

Functional Requirements
vs.
Non-Functional Requirements

Functional Requirements

- We've mostly discussed **functional requirements** so far
 - Describes a function of the system
 - Potentially includes its **inputs**, **behaviors** and **output**
 - Feature
- Familiar Examples
 - As a potential user, I need to login in order to access my account data
 - As a new user, I need to create a new account that will be associated with my credentials

Non-Functional Requirements

- Constraints on the product's services or functions
- Often apply to the system as a whole, not to a specific feature
 - Reliability
 - Recovery (from failure)
 - Availability
 - Usability
 - Performance
 - Regulatory
 - Supportability (bug fixing)

Non-Functional Requirements Not Important?!

- How much \$\$\$ does Amazon lose for every second their services are not running?

Non-Functional Requirements Not Important?!

- Amazon's revenue is $\approx \$177,860,000,000/\text{year}$
- That's $> \$5,600/\text{second}$
- While Amazon is an extreme example, reliability, supportability and availability are important to all eBusiness

February 28, 2017



Amazon AWS support team offices right now. [#awsoutage](#)



I Survived the Great ASW S3 in US-EAST-1 Outage of February 28, 2017 -

#awsoutage #awscloud #awsdown



When **Amazon S3** is down. [#awscloud](#) [#awss3](#)

whats S3

The internet's hard drive

S3 size

- “The S3 system is used by 148,213 sites according to market research firm SimilarTech”
 - <https://www.similartech.com/technologies/amazon-s3>
- “It has “north of three to four trillion pieces of data stored in it,” Bartoletti said.”

Amazon S3 according to the [#AWS](#) status page.



Irony, defined. [#awsoutage](#)



The www.isitdownrightnow.com page isn't working

www.isitdownrightnow.com is currently unable to handle this request.

HTTP ERROR 503

Reload

AWS Outage – What caused it?

"[...] The Amazon Simple Storage Service (S3) team was debugging an issue causing the S3 billing system to progress more slowly than expected. At 9:37AM PST, an authorized S3 team member using an established playbook executed a command which was intended to remove a small number of servers for one of the S3 subsystems that is used by the S3 billing process. Unfortunately, one of the inputs to the command was entered incorrectly and a larger set of servers was removed than intended. The servers that were inadvertently removed supported two other S3 subsystems. One of these subsystems, the index subsystem, manages the metadata and location information of all S3 objects in the region. This subsystem is necessary to serve all GET, LIST, PUT, and DELETE requests. The second subsystem, the placement subsystem, manages allocation of new storage and requires the index subsystem to be functioning properly to correctly operate. The placement subsystem is used during PUT requests to allocate storage for new objects. Removing a significant portion of the capacity caused each of these systems to require a full restart. While these subsystems were being restarted, S3 was unable to service requests. Other AWS services in the US-EAST-1 Region that rely on S3 for storage, including the S3 console, Amazon Elastic Compute Cloud (EC2) new instance launches, Amazon Elastic Block Store (EBS) volumes (when data was needed from a S3 snapshot), and AWS Lambda were also impacted while the S3 APIs were unavailable. " - <https://aws.amazon.com/message/41926/>

Non-Functional Requirements

- Constraints on the product's services or functions
- Often apply to the system as a whole, not to a specific feature
 - Reliability
 - Recovery (from failure)
 - Availability
 - Usability
 - Performance
 - Regulatory
 - Supportability (bug fixing)

Usability Example

- The average first-time user will successfully create a new account in less than 60 seconds
- The user should be able to go from X to Y in less than 3 clicks

Reliability Example

- The **Mean Time To Failure (MTTF)** for the server will exceed 720 hours (~1 month)

Recovery Example

- The Mean Time To Recover (MTTR) the server following a failure will be less than 1 hour

Availability Example

- The server will be available more than 99.86% of the time
 - Note: Is preventive maintenance included?
- How to compute the 99.86%?

Availability Example

- The server will be available more than 99.86% of the time
 - Note: Is preventive maintenance included?
- A_I = Inherent Availability (excludes preventive maintenance)

$$A_I =$$

Availability Example

- The server will be available more than 99.86% of the time
 - Note: Is preventive maintenance included?
- A_I = Inherent Availability (excludes preventive maintenance)
- **MTTF** = Mean Time To Failure
- **MTTR** = Mean Time To Recover
- MTBF = Mean Time Between Failures
(**MTTF**+**MTTR**)

$$A_I = \frac{MTTF}{MTTF + MTTR}$$

Availability Example

- The server will be available more than 99.86% of the time
 - Note: Is preventive maintenance included?
- A_I = Inherent Availability (excludes preventive maintenance)
- **MTTF** = Mean Time To Failure
- **MTTR** = Mean Time To Recover
- MTBF = Mean Time Between Failures
(**MTTF**+**MTTR**)

$$A_I = \frac{MTTF}{MTTF + MTTR} = \frac{720}{720 + 1} = 99.86\%$$

Amazon AWS Outage – February 28, 2017

- On the day of the outage, it was reported that AWS up-time was ~99.59% (~178 minutes/month downtime)
- which is under their 99.95% (~21 minutes/month downtime) target

Amazon AWS Service Commitment

“AWS will use **commercially reasonable efforts** to make Amazon EC2 and Amazon EBS each available with a Monthly Uptime Percentage (defined below) **of at least 99.95%**, in each case during any monthly billing cycle (the “Service Commitment”). In the event Amazon EC2 or Amazon EBS does not meet the Service Commitment, **you will be eligible to receive a Service Credit as described below.**”

Performance Examples

- We are considering both **timing** and **load** here (some authors separate them)
- The server will provide an **average response in 1 second**
- The server will **support 100 users**
 - Requirements may have interdependencies
 - And still provide a 1 second average response?
 - And still provide 99.86% availability? :)

Regulatory Examples

- The server will comply with all **H**ealth **I**nsurance **P**ortability and **A**ccountability **A**ct (**HIPAA**) requirements
- **F**amily **E**ducational **R**ights and **P**rivacy **A**ct (**FERPA**)
- The server will be **ISO 9001:2015** certified
- The system will comply with **DO-178B**, Software Considerations in Airborne Systems and Equipment Certification
- **MIL-STD**
- **IEEE 754** Standard for Floating-Point Arithmetic
- European Union regulations

Supportability Examples

- Uncommon but important to certain IT circles (e.g., Facebook)
- The average time required to repair a defect in the server will be less than 1 day
 - How can it be “enforced”?

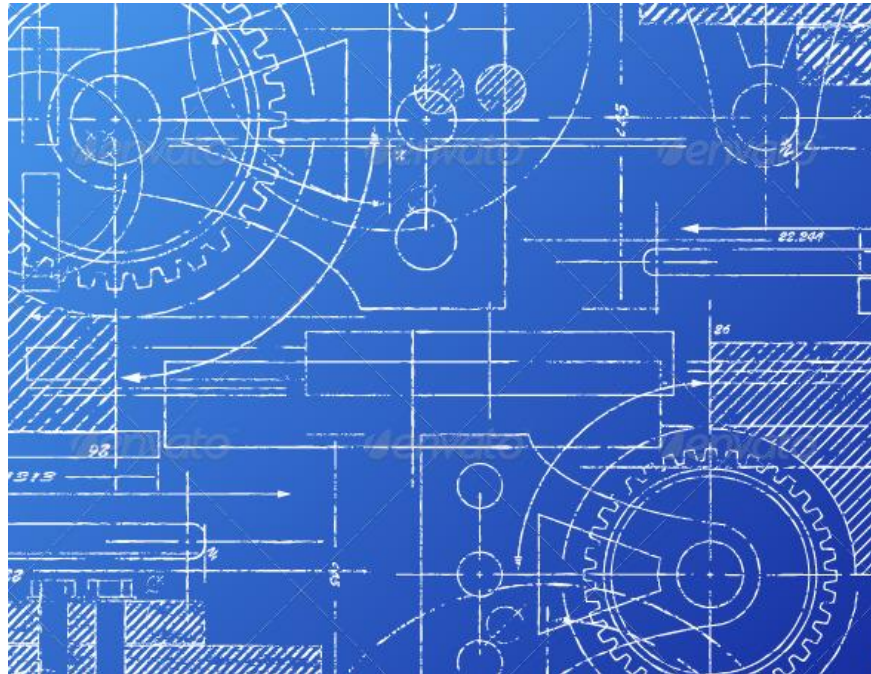
Supportability Examples

- Uncommon but important to certain IT circles (e.g., Facebook)
- The average time required to repair a defect in the server will be less than 1 day
 - How will your project demonstrate compliance?
 - Does the time to repair include the time to run all the tests?
 - Does rolling back to a previous (working) version “repair” the defect?
 - If so, can we keep a previous version of the server in hot-standby ready to go online if the new version fails?

Principles of Software Analysis and Design

What is *Software Design*?

- A **plan** or **pattern** (e.g., blueprint) for constructing software
- A model, a simplified illustration of software



Communicating Software Design

- **UML** is widely used to communicate design
 - Class Diagrams
 - Sequence Diagrams
 - State Charts
- *Design Patterns* have emerged from a search for higher-level abstractions, best practices in design

When to Communicate Design?

- **Prior to implementation** during the formative design process
 - To communicate and test design proposals (“We could do this...”)
 - To critique their limitations and/or failures (“THAT won’t work!”)
- **Following the design’s implementation**
 - To teach future developers how the design works
 - To teach future developers why some choices were avoided

What type of properties do we
want from design?

What We Want from Design:

Sufficiency

- The design needs to satisfy the product requirements
- But sufficiency is inadequate by itself

What We Want from Design: Understandability

- You must be able to **communicate** your design
- Your development team **will not adopt your design if they cannot understand it...**
- ...even if you have the best design!

What We Want from Design:

Encapsulation

- We seek to **conceal the internal operation** of a module:
 - ...so that we can someday change its implementation without that change rippling through other modules that use it
 - ...so that its user need not understand its internal operation
- Example: If I build a List data structure, I wish to protect my implementation from unwanted coupling

What We Want from Design:

Flexibility

- Want to be able to add, modify or remove features...
- ...as business conditions change

What We Want from Design: Reusability

- Want to leverage reusable components from other designs
- Want to create new components to leverage in future designs

Virtues of Programmers

- Laziness?



Virtues of Programmers

Larry Wall, Inventor of Perl

■ Laziness?



“It makes you write labor-saving programs that other people will find useful, and document what you wrote so you don't have to answer so many questions about it.”

What We Want from Design: Modularity

What We Want from Design:

Modularity

- Need to divide a large system into **smaller modules**, each to be **implemented independently** of the others
- This enables us to apply dozens, hundreds, even thousands of developers to large problems
- ...an approach to achieving understandability

What We Want from Design:

Cohesion and Coupling

- Modularity must be achieved with attention to *cohesion* and *coupling*

The Object-Oriented Paradigm: Cohesion

- *Cohesion* is good!
- Cohesion refers to the degree of **interaction within a module**
- OOP can achieve **cohesion** by encapsulating
 - related methods and
 - state variables
 - NB: But Java doesn't force you to do that — it's up to you to ensure the members are related!

The Object-Oriented Paradigm: Cohesion

- Each object should have a **single responsibility**
- **Instance variables** model the state of an object.
Instance variables should be:
 - **orthogonal**
 - captures “independence” between different dimensions
 - do not store redundant data, or data that can be derived from existing data (e.g., DOB vs. age)
 - **related** and
 - **necessary** to model the object’s responsibility

The Object-Oriented Paradigm:

Cohesion

- **Methods** model the behaviors of that object. Each method should reference:
 - an instance variable, or
 - at least another method in the same object

The Object-Oriented Paradigm: Coupling

The Object-Oriented Paradigm: Coupling

- *Coupling* refers to the degree of interaction between modules
- Coupling often arises in
 - a dependency,
 - a local variable or
 - parameter referencing an object of a different data type

The Object-Oriented Paradigm: Coupling

- *Coupling* is undesirable

The Object-Oriented Paradigm: Coupling

- *Coupling* is undesirable but always necessary
- Approaches for minimizing coupling:
 - Use built-in data types for parameters
 - Use interfaces
 - Use getters/setters

What We Want from Design:

Cohesion and Coupling

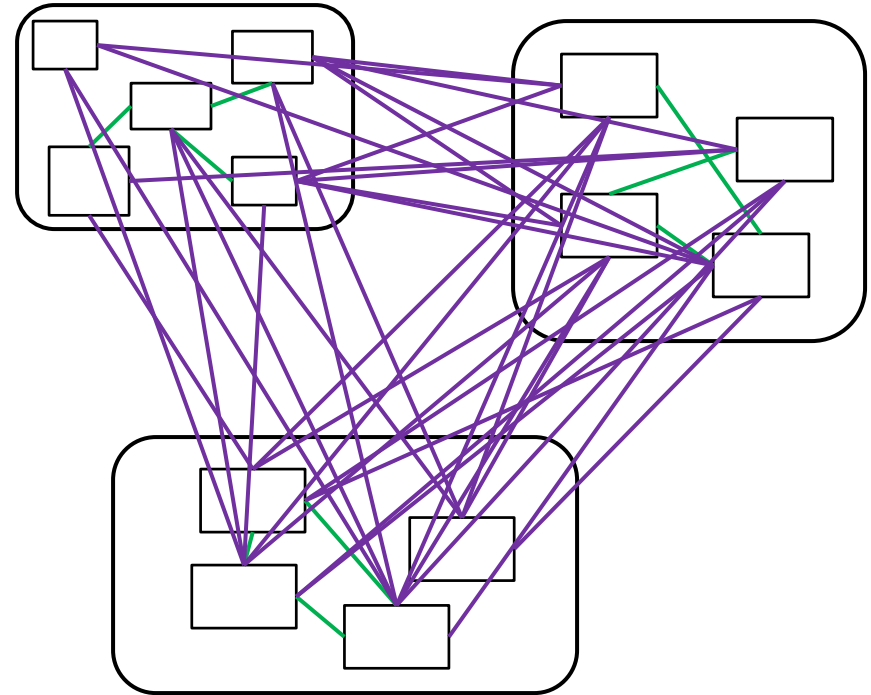
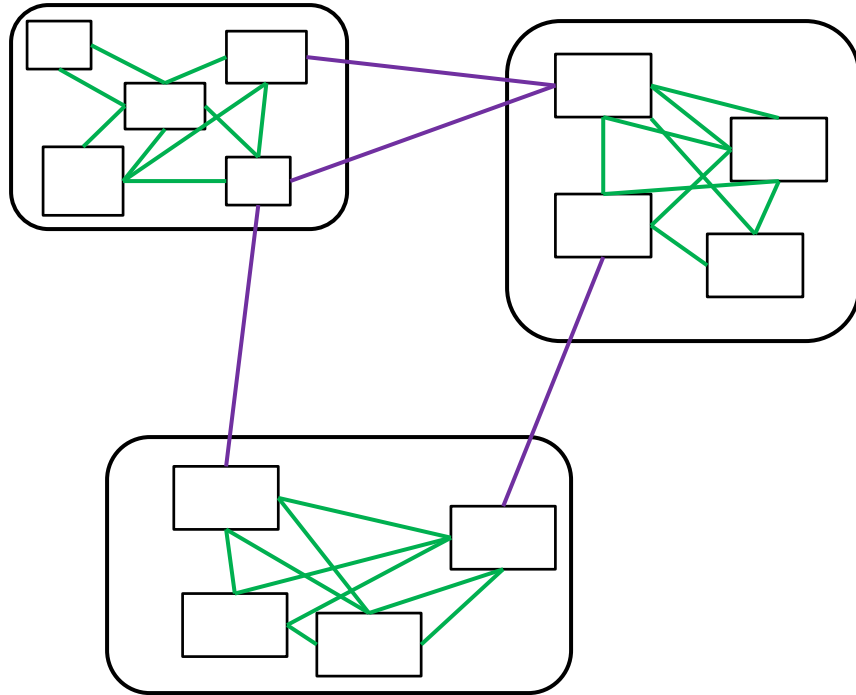
- Modularity must be achieved with attention to *cohesion* and *coupling*
 - *Cohesion* refers to interactions within a module
 - *Coupling* refers to interactions between modules

What We Want from Design:

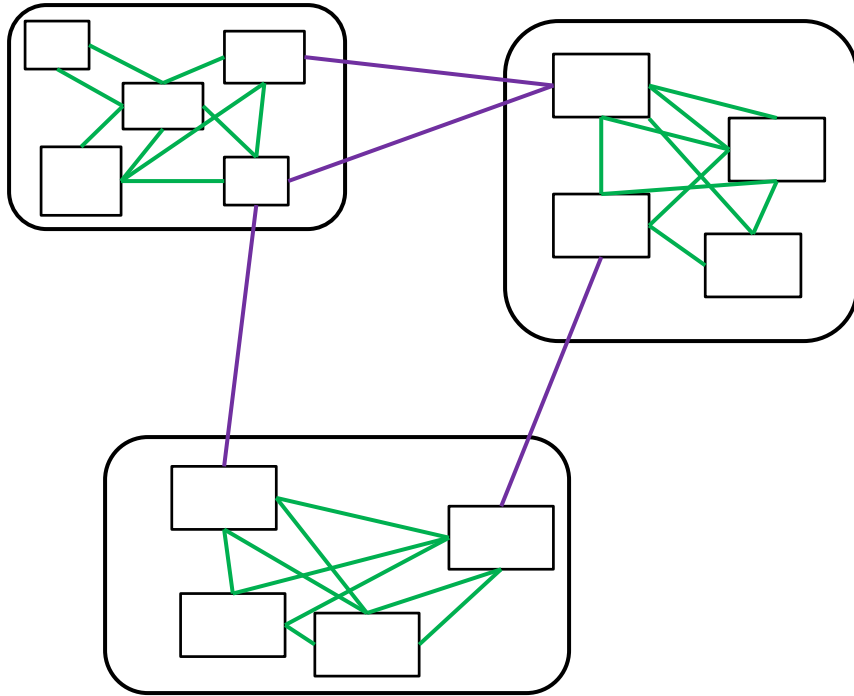
Cohesion and Coupling

- Modularity must be achieved with attention to *cohesion* and *coupling*
 - *Cohesion* refers to interactions *within a module*
 - *Coupling* refers to interactions *between modules*
- Maximize *cohesion* by grouping like-minded parts in a module
- Minimize *coupling* (dependencies) between modules

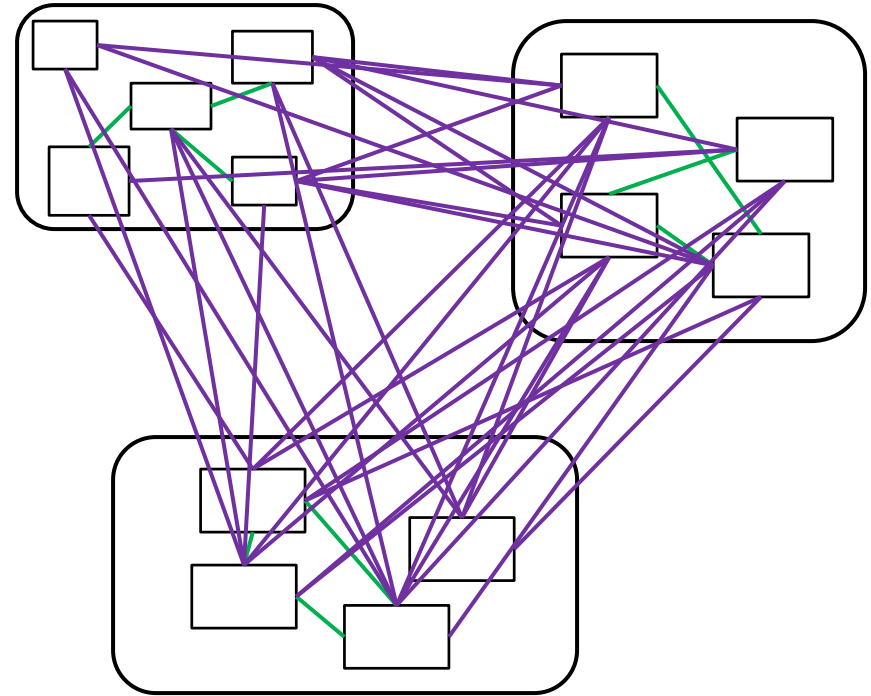
Which Design is "Better"?



Which Design is "Better"?

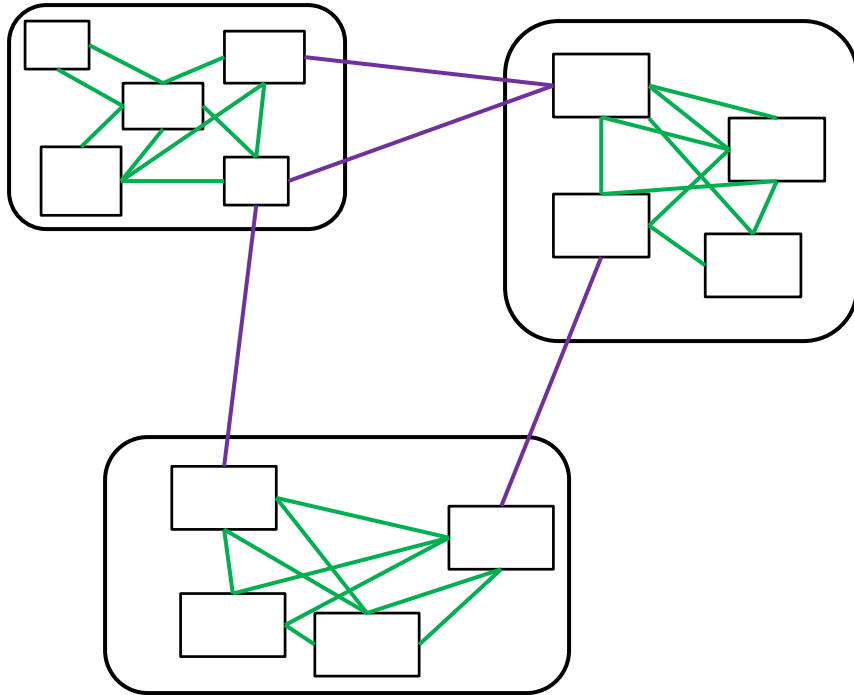


High Cohesion
Low Coupling

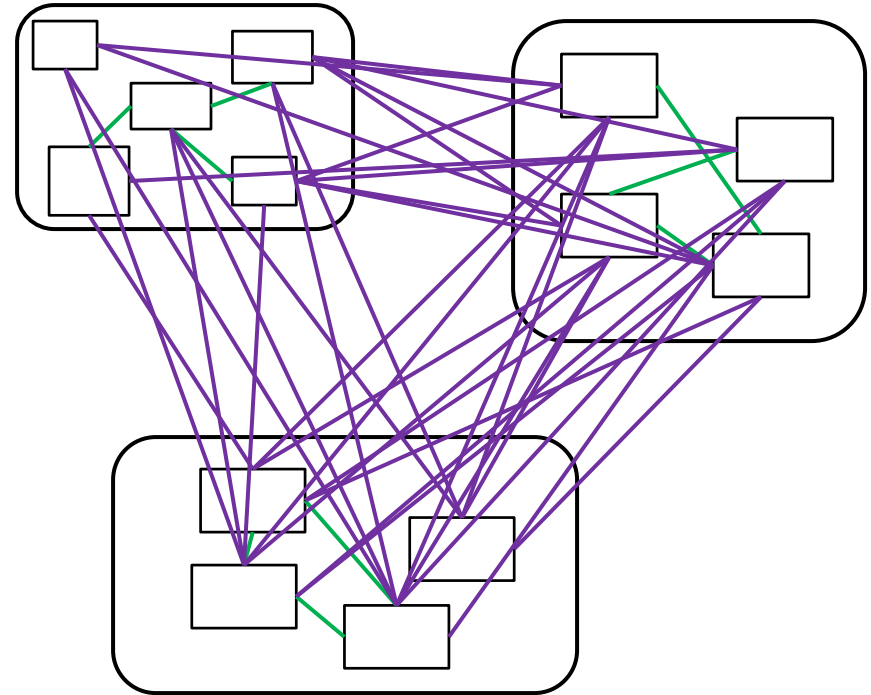


Low Cohesion
High Coupling

Which Design is "Better"?



High Cohesion
Low Coupling



Low Cohesion
High Coupling



Software Metrics

Software Metrics

- A *metric* is a system of measurement
- A *software metric* is a measurement of a software project
 - Allows the software to be *quantified*

Really Basic Scrum Metrics

- **Velocity** (e.g., Story Points per Sprint)
 - Useful for ...
- **Work Remaining in Sprint** (e.g., Story Points)
 - Applicability ...

Really Basic Scrum Metrics

- **Velocity** (e.g., Story Points per Sprint)
 - Useful for planning
- **Work Remaining in Sprint** (e.g., Story Points)
 - Applicability: burndown chart



Software Metrics: Examples

■ Code complexity:

- Lines-of-Product-Code
- Fan-in/fan-out
 - Number of “incoming” / “outgoing” dependencies
- Cyclomatic Complexity
 - “counts” the number of conditions in a program

■ Productivity:

- Lines-of-Product-Code per engineer-day

Example metrics:

<http://metrics.sourceforge.net/>

Metrics - /net.sourceforge.metrics						
Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
[-] Number of Packages	16					
[-] Number of Methods (avg/max per type)	1310	6.65	8.553	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
[-] tgsrc	489	7.191	11.544	76	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
[-] src	761	6.238	6.553	45	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.core.sources	108	15.429	12.129	45	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.ui	77	9.625	10.111	33	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.core	198	6.6	7.093	27	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.ui.preferences	52	6.5	7.467	26	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.ui.dependencies	95	5.588	3.727	15	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.internal.persistence	18	4.5	4.33	12	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.internal.prevayler.implementa...	54	5.4	2.871	10	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.internal.xml	41	4.1	2.022	9	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.calculators	79	4.158	2.254	8	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.propagators	31	5.167	1.067	7	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.internal.tests	8	2.667	1.886	4	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] net.sourceforge.metrics.internal.prevayler	0	0	0			
[-] classycle	60	8.571	2.556	13	/net.sourceforge.metrics/classycle/classycle/g...	
[-] Lines of Code (avg/max per type)	6593	33.467	49.02	339	/net.sourceforge.metrics/tgsrc/com/touchgrap...	
[-] Number of Interfaces (avg/max per packageFragment)	16	1	1.414	4	/net.sourceforge.metrics/src/net/sourceforge/...	
[-] Lines of Code (avg/max per method)	6593	4.812	7.355	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
[-] classycle	324	5.4	9.94	69	/net.sourceforge.metrics/classycle/classycle/g...	calculateAttributes
[-] tgsrc	2321	4.661	8.278	59	/net.sourceforge.metrics/tgsrc/com/touchgrap...	scrollSelectPanel
[-] src	3948	4.862	6.473	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
[-] net.sourceforge.metrics.ui	544	6.8	8.707	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
[-] MetricsTable.java	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
[-] MetricsTable	194	10.778	13.831	52	/net.sourceforge.metrics/src/net/sourceforge/...	setMetrics
[-] setMetrics	52					

Software Metrics: Examples

■ Object-Oriented Metrics:

- Depth of Inherence
- Method fan-in/fan-out
- Number of overriding operations
- **Coupling** between Object Classes (COB)
- Lack of **Cohesion** in Methods (LCOM)

Software Metrics: Examples

■ Quality:

- Defect Density
- Number of errors found per person hour
- Availability

■ Test Suite Effectiveness:

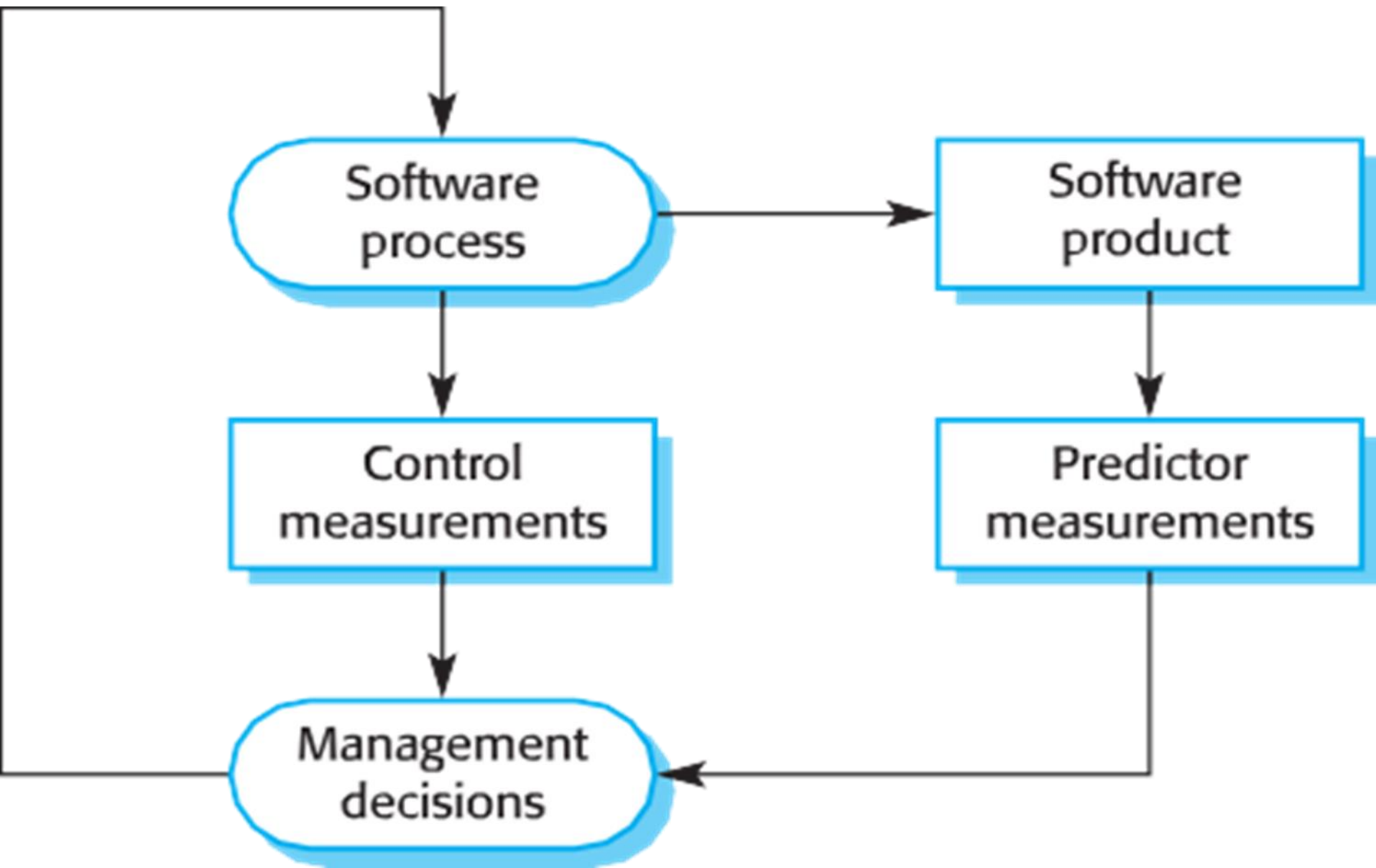
- Lines-of-Test-Code / Lines-of-Product-Code
- Defect Removal Efficiency (Effectiveness)
- Test (AKA Code) Coverage
- Killed-Mutant Fraction

Example Industry Data

- Product Size: 207 KLOC
- Productivity: 24 LinesOfProductCode/EngineerDay
- Delivered Quality: 5 Defects/KLOC
- Test Coverage: 55%
- System-Level Defects Removed/EngineerWeek: 4.3

Software Metrics Applicability

Software Metrics Applicability



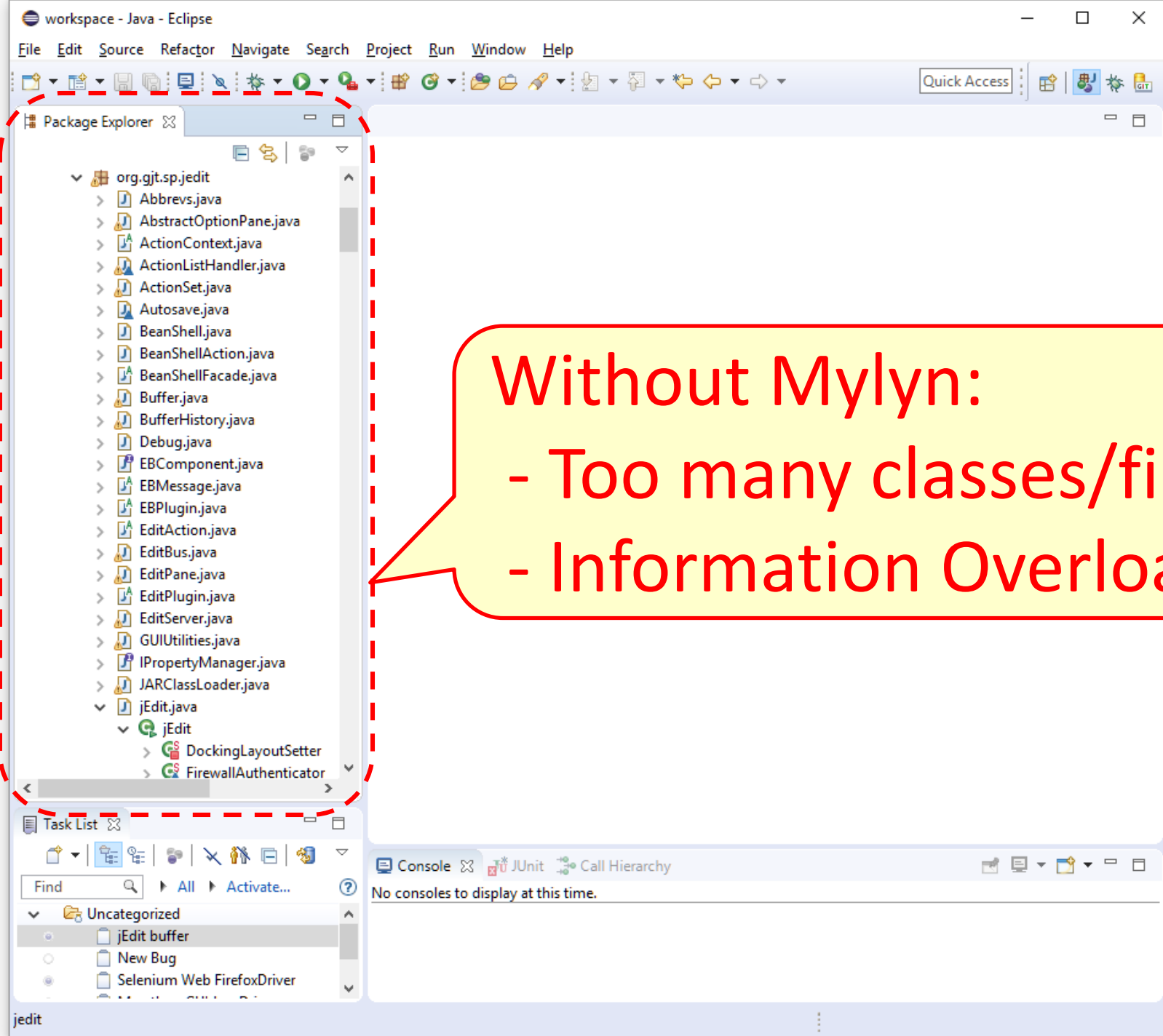
Software Metrics Applicability

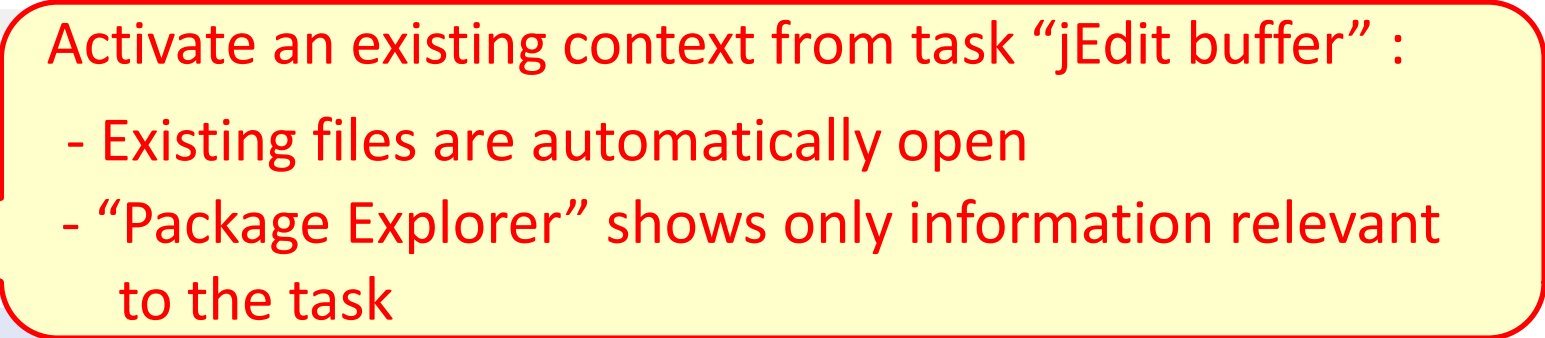
- **predict** product attributes/artifacts
 - e.g., we will be able to release by ...
- **control** the software process
 - Improve the code quality
- Note: There are **no standardized** and **universally applicable** software metrics

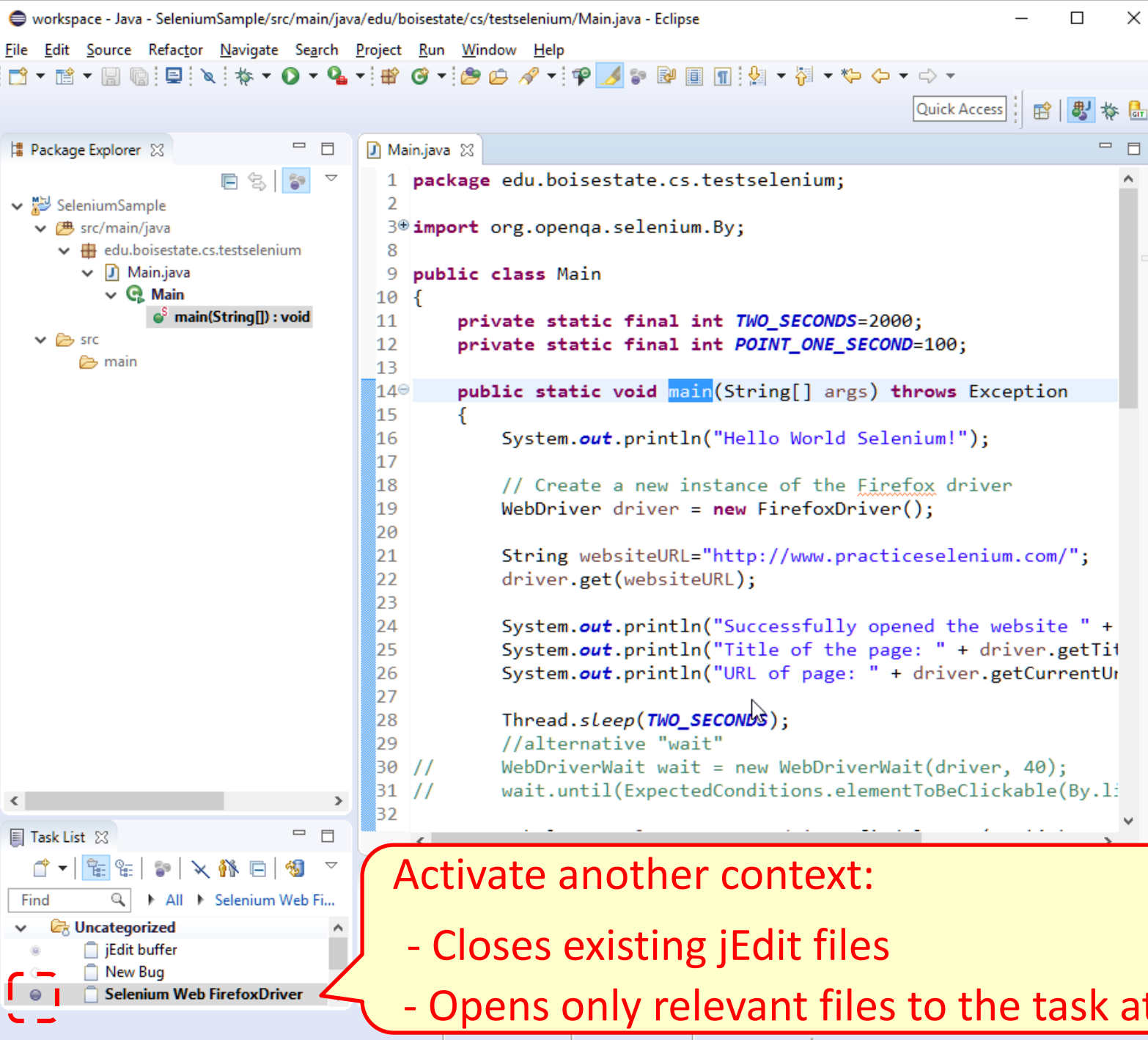
Mylyn

Mylyn

- Eclipse Plugin (<http://www.eclipse.org/mylyn/>)
 - Already installed on Eclipse distributions
- Associate a *context* (i.e., set of software artifacts related to a task) with a maintenance issue
- Helps reduce information overload
- Allows to easily switch context, i.e.,
 - set of files opened for a maintenance task
 - methods/classes already visited







Activate another context:

- Closes existing jEdit files
- Opens only relevant files to the task at hand

Time-travel debugging (TTD)

[Time-Travel Debugging for JavaScript/Node.js \(Demo\)](#), *Earl T. Barr, Mark Marron, Ed Maurer, Dan Moseley, and Gaurav Seth – FSE'16*, <https://github.com/nodejs/node-chakracore/tree/debugging-ttd-preview>

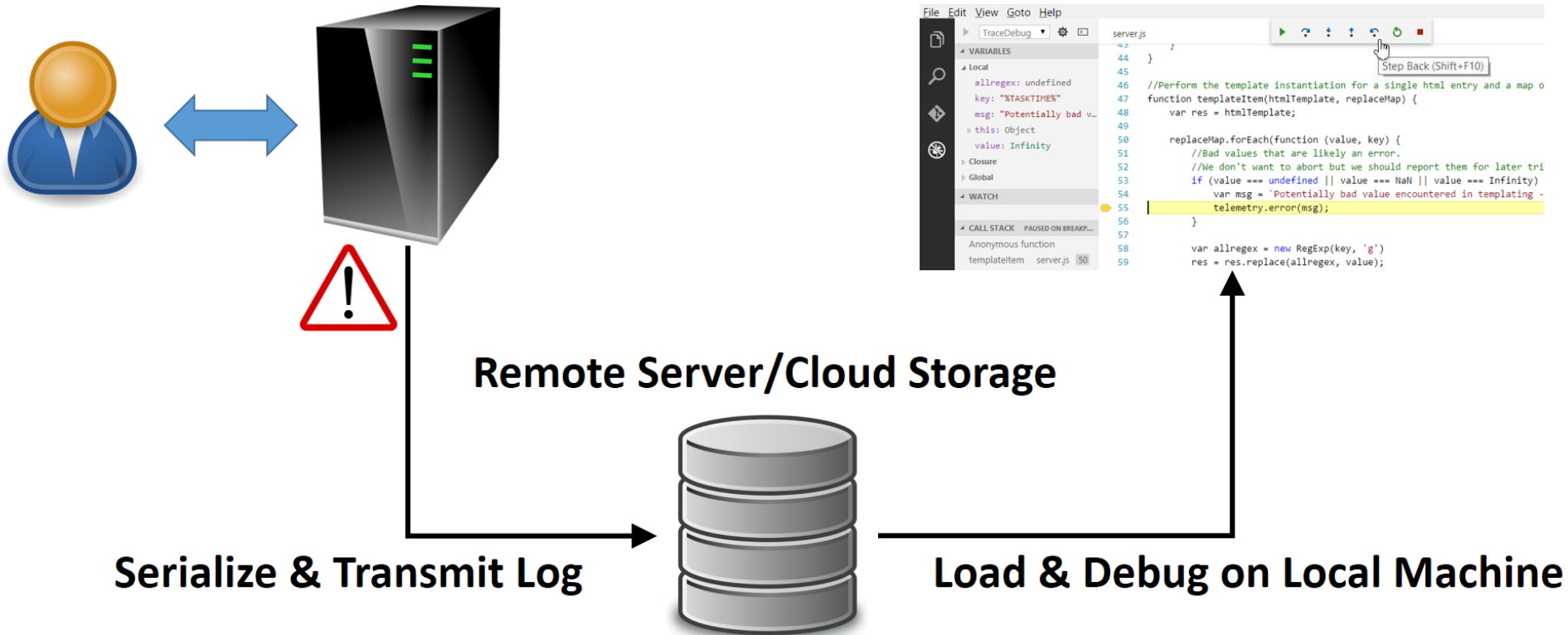
Time-travel debugging (TTD) for JavaScript/Node.js

- **record** and **playback** principle:
 - the **record** mode creates a trace file during execution
 - file can be **played back** allowing developers to deeply inspect the code as it was during the original execution
- look at the faulting code within the full fidelity of the debugger
- all the runtime context preserved

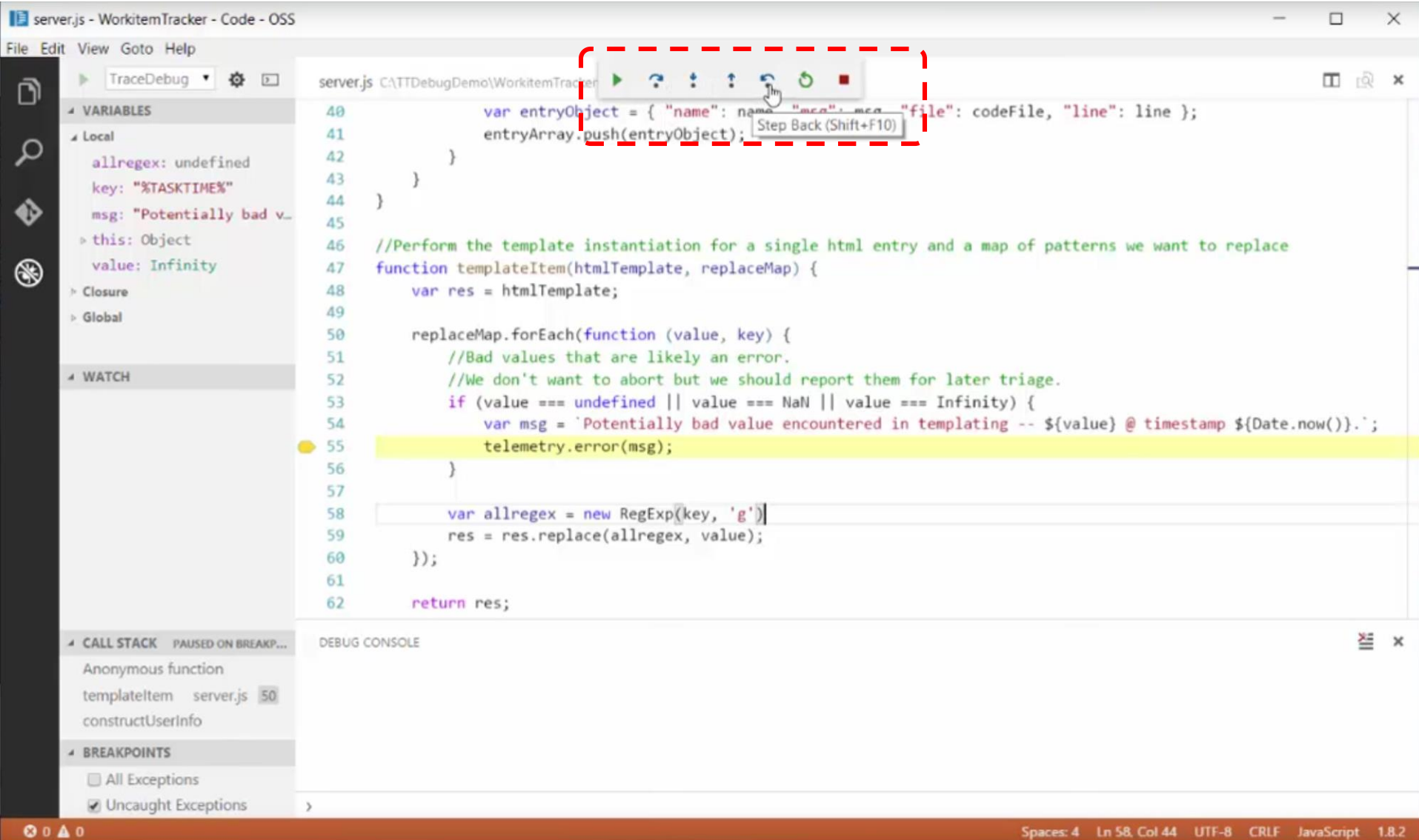
TTD Architecture

```
>node -TTRecord:\\remote\\ttlog app.js
```

```
>TTDebugHost \\remote\\ttlog
```



TTD Demo



Continuous Integration (CI) vs. Continuous Delivery (CD)

Continuous Integration (CI) vs. Continuous Delivery (CD)

- CI = the practice of merging all developer code to a shared branch as often as possible (daily)
- CD = the practice of ensuring that the software from the main branch is always in a deployable state
 - the deployment process should be very fast
 - CD is an extension of CI

Engineering Activities at Facebook

- 1 billion users / month
- 2.5 billion content items / day

Frequency of Deployments

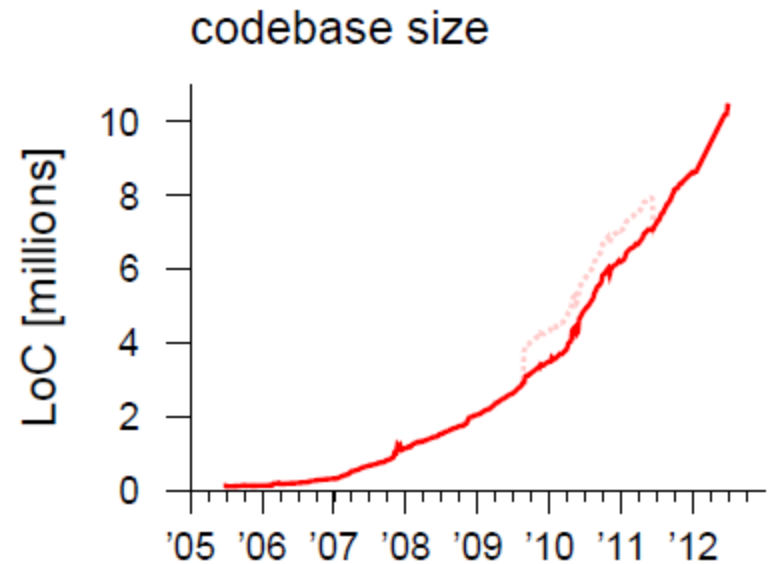
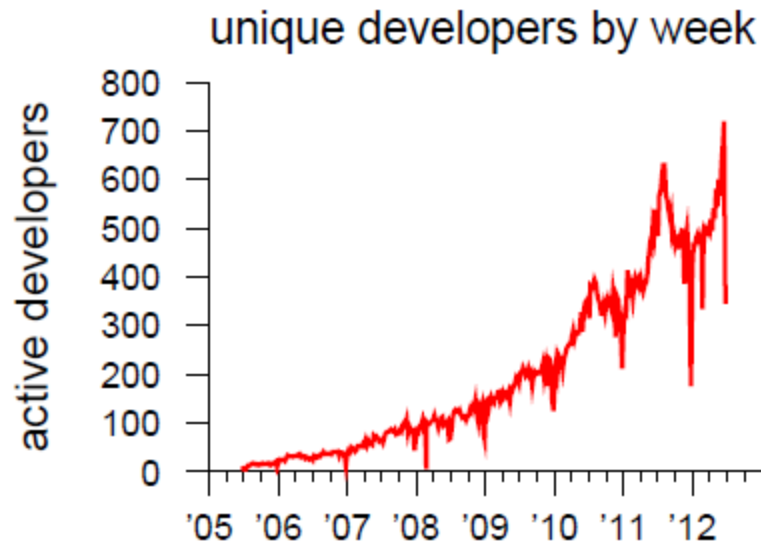
waterfall or unified process	evolutionary development	agile development	Facebook
once	months	weeks	one day

Why Continuous Deployment?

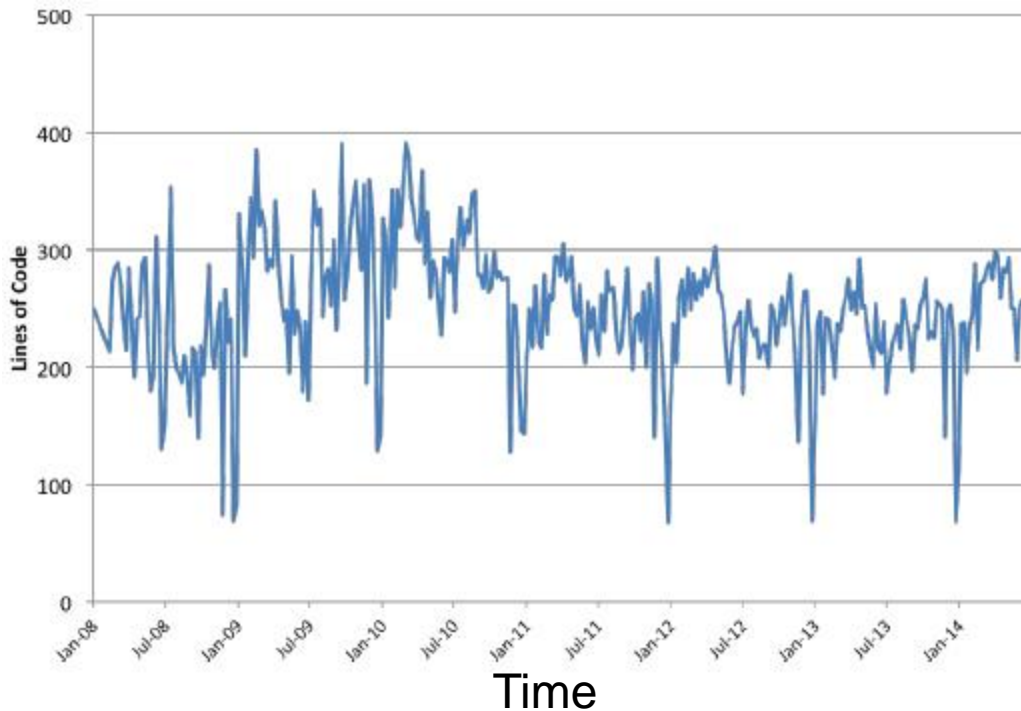
- Frequent deployment imply that each deployment introduces only a limited amount of new code
 - Reduces the risk that something will go wrong
 - Easily identify the source of and solution to problems while they are fresh in engineers minds
- All commits are individually tested for regression
- Deploying code quickly in small increments and without fear enables rapid innovation

Web Frontend: 10.5 MillionsLOC (MLOC)

■ 8.5 MLOC PHP + Python



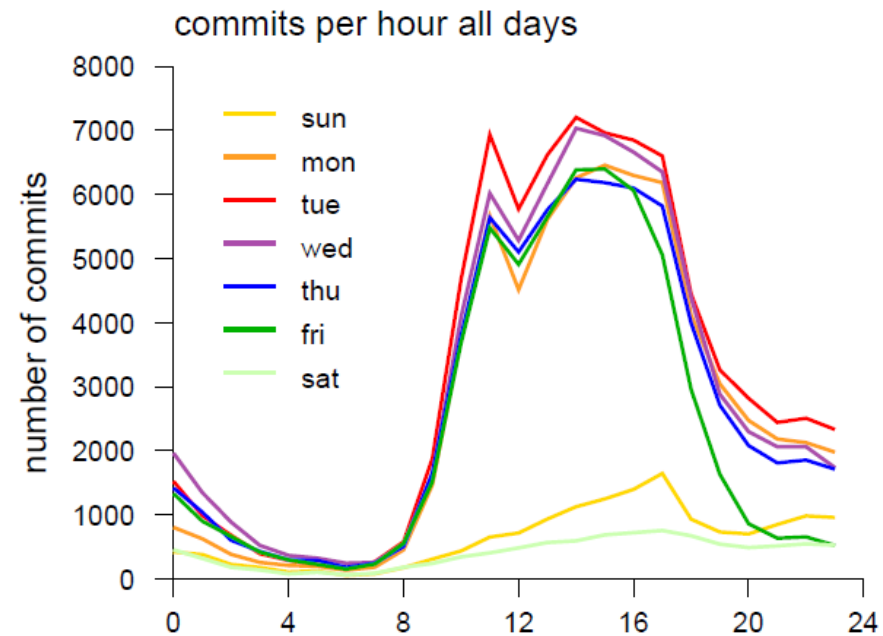
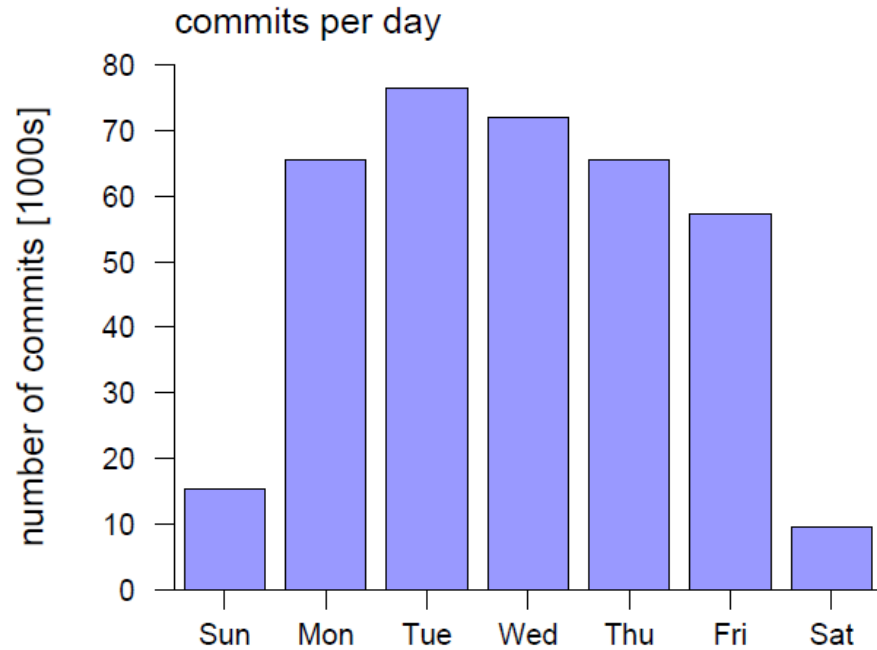
Developer Productivity



Codebase increased 50x

- Each developer releases 3.5 updates to production per week
 - Average: 92 LOC / update
 - Median: 33 LOC / update

Sustainable Work Practices



- Developer are free to [join/switch teams](#)

Perpetual Development

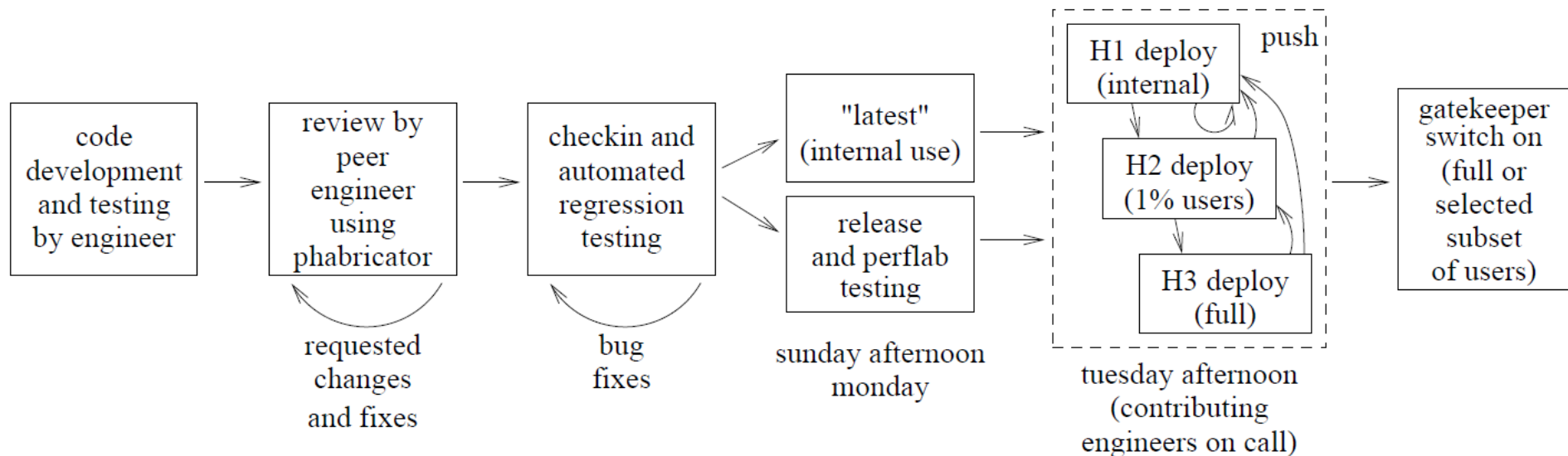
- **Mindset:** “system will continue to be developed indefinitely”
- **Engineers** are also **users**
 - first-hand knowledge of what the system does and what services it provides
- **Live experimentation using A/B testing**
 - Finding what users want, rather than trying to elicit requirements in advance and writing specifications
- In-house usability tests with **user focus groups**

What is the ratio of developers to testers
at Facebook?

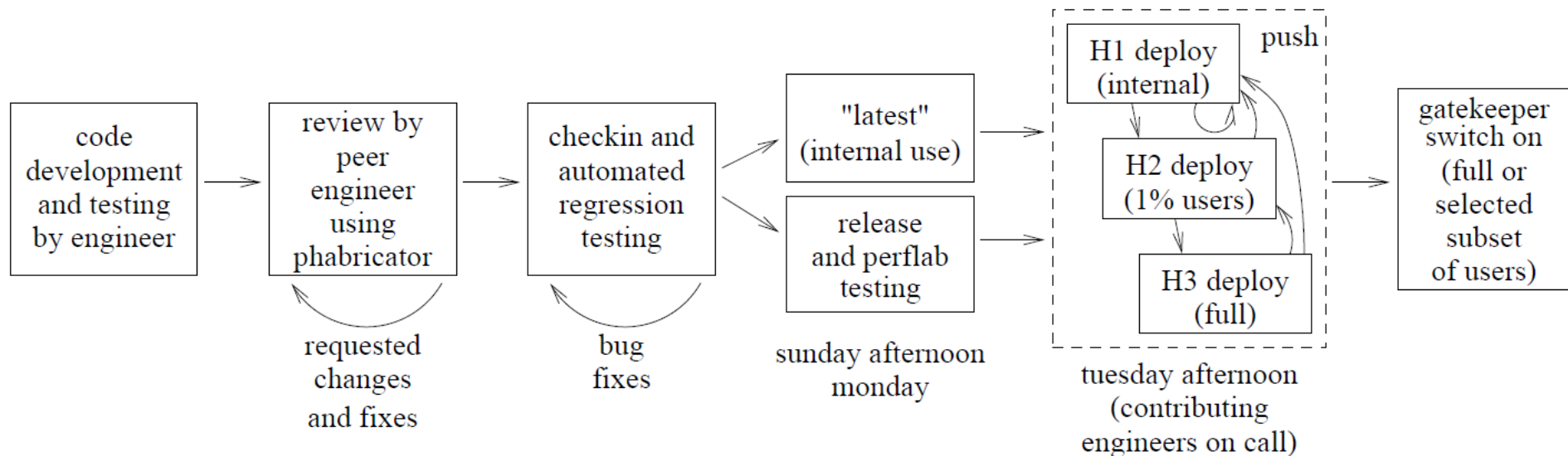
Personal Responsibility

- 1,000 development engineers responsible for
 - Writing code
 - Writing test cases
 - Fixing bugs for their code discovered during regression testing or after deployment
- 3 release engineers
- No separate QA/Testing team

Facebook's Deployment Pipeline



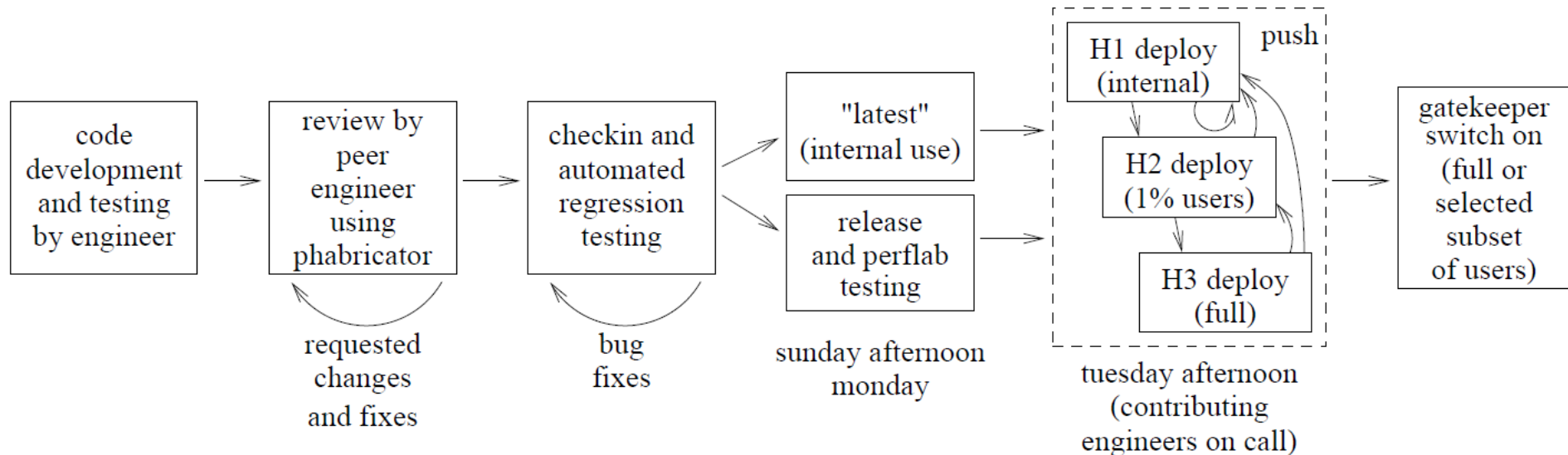
Facebook's Deployment Pipeline



■ Gatekeeper

- Server side feature control
- Gradually release code (specific regions, users, platforms)
- Rollback

Facebook's Deployment Pipeline



■ Gatekeeper

- Server side feature control
- Gradually release code (specific regions, users, platforms)
- Rollback

■ Automation

■ Monitoring ⇒ Metrics

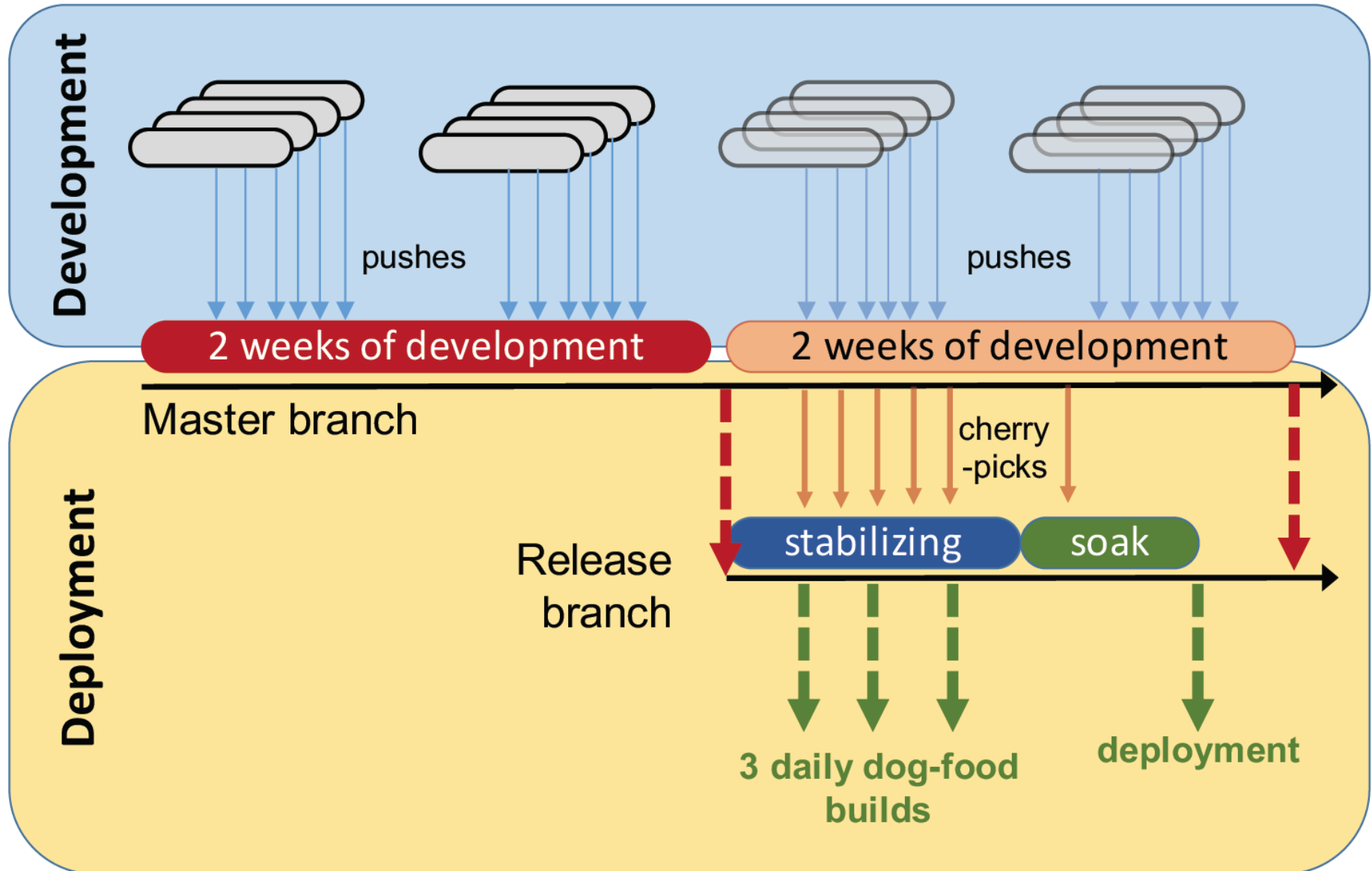
Automation

- 5% of engineers at Facebook develop in-house automation tools:
 - Testing
 - Deploying
 - Monitoring

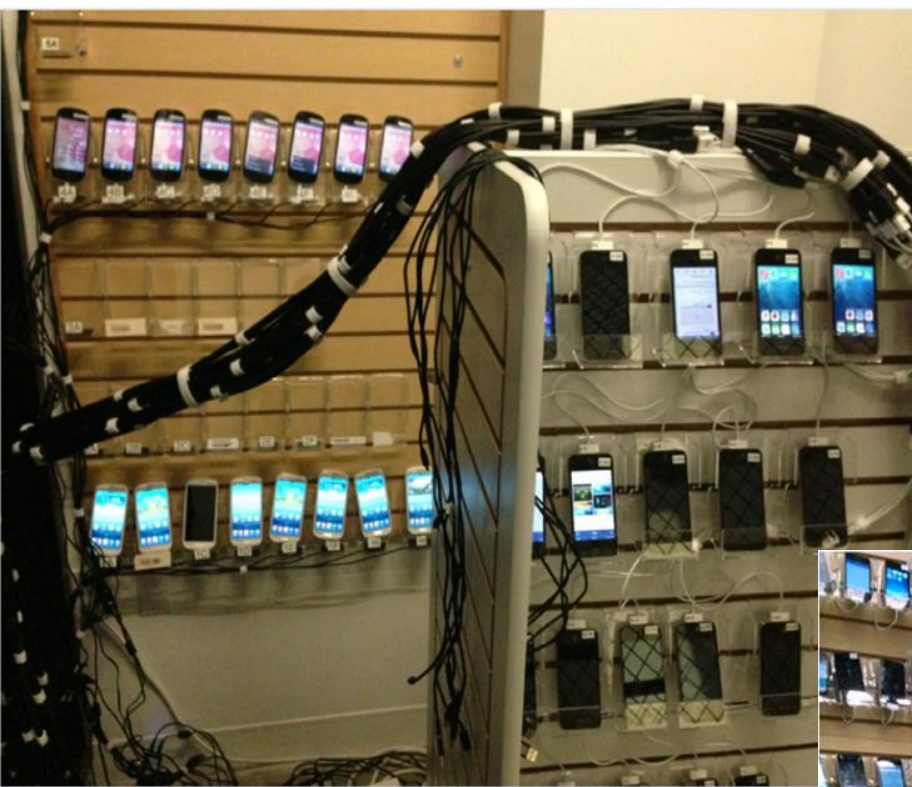
Code Base

System	# of commits 2008 – 2014	LOC added/modified
Web	705K	76 Million LOC
Android	68K	10 Million LOC
iOS	146K	12 Million LOC
Backend	238K	30 Million LOC

Release cycle for Facebook iOS Mobile App



Testing on Real Hardware (2000+ phones)



- Battery consumption
- Scroll performance
- Start times



Bibliography (available in Piazza)

- Dror Feitelson, Eitan Frachtenberg, Kent Beck, "Development and Deployment at Facebook", IEEE Internet Computing 17(4), 2013
<https://research.fb.com/publications/development-and-deployment-at-facebook/>
- Chuck Rossi, Elisa Shibley, Shi Su, Kent Beck, Tony Savor, Michael Stumm, "Continuous Deployment of Mobile Software at Facebook (Showcase)", ACM SIGSOFT: International Symposium on the Foundations of Software Engineering (FSE), 2016 <https://research.fb.com/publications/continuous-deployment-of-mobile-software-at-facebook-showcase/>
- Tony Savor, Mitchell Douglas, Michael Gentili, Laurie Williams, Kent Beck, Michael Stumm, "Continuous Deployment at Facebook and OANDA", 38th IEEE International Conference on Software Engineering (ICSE), 2016
<https://research.fb.com/publications/continuous-deployment-at-facebook-and-oanda/>

Fun Facts about Google

- How many repositories does Google have?
- How many LOC are in those repositories?

Fun Facts about Google

(data was “a few months old” when it was presented during the FSE’16 conference in November’16)

- Monolithic repository

- 2 billion LOC
- 1 billion files
- 9 million unique source files
- most development at head, very few branches

- Lots of commits

- 35 million commits
- 40K commits per workday
- 24K commits by automated systems each day

Fun Facts about Google

(data was “a few months old” when it was presented during the FSE’16 conference in November’16)

- No binaries in repository \Rightarrow compile everything from sources
- Identical work environments for each developer
- Code \Rightarrow Build \Rightarrow Test \Rightarrow Analyze \Rightarrow Review \Rightarrow Release \Rightarrow Monitor

Bibliography

- Conference presentation of the following showcase:
Caitlin Sadowski, "Developer workflow at Google (showcase)", ACM SIGSOFT: International Symposium on the Foundations of Software Engineering (FSE), 2016, <https://dl.acm.org/citation.cfm?id=2994156>

Building high quality software is a
marathon...

A full-page image with a blue color overlay. It depicts a male runner in profile, wearing a white cap, a black and yellow athletic shirt, black shorts, and white running shoes. He is captured in a dynamic running pose on a paved path, with a dense line of green bushes in the background. The word "MARATHON" is superimposed in large, white, bold, sans-serif capital letters across the center of the image, partially overlapping the runner's torso and legs.

MARATHON

...not a hackathon



“Hacking”







Final Exam

- Open-note, take-home exam via Blackboard
- Format is very similar to previous midterms
- The exam is **time limited** (~ 2 hours)
- Assigned ~ Friday (April 27)
- Due next Friday (May 4)

Final Exam

- Material covered in the final:
 - All the lecture materials and class discussions from the beginning of the semester
- Resources:
 - Have a calculator ready
 - [CS471_S18_Sprint2_TemplateQualityPlanAndDefectRemovalModel.xlsx](#)
 - For computing defect removal

Final Exam

- Only provide answers you are confident are correct
- Submitting incorrect answers will result in deducted points

Final Exam

- Only provide answers you are confident are **correct**
- Submitting **incorrect** answers will result in deducted points

Answer	Partial Credit
Correct answer 1	33%
Correct answer 2	33%
Correct answer 3	33%
Incorrect answer 4	-50%
Incorrect answer 5	-50%

Availability

- <https://bdit.youcanbook.me/>
- All assignments/exams will be graded by Monday, May 7
- Grades submitted to registrar Tuesday, May 8