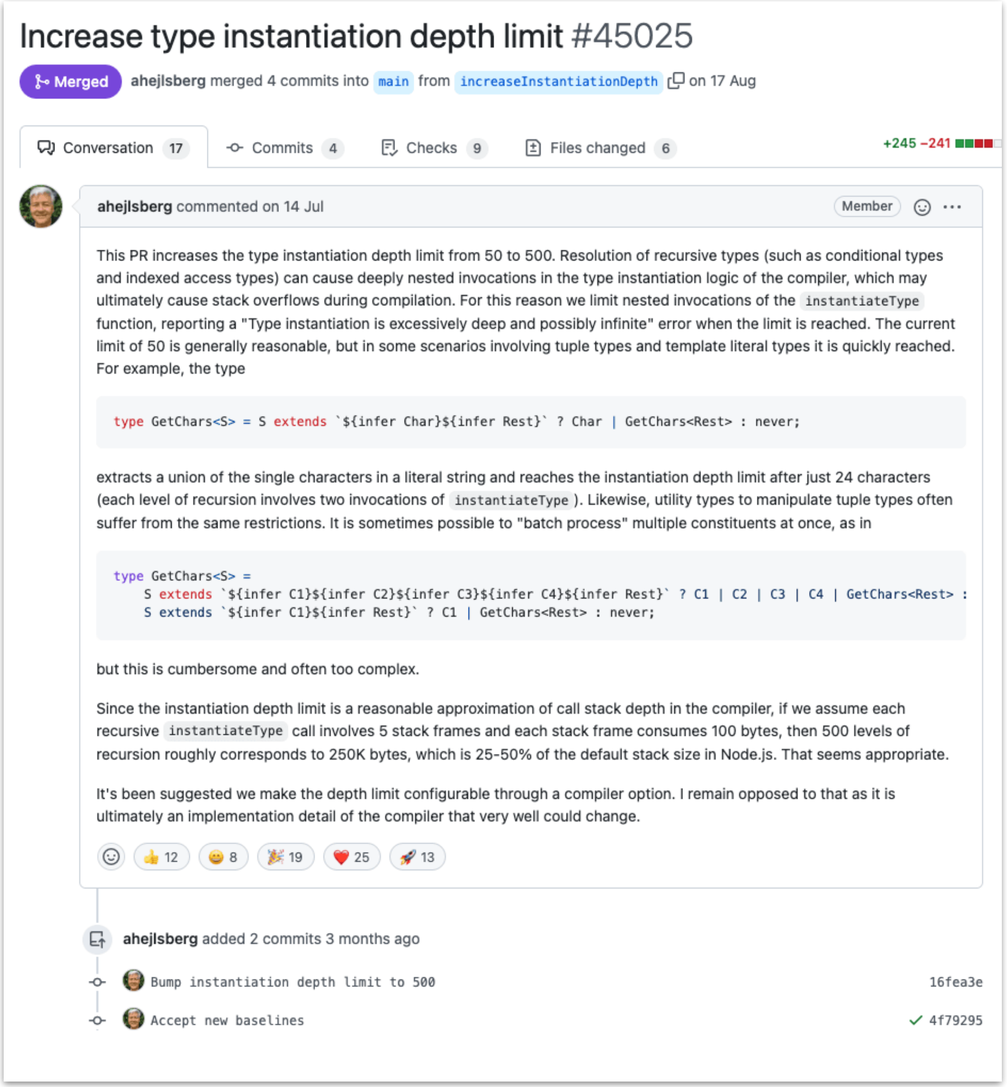
- "Fork & Pull" model vs. "Shared Repo" model



Working with PRs

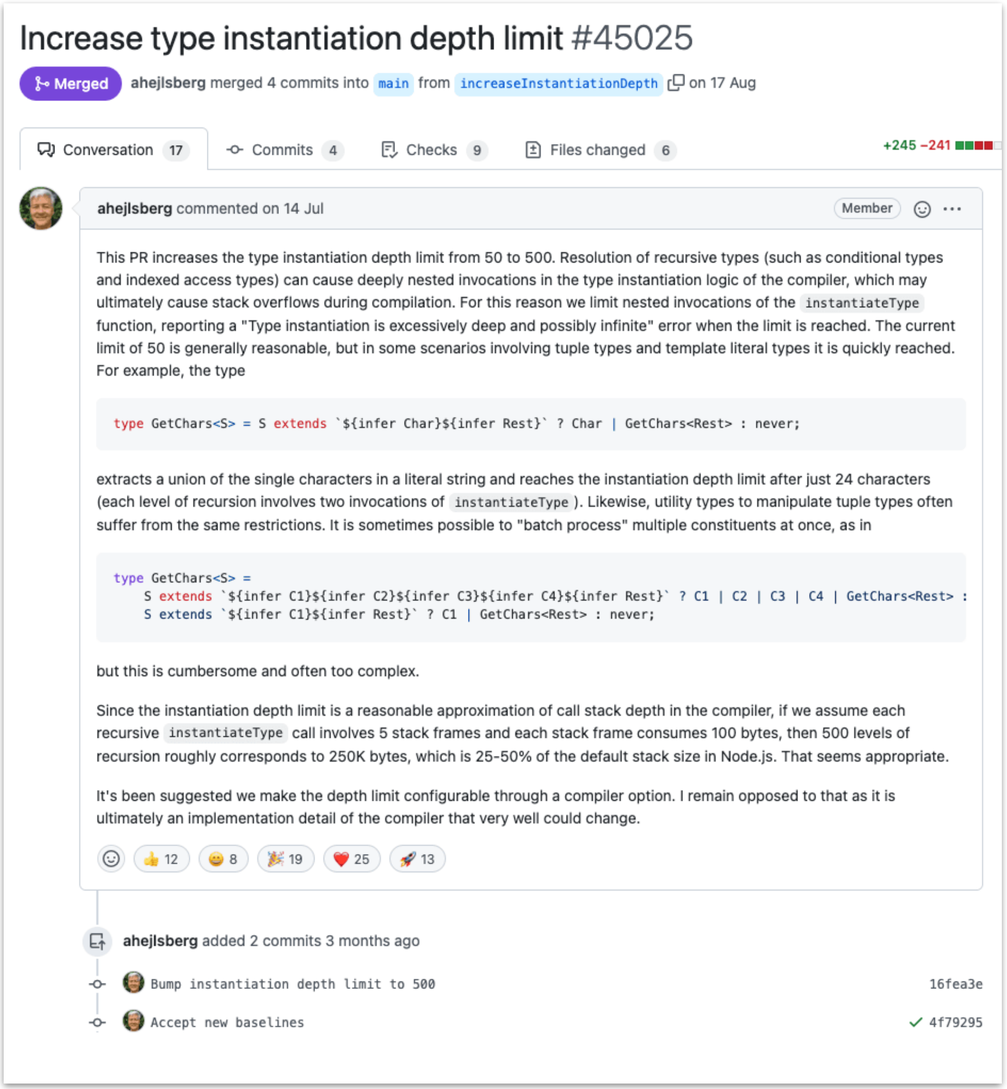
- Provide context for each PR





Working with PRs

- Provide context for each PR
- Link PR to corresponding issue (if any)



Increase type instantiation depth limit #45025

Merged ahejlsberg merged 4 commits into main from increaseInstantiationDepth on 17 Aug

Conversation 17 Commits 4 Checks 9 Files changed 6 +245 −241

ahejlsberg commented on 14 Jul — Member

This PR increases the type instantiation depth limit from 50 to 500. Resolution of recursive types (such as conditional types and indexed access types) can cause deeply nested invocations in the type instantiation logic of the compiler, which may ultimately cause stack overflows during compilation. For this reason we limit nested invocations of the `instantiateType` function, reporting a "Type instantiation is excessively deep and possibly infinite" error when the limit is reached. The current limit of 50 is generally reasonable, but in some scenarios involving tuple types and template literal types it is quickly reached. For example, the type

```
type GetChars<S> = S extends `${infer Char}${infer Rest}` ? Char | GetChars<Rest> : never;
```

extracts a union of the single characters in a literal string and reaches the instantiation depth limit after just 24 characters (each level of recursion involves two invocations of `instantiateType`). Likewise, utility types to manipulate tuple types often suffer from the same restrictions. It is sometimes possible to "batch process" multiple constituents at once, as in

```
type GetChars<S> =
    S extends `${infer C1}${infer C2}${infer C3}${infer C4}${infer Rest}` ? C1 | C2 | C3 | C4 | GetChars<Rest> :
    S extends `${infer C1}${infer Rest}` ? C1 | GetChars<Rest> : never;
```

but this is cumbersome and often too complex.

Since the instantiation depth limit is a reasonable approximation of call stack depth in the compiler, if we assume each recursive `instantiateType` call involves 5 stack frames and each stack frame consumes 100 bytes, then 500 levels of recursion roughly corresponds to 250K bytes, which is 25-50% of the default stack size in Node.js. That seems appropriate.

It's been suggested we make the depth limit configurable through a compiler option. I remain opposed to that as it is ultimately an implementation detail of the compiler that very well could change.

👍 12 😄 8 🎉 19 ❤️ 25 🚀 13

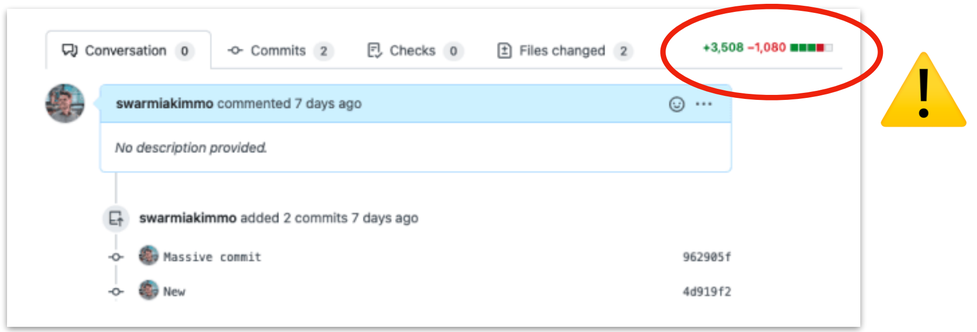
ahejlsberg added 2 commits 3 months ago

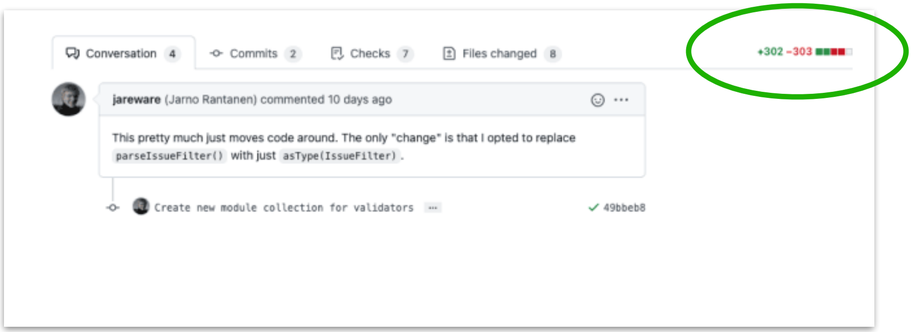
Bump instantiation depth limit to 500 16fea3e

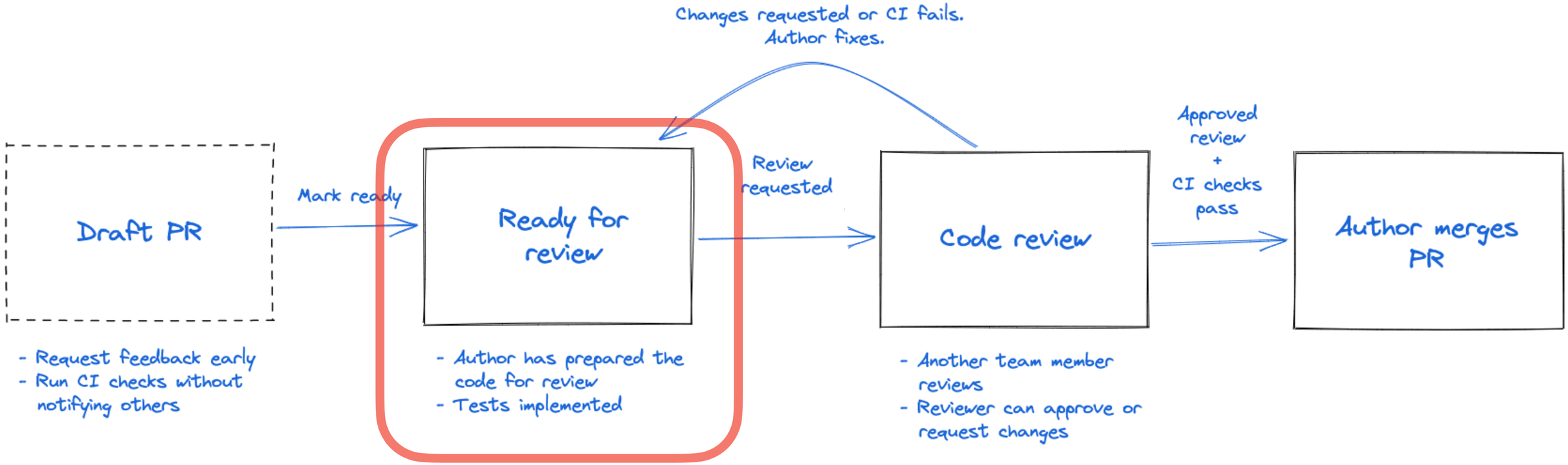
Accept new baselines ✓ 4f79295

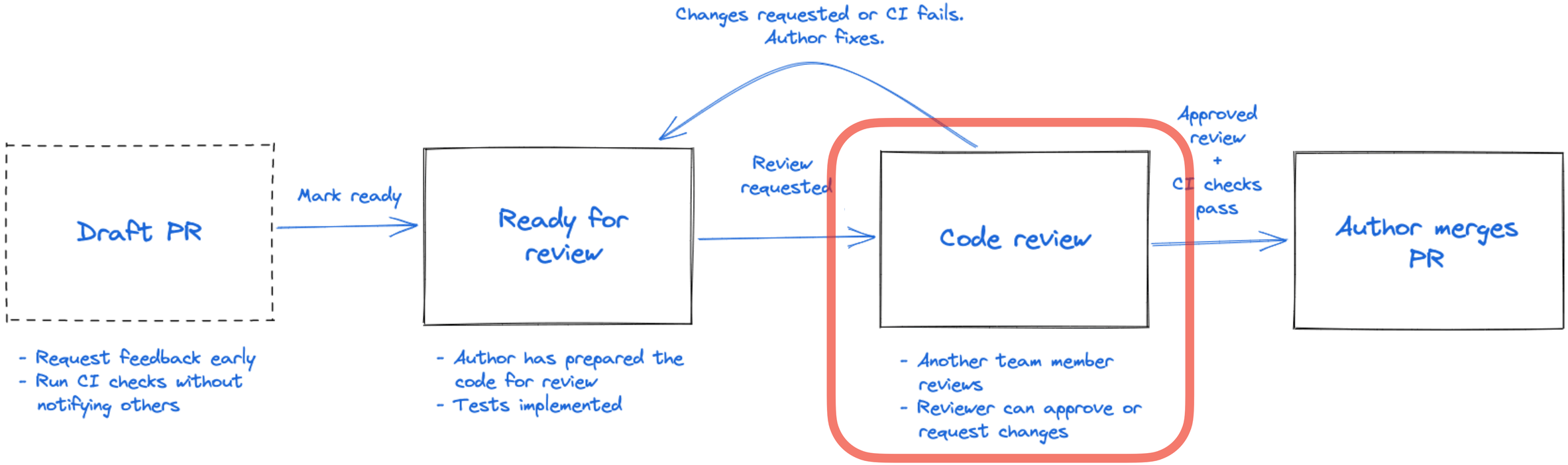
Keep PRs small

- Good for author
  - *higher quality code*
  - *less wasted work if PR is rejected*
  - *easier to merge*
- Good for reviewer
  - *more quickly*
  - *more thoroughly*






Changes requested or CI fails.
Author fixes.
Draft PR
Mark ready
Ready for review
Review requested
Code review
Approved review + CI checks pass
Author merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes


Changes requested or CI fails.
Author fixes.
Draft PR
Mark ready
Ready for review
Review requested
Code review
Approved review + CI checks pass
Author merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes

Getting your PR Reviewed

- First review it yourself !
- Pick PR reviewers with care
  - *author / owner of the code you modified*
  - *peers on your dev team*
  - *experts on the code or on the kind of changes you made*
  - *other stakeholders who may care (and who might cause trouble later)*
- Communicate clearly with the reviewers
  - *define the goals and objectives of the code review*
  - *e.g., review functionality, performance, reliability, maintainability, security, ... ???*

Reviewing The PR

- Develop a checklist
  - *E.g., functionality, design, complexity, naming, comments, documentation*
- Review rate: 200-500 LOC/hour*
  - *Do not review for more than 1h at a time*
- Finish review speedily and decide (Comment/Approve/Request changes)
- Be rational, clear, and explicit
- Don't point out defects but opportunities for improvement

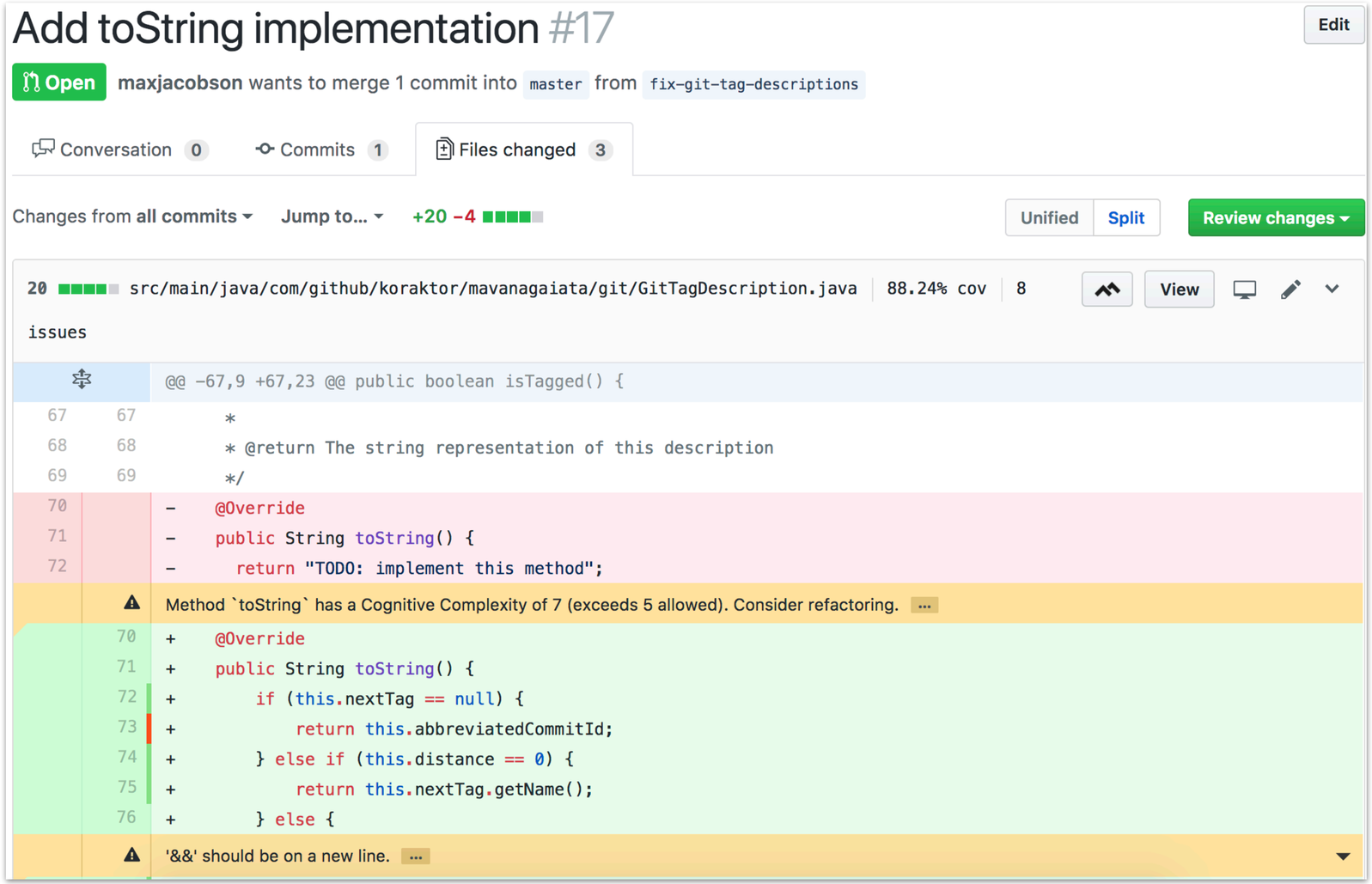
https://www.freepik.com/premium-photo/man-thinking-hand-book-dark-background_9555778.htm

Check out
https://google.github.io/eng-practices/review/reviewer/

* https://static1.smartbear.co/support/media/resources/cc/book/code-review-cisco-case-study.pdf

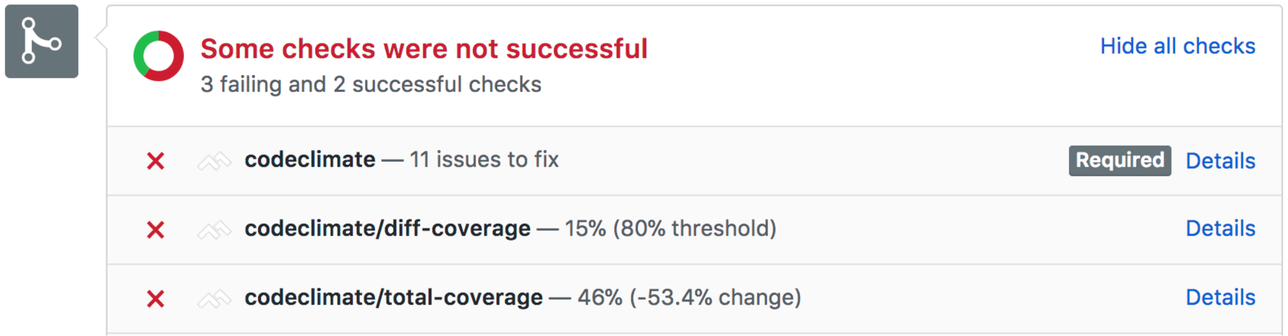
Delegate to Computers Everything that Computers Can Do

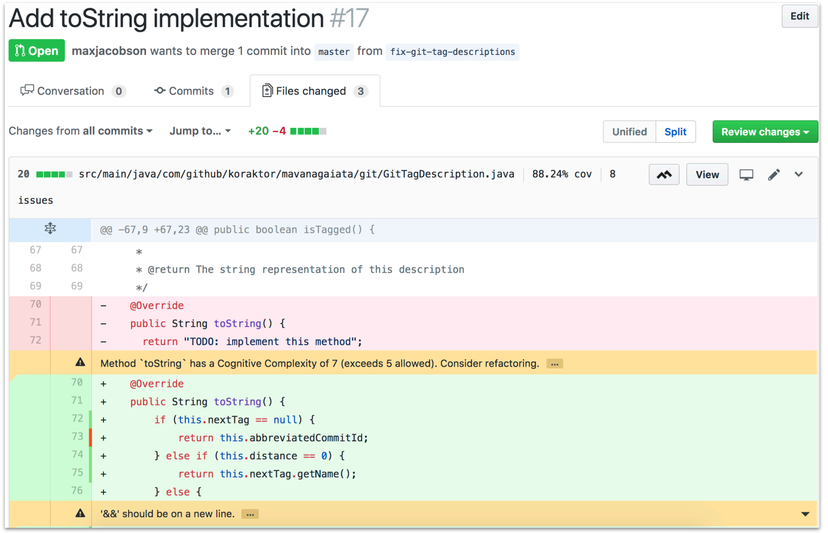
- Use lint tools

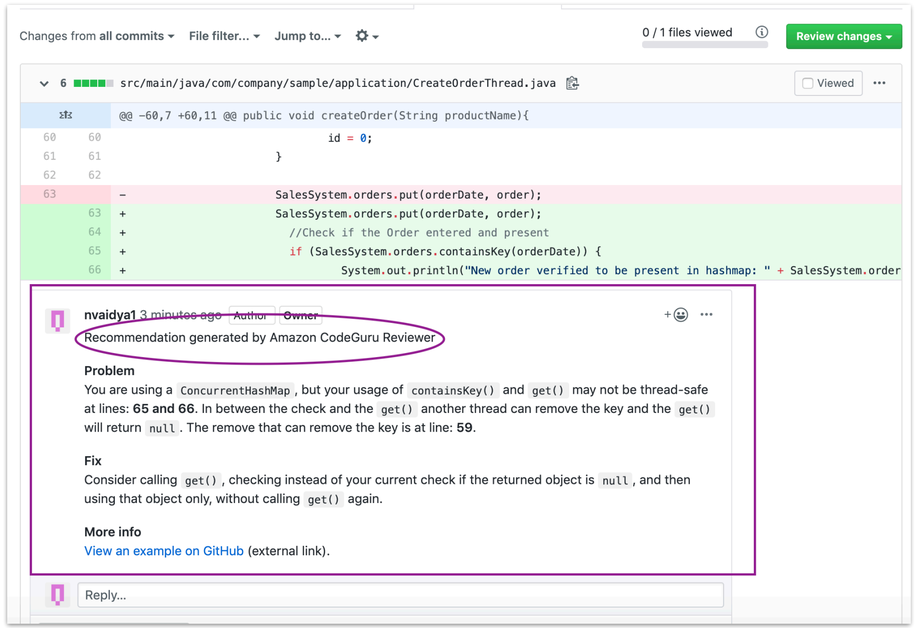


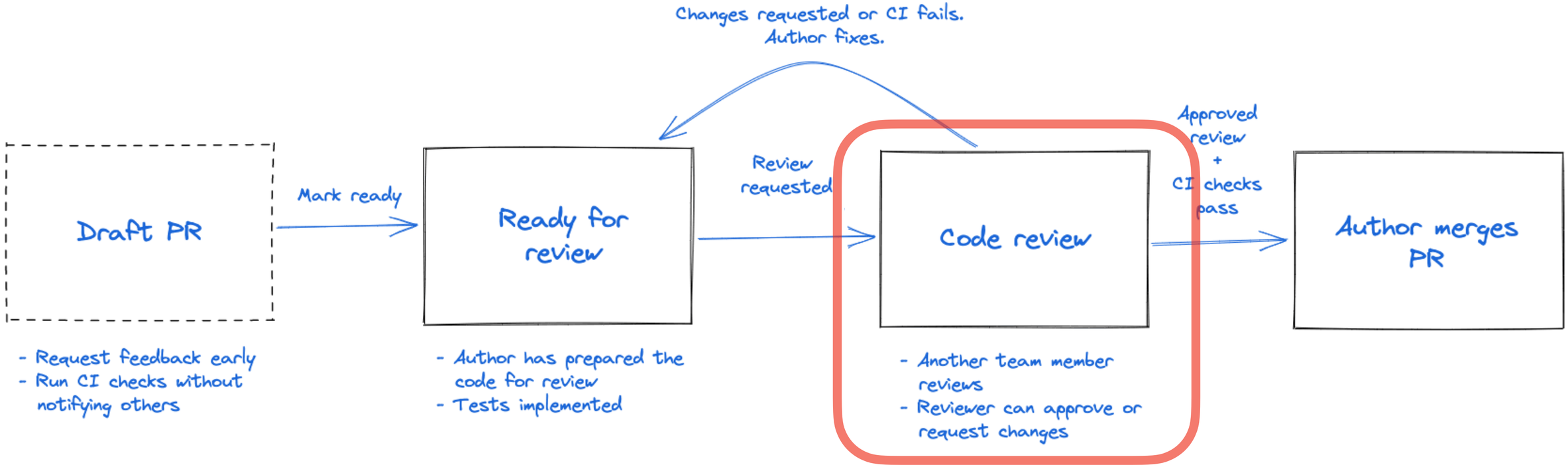
Delegate to Computers Everything that Computers Can Do

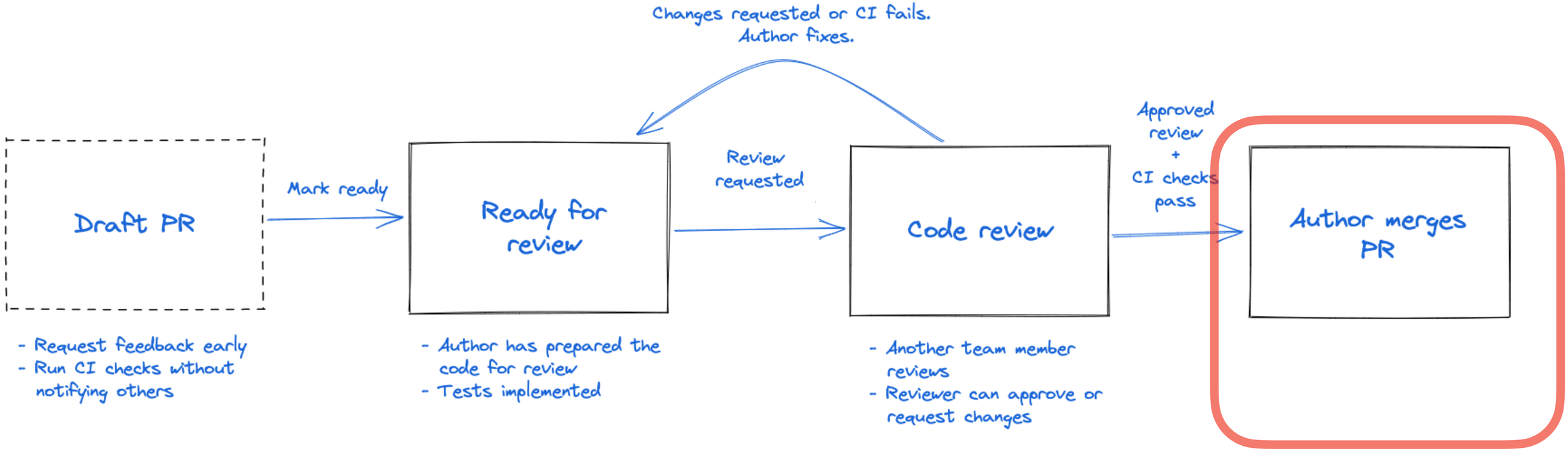
- Use lint tools
- Use static analysis tools
- Integrate with CI

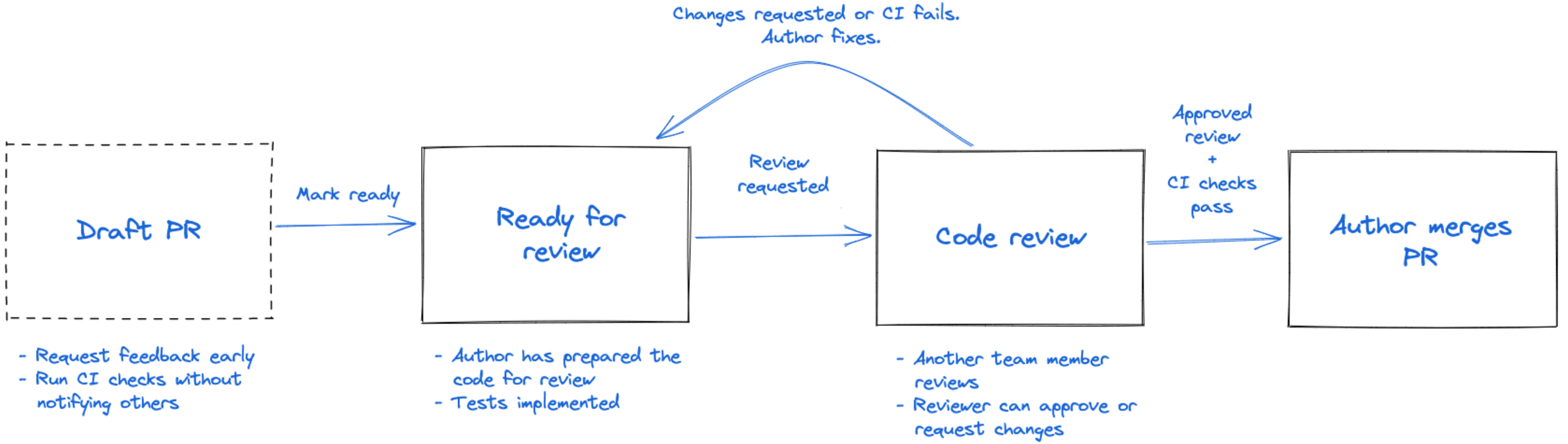







Changes requested or CI fails.
Author fixes.
Draft PR
Mark ready
Ready for review
Review requested
Code review
Approved review + CI checks pass
Author merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes


Changes requested or CI fails.
Author fixes.
Draft PR
Mark ready
Ready for review
Review requested
Code review
Approved review + CI checks pass
Author merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes


Changes requested or CI fails.
Author fixes.
Draft PR
Mark ready
Ready for review
Review requested
Code review
Approved review + CI checks pass
Author merges PR
- Request feedback early
- Run CI checks without notifying others
- Author has prepared the code for review
- Tests implemented
- Another team member reviews
- Reviewer can approve or request changes


main
feature1
feature2
PR
PR

Outline

- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

Writing Good Commit Messages

Typical format

1-line summary

<blank line>

Details

as many

as needed

Log of commits (web version)

Commits on May 4, 2022

Improving the 64-bit number formatting to better match the 32-bit alg... tannergooding committed 12 hours ago ✗	Verified	874c6a9	<>
Update message for unsupported char marshalling (#68865) elinor-fung committed 13 hours ago ✓	Verified	881231a	<>
Add a four-element AsyncLocalValueMap type (#68790) ... stephentoub committed 14 hours ago ✓	Verified	16b6369	<>
Stop escaping ' and " in generated regex XML comments (#68856) stephentoub committed 15 hours ago ✓	Verified	f9a459e	<>
Move `CustomTypeMarshaller` APIs to `System.Runtime.InteropServices.M... elinor-fung committed 15 hours ago ✓	Verified	bcded44	<>
Delete stale IgnoreCaseRelation regex tests (#68857) ... stephentoub committed 15 hours ago ✓	Verified	3929af4	<>
Fix singlefile on OSX ARM64 (#68845) ... VSadov committed 17 hours ago ✓	Verified	d44343b	<>

Log of commits (web version)

Commits on May 4, 2022

Improving the 64-bit number formatting to better match the 32-bit alg... ...
tannergooding committed 12 hours ago ✗ — Verified — 874c6a9

Update message for unsupported char marshalling (#68865)
elinor-fung committed 13 hours ago ✓ — Verified — 881231a

Add a four-element AsyncLocalValueMap type (#68790) ...

```
* Add a four-element AsyncLocalValueMap type

We previously special-cased up to three active AsyncLocals in a given async flow, but it seems four is also very common.  Special-casing four as well results in four using ~20% less allocation and ~10% less CPU overhead.

* Fix downgrading to FourElementAsyncLocalValueMap, and clean up source
```

stephentoub committed 14 hours ago ✓ — Verified — 16b6369

Stop escaping ' and " in generated regex XML comments (#68856)
stephentoub committed 15 hours ago ✓ — Verified — f9a459e

Move `CustomTypeMarshaller` APIs to `System.Runtime.InteropServices.M... ...
elinor-fung committed 15 hours ago ✓ — Verified — bcded44

5 Rules of Commit Messages

- Separate subject from body with a blank line
- Limit the subject line to 50 characters

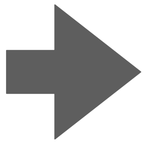
```
$ git log --oneline -5 --author cbeams --before "Fri Mar 26 2019"

e5f4b49 Re-adding ConfigurationPostProcessorTests after its brief removal in r814. @Ignore-ing the testCglibClassesAreLoadedJustI
2db0f12 fixed two build-breaking issues: + reverted ClassMetadataReadingVisitor to revision 794 + eliminated ConfigurationPostPro
147709f Tweaks to package-info.java files
22b25e0 Consolidated Util and MutableAnnotationUtils classes into existing AsmUtils
7f96f57 polishing
```

```
$ git log --oneline -5 --author pwebb --before "Sat Aug 30 2014"

5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

5 Rules of Commit Messages

- Separate subject from body with a blank line
- Limit the subject line to 50 characters
  - *Team-level convention on*
    - *style (markup syntax, wrap margins, capitalization, punctuation) and*
    - *metadata (how to reference issue tracking IDs, pull request numbers, etc.)*
- Use the imperative mood in the subject line
  - *"If applied, this commit will ____"*

```
Refactor subsystem X for readability
Update getting started documentation
Remove deprecated methods
```



```
If applied, this commit will refactor subsystem X for readability
If applied, this commit will update getting started documentation
If applied, this commit will remove deprecated methods
```

5 Rules of Commit Messages

- Separate subject from body with a blank line
- Limit the subject line to 50 characters
  - *Team-level convention on*
    - *style (markup syntax, wrap margins, capitalization, punctuation) and*
    - *metadata (how to reference issue tracking IDs, pull request numbers, etc.)*
- Use the imperative mood in the subject line
  - *"If applied, this commit will ____"*
- Wrap the body at 72 characters

5 Rules of Commit Messages

- Separate subject from body with
- Limit the subject line to 50 chara
  - *Team-level convention on*
    - *style (markup syntax, wrap margin*
    - *metadata (how to reference issue t*
- Use the imperative mood in the
  - *"If applied, this commit will _____"*
- Wrap the body at 72 characters

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

    Simplify serialize.h's exception handling

    Remove the 'state' and 'exceptmask' from serialize.h's stream
    implementations, as well as related methods.

    As exceptmask always included 'failbit', and setstate was always
    called with bits = failbit, all it did was immediately raise an
    exception. Get rid of those variables, and replace the setstate
    with direct exception throwing (which also removes some dead
    code).

    As a result, good() is never reached after a failure (there are
    only 2 calls, one of which is in tests), and can just be replaced
    by !eof().

    fail(), clear(n) and exceptions() are just never called. Delete
    them.
```

Outline

- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

Coding Standards

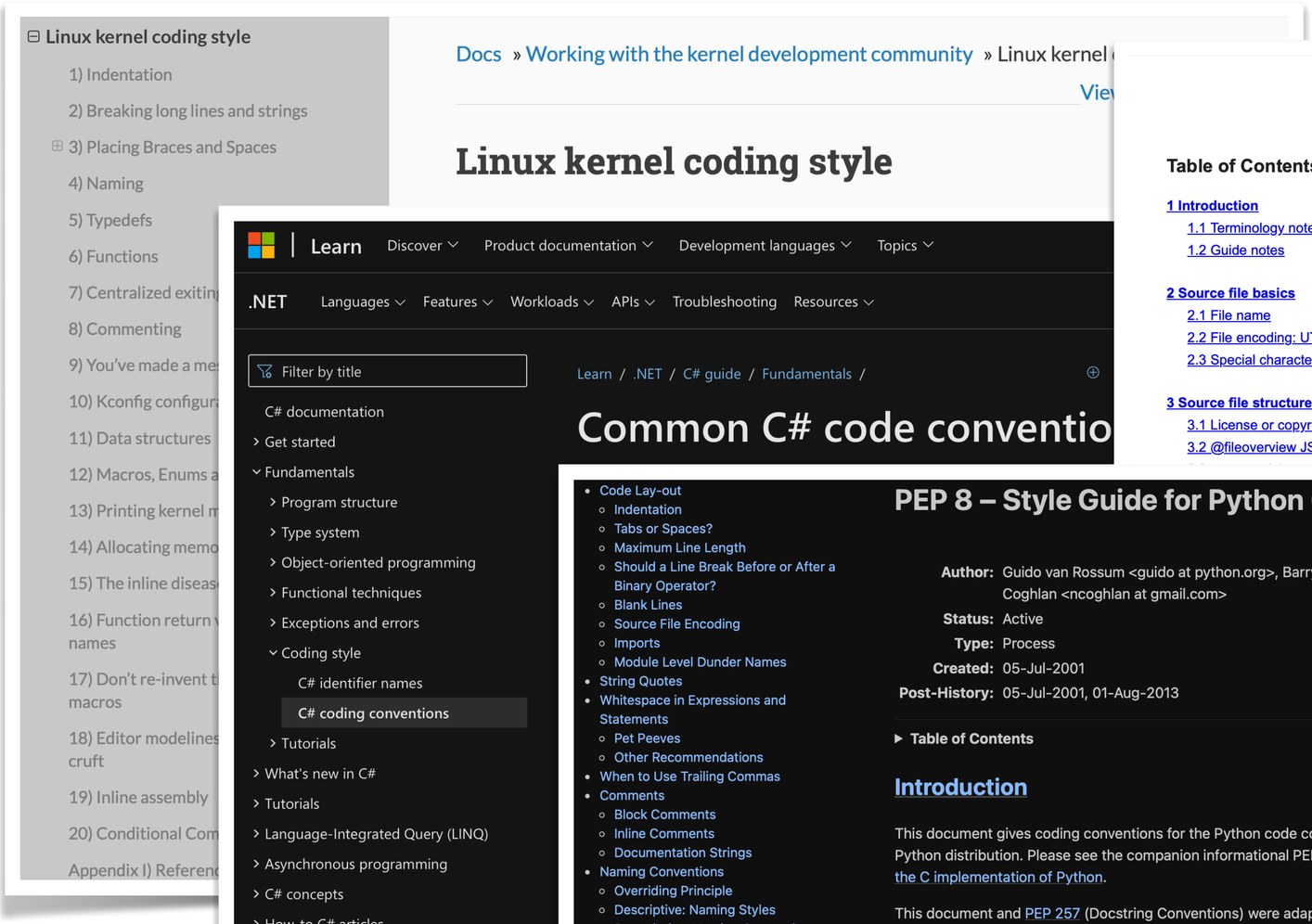
Coding Standards

- Naming conventions
  - *how to choose meaningful names for classes, methods, and variables*
  - *camelCase vs. snake_case vs. kebab-case vs. PascalCase for each of the above*
- Formatting
  - *spaces vs. tabs, max line length, ...*
  - *braces on same vs. next line, alignment of method parameters, etc.*
- Commenting and documentation
- Best practices

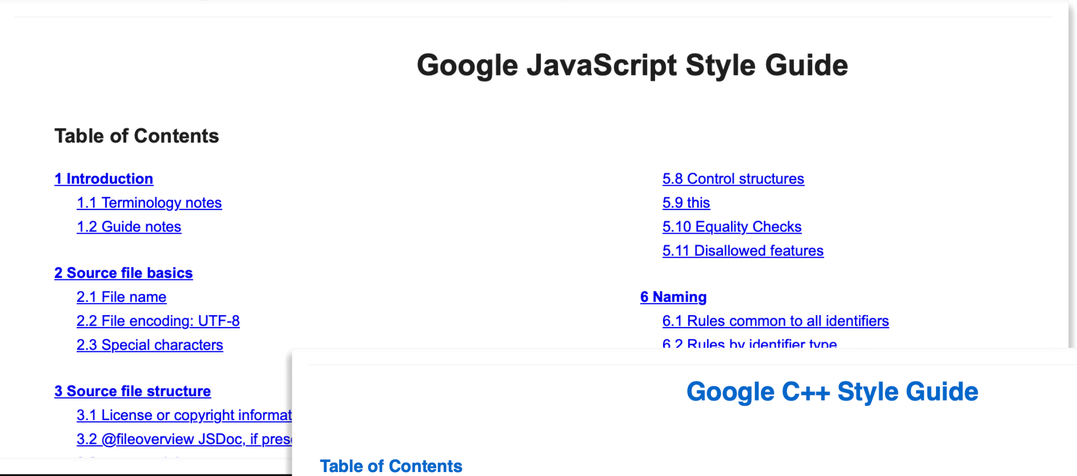
```
class OrderManager(private val server: Server) {

    // Place an order and return a corresponding OrderResult
    fun placeOrder(orderDetails: OrderDetails): OrderResult {
        val response = server.submitOrder(orderDetails)

        if (response.isSuccess) {
            return OrderResult.Success(response.orderId)
        }
        return OrderResult.Failure(response.errorMessage)
    }

    // Cancel an order by ID; returns true if successful, false otherwise
    fun cancelOrder(orderId: String): Boolean {
        val response = server.cancelOrder(orderId)

        return response.isSuccess
    }
}
```

```
class OrderManager(private val server: Server) {

    // Place an order and return a corresponding OrderResult
    fun placeOrder(details: OrderDetails) = server.submitOrder(details).let {
        if (it.isSuccess) OrderResult.Success(it.orderId) else OrderResult.Failure(it.errorMessage)
    }

    // Cancel an order by ID; returns true if successful, false otherwise
    fun cancelOrder(orderId: String): Boolean =
        server.cancelOrder(orderId).isSuccess
}
```

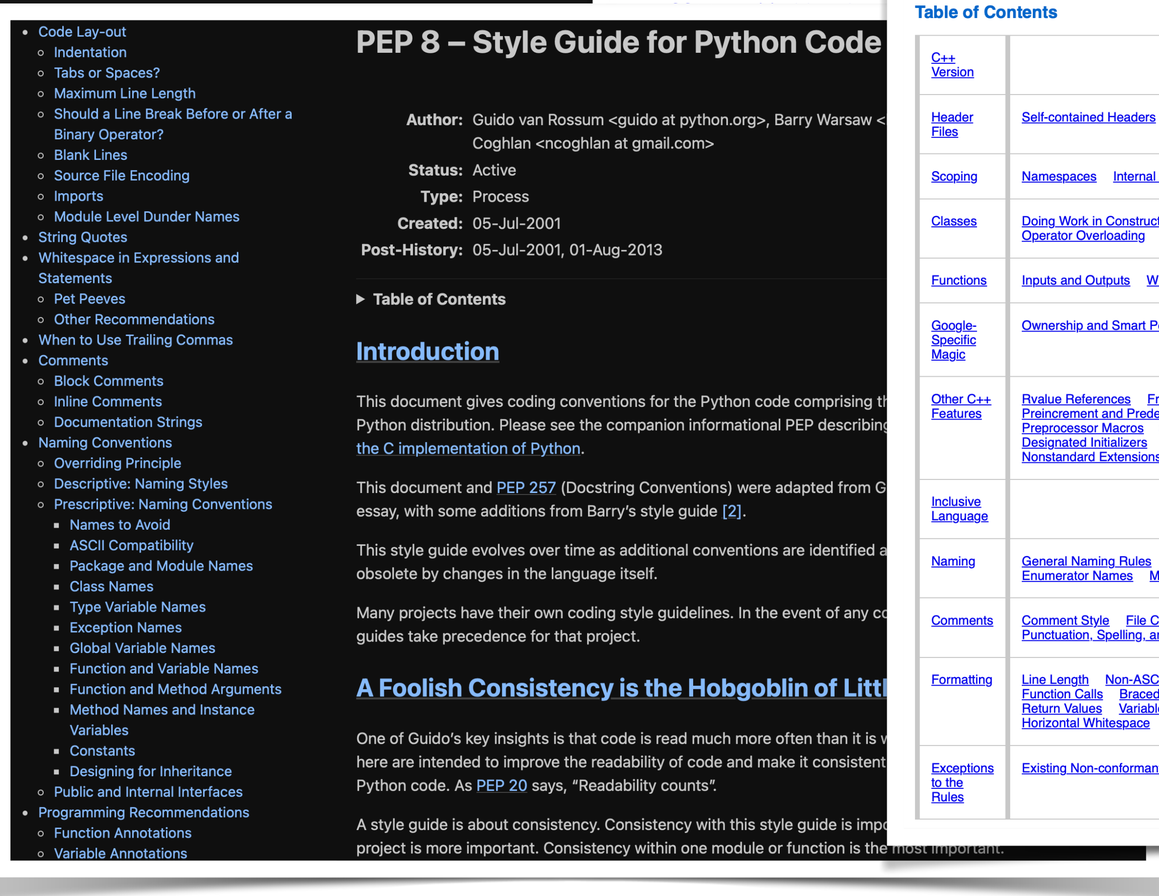
```
/**
 * Manages food orders by interfacing with a server for order operations.
 *
 * @property server The server used for order operations.
 */
class OrderManager(private val server: Server) {

    /**
     * Places a new order with specified details.
     *
     * Returns an [OrderResult] indicating the outcome.
     *
     * @param orderDetails Details of the order being placed.
     * @return [OrderResult] indicating success or failure.
     */
    fun placeOrder(orderDetails: OrderDetails): OrderResult =
        server.submitOrder(orderDetails).let { response ->
            if (response.isSuccess) OrderResult.Success(response.orderId)
            else OrderResult.Failure(response.errorMessage)
        }

    /**
     * Cancels an order by its ID.
     *
     * Returns a boolean indicating the operation's success.
     *
     * @param orderId The ID of the order to cancel.
     * @return True if cancellation was successful, false otherwise.
     */
    fun cancelOrder(orderId: String): Boolean =
        server.cancelOrder(orderId).isSuccess
}
```

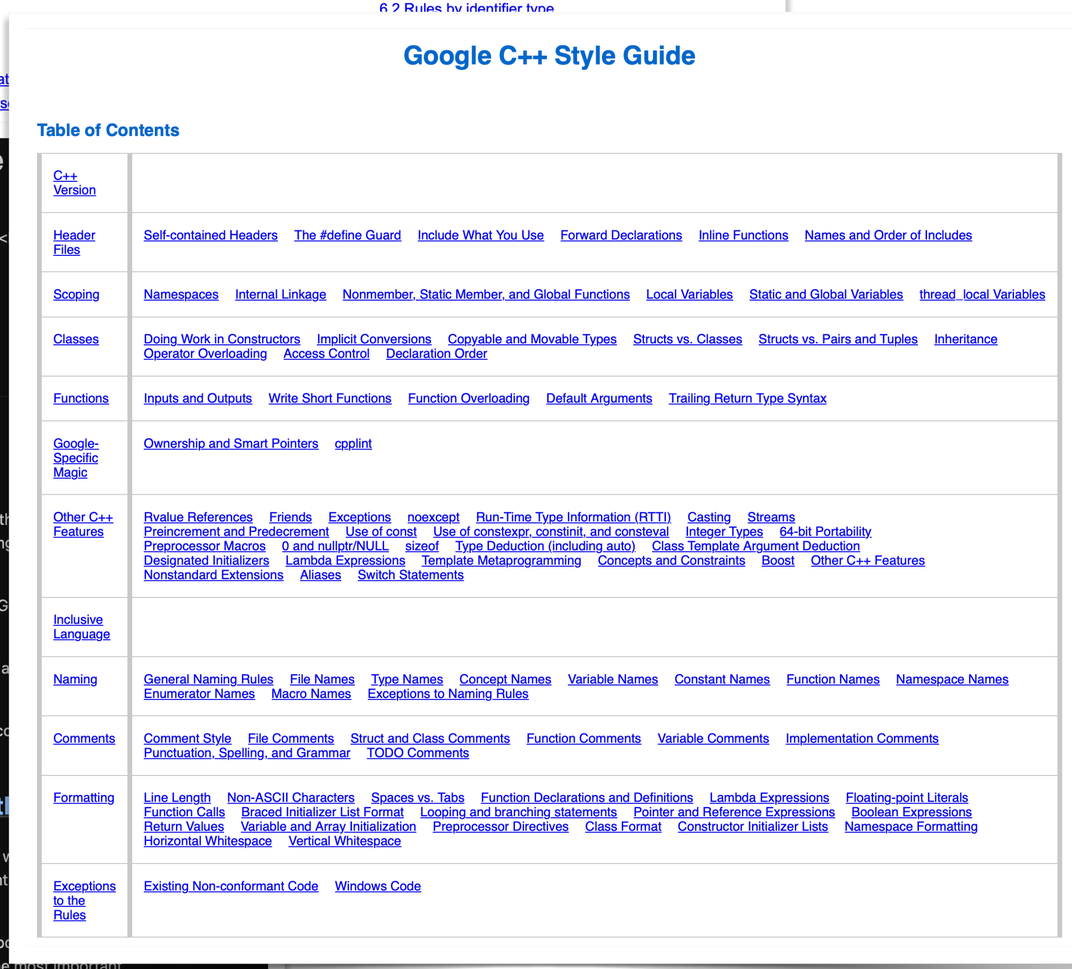

Linux kernel coding style
1) Indentation
2) Breaking long lines and strings
3) Placing Braces and Spaces
4) Naming
5) Typedefs
6) Functions
7) Centralized exiting
8) Commenting
9) You've made a me
10) Kconfig configura
11) Data structures
12) Macros, Enums a
13) Printing kernel m
14) Allocating memo
15) The inline disease
16) Function return v names
17) Don't re-invent t macros
18) Editor modelines cruft
19) Inline assembly
20) Conditional Com
Appendix I) Referenc

Docs » Working with the kernel development community » Linux kernel
Linux kernel coding style


Google JavaScript Style Guide
Table of Contents
1 Introduction
1.1 Terminology notes
1.2 Guide notes
2 Source file basics
2.1 File name
2.2 File encoding: UTF-8
2.3 Special characters
3 Source file structure
3.1 License or copyright informat
3.2 @fileoverview JSDoc, if pres
5.8 Control structures
5.9 this
5.10 Equality Checks
5.11 Disallowed features
6 Naming
6.1 Rules common to all identifiers
6.2 Rules by identifier type

Learn
Discover
Product documentation
Development languages
Topics
.NET
Languages
Features
Workloads
APIs
Troubleshooting
Resources
Filter by title
C# documentation
Get started
Fundamentals
Program structure
Type system
Object-oriented programming
Functional techniques
Exceptions and errors
Coding style
C# identifier names
C# coding conventions
Tutorials
What's new in C#
Tutorials
Language-Integrated Query (LINQ)
Asynchronous programming
C# concepts
How-to C# articles
Advanced topics
The .NET Compiler Platform SDK (Roslyn APIs)
C# programming guide
Language reference
Specifications
Learn / .NET / C# guide / Fundamentals /
Common C# code conventio


Code Lay-out
Indentation
Tabs or Spaces?
Maximum Line Length
Should a Line Break Before or After a Binary Operator?
Blank Lines
Source File Encoding
Imports
Module Level Dunder Names
String Quotes
Whitespace in Expressions and Statements
Pet Peeves
Other Recommendations
When to Use Trailing Commas
Comments
Block Comments
Inline Comments
Documentation Strings
Naming Conventions
Overriding Principle
Descriptive: Naming Styles
Prescriptive: Naming Conventions
Names to Avoid
ASCII Compatibility
Package and Module Names
Class Names
Type Variable Names
Exception Names
Global Variable Names
Function and Variable Names
Function and Method Arguments
Method Names and Instance Variables
Constants
Designing for Inheritance
Public and Internal Interfaces
Programming Recommendations
Function Annotations
Variable Annotations
PEP 8 – Style Guide for Python Code
Author: Guido van Rossum <guido at python.org>, Barry Warsaw <
Coghlan <ncoghlan at gmail.com>
Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013
Table of Contents
Introduction
This document gives coding conventions for the Python code comprising th
Python distribution. Please see the companion informational PEP describing
the C implementation of Python.
This document and PEP 257 (Docstring Conventions) were adapted from G
essay, with some additions from Barry's style guide [2].
This style guide evolves over time as additional conventions are identified a
obsolete by changes in the language itself.
Many projects have their own coding style guidelines. In the event of any c
guides take precedence for that project.
A Foolish Consistency is the Hobgoblin of Litt
One of Guido's key insights is that code is read much more often than it is v
here are intended to improve the readability of code and make it consistent
Python code. As PEP 20 says, "Readability counts".
A style guide is about consistency. Consistency with this style guide is impo
project is more important. Consistency within one module or function is the


Google C++ Style Guide
Table of Contents
C++ Version
Header Files
Self-contained Headers
The #define Guard
Include What You Use
Forward Declarations
Inline Functions
Names and Order of Includes
Scoping
Namespaces
Internal Linkage
Nonmember, Static Member, and Global Functions
Local Variables
Static and Global Variables
thread_local Variables
Classes
Doing Work in Constructors
Implicit Conversions
Copyable and Movable Types
Structs vs. Classes
Structs vs. Pairs and Tuples
Inheritance
Operator Overloading
Access Control
Declaration Order
Functions
Inputs and Outputs
Write Short Functions
Function Overloading
Default Arguments
Trailing Return Type Syntax
Google-Specific Magic
Ownership and Smart Pointers
cpplint
Other C++ Features
Rvalue References
Friends
Exceptions
noexcept
Run-Time Type Information (RTTI)
Casting
Streams
Preincrement and Predecrement
Use of const
Use of constexpr, constinit, and consteval
Integer Types
64-bit Portability
Preprocessor Macros
0 and nullptr/NULL
sizeof
Type Deduction (including auto)
Class Template Argument Deduction
Designated Initializers
Lambda Expressions
Template Metaprogramming
Concepts and Constraints
Boost
Other C++ Features
Nonstandard Extensions
Aliases
Switch Statements
Inclusive Language
Naming
General Naming Rules
File Names
Type Names
Concept Names
Variable Names
Constant Names
Function Names
Namespace Names
Enumerator Names
Macro Names
Exceptions to Naming Rules
Comments
Comment Style
File Comments
Struct and Class Comments
Function Comments
Variable Comments
Implementation Comments
Punctuation, Spelling, and Grammar
TODO Comments
Formatting
Line Length
Non-ASCII Characters
Spaces vs. Tabs
Function Declarations and Definitions
Lambda Expressions
Floating-point Literals
Function Calls
Braced Initializer List Format
Looping and branching statements
Pointer and Reference Expressions
Boolean Expressions
Return Values
Variable and Array Initialization
Preprocessor Directives
Class Format
Constructor Initializer Lists
Namespace Formatting
Horizontal Whitespace
Vertical Whitespace
Exceptions to the Rules
Existing Non-conformant Code
Windows Code

Outline

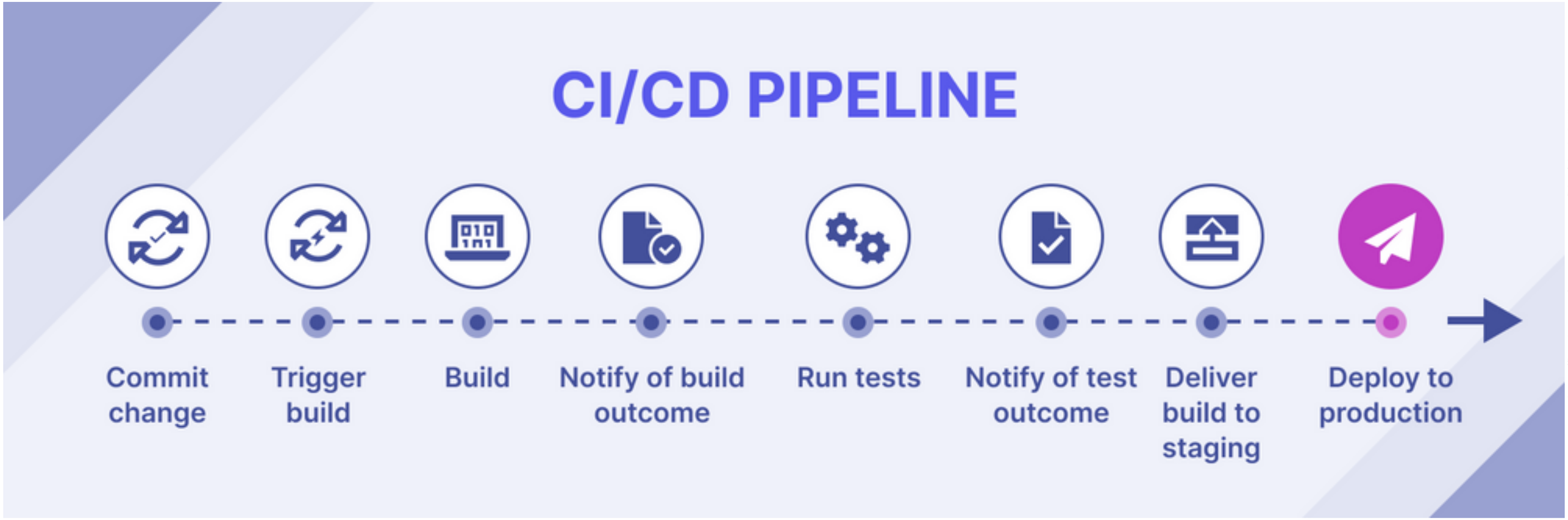
- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

CI / CD

Continuous Integration / Continuous Delivery & Deployment

- = automated assembly line for your software product
- goals of Continuous Integration
  - *merge code changes from multiple devs and ensure that they work together*
  - *catch bugs and integration problems as early as possible*
- goals of Continuous Delivery
  - *constantly have the software in "deployable state"*
  - *(automatically package software ready for deployment)*
- goals of Continuous Deployment
  - *automatically deploy to testing / staging / production*

Typical CI/CD Workflow

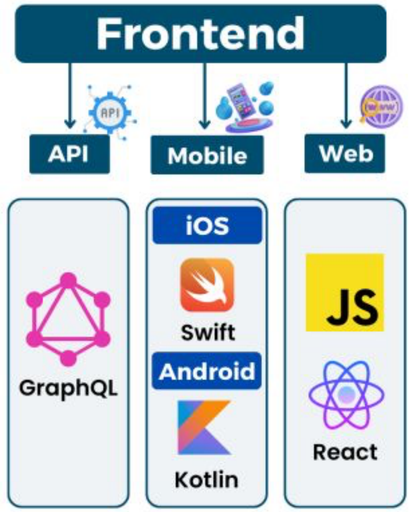


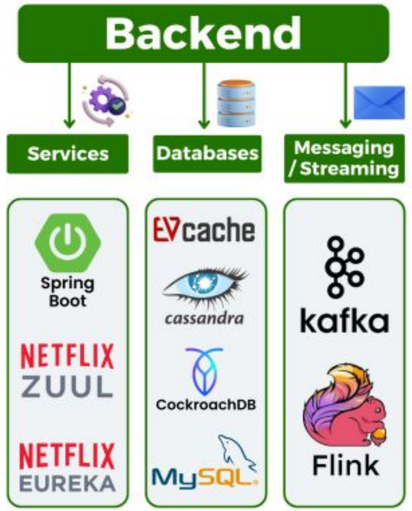
https://katalon.com/resources-center/blog/ci-cd-pipeline

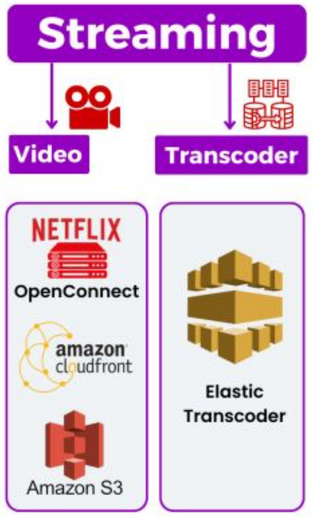
CI/CD Tools

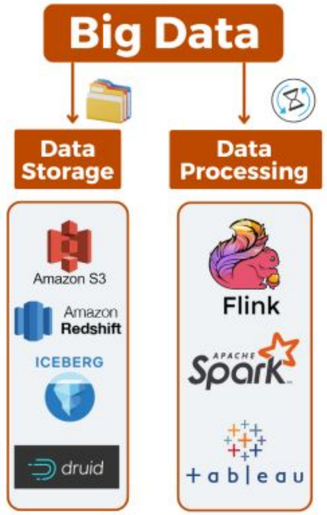
- Version control
- CI/CD server to orchestrate the build—test—deploy processes
- Builder and dependency handler
- Testing frameworks
- Code review tools
- Static code analysis
- VMs and/or containers
- Infrastructure as Code (IaC)
- Monitoring and alerting

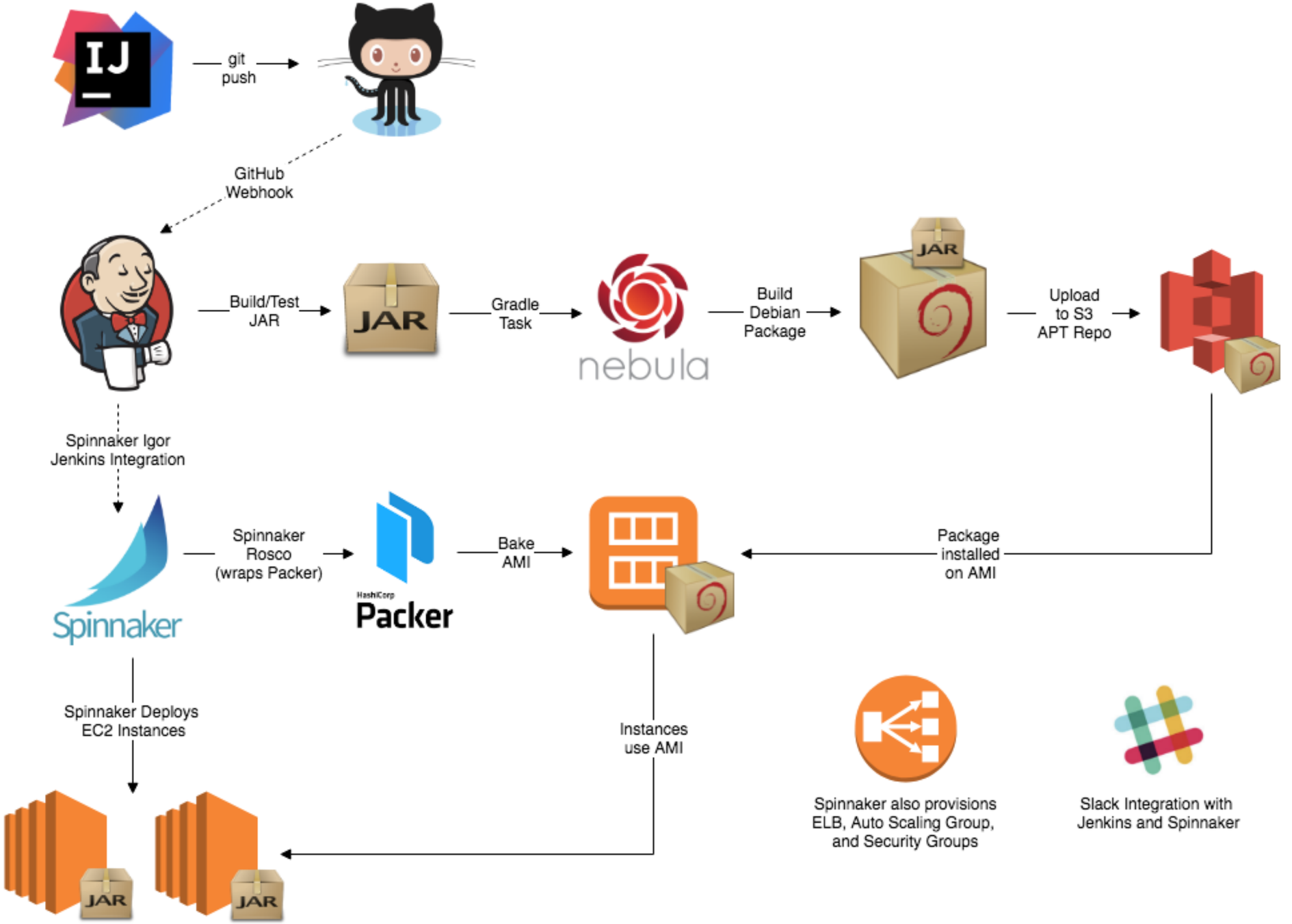
NETFLIX











https://programmaticponderings.com/2018/05/12/deploying-spring-boot-apps-to-aws-with-netflix-nebula-and-spinnaker-part-2-of-2/

Chaos Monkey

Randomly disables production instances

Janitor Monkey

Identifies and disposes unused resources

Chaos Kong

Drops a full AWS Region

Conformity Monkey

Shuts down instances not adhering to best-practices

Chaos Gorilla

Outage of entire Amazon Availability Zone

Security Monkey

Finds security violations and vulnerability

NETFLIX SIMIAN ARMY

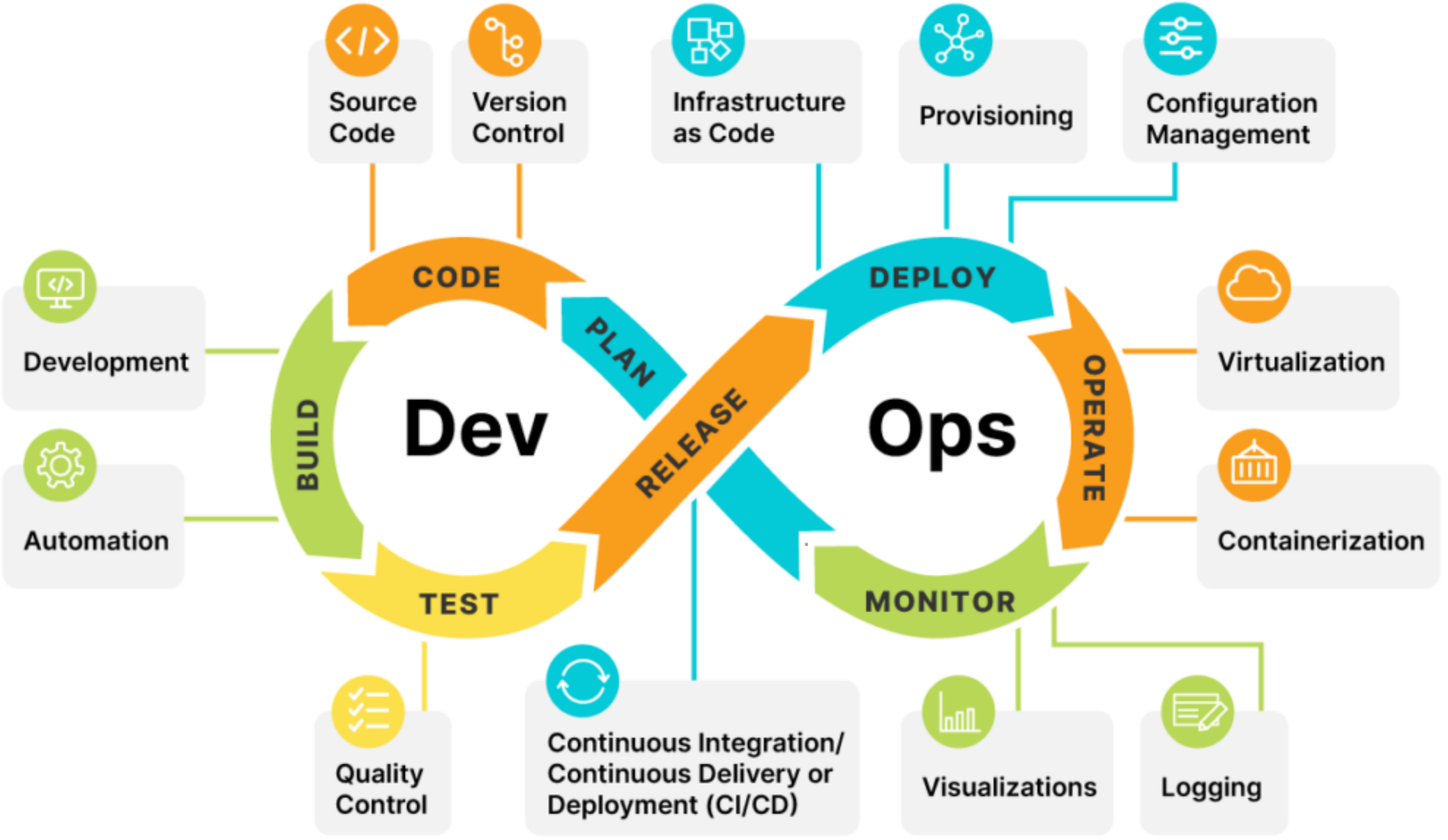
@geosley

Doctor Monkey

Taps into health checks and fixes unhealthy resources

Latency Monkey

Simulate degradation or outages in a network


Source Code
Version Control
Infrastructure as Code
Provisioning
Configuration Management
Development
Automation
CODE
PLAN
BUILD
TEST
RELEASE
DEPLOY
OPERATE
MONITOR
Dev
Ops
Virtualization
Containerization
Quality Control
Continuous Integration/ Continuous Delivery or Deployment (CI/CD)
Visualizations
Logging

Outline

- Development process: Scrum
- Collaboration workflows
- Writing good commit messages
- Coding standards
- CI / CD

Next week: The Real Project ...

https://www.flickr.com/photos/jinka_matousek/8437433279