SER516 Software Agility

Module 4: CI/CDe/CD

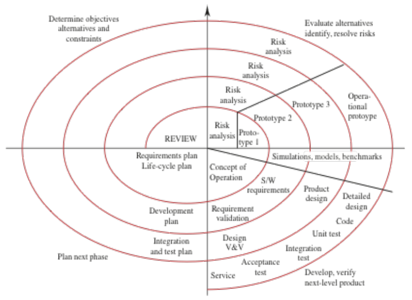
Traditional Delivery Pipeline for Software

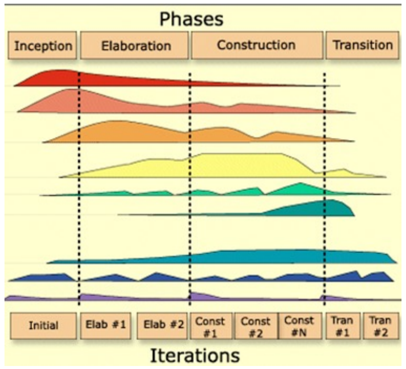
All of our traditional software process models have a “phased” approach to delivery

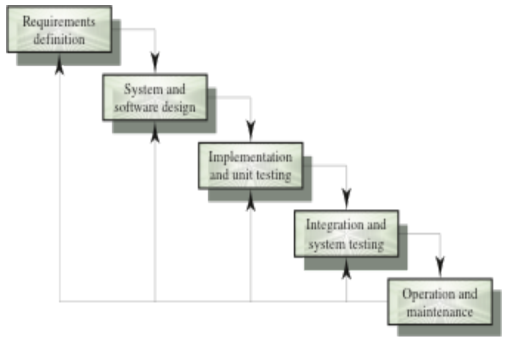
- Each phase transition brings waste
  - Handoffs, transition, relearning
- Phasing also creates “walls”
  - “Throw it over the wall” is the “transition to a new environment” issue and a euphemism for “it isn’t my problem anymore”
  - We have *psychological distress waste* now as well

The idea of being *continuous* did <u>not</u> arise to address this

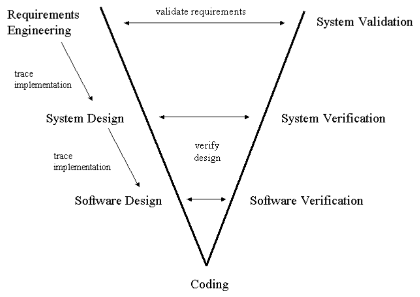
- Managers were already trying to be continuous – as in *keeping their resources continuously busy*
- Remember *resource utilization*?

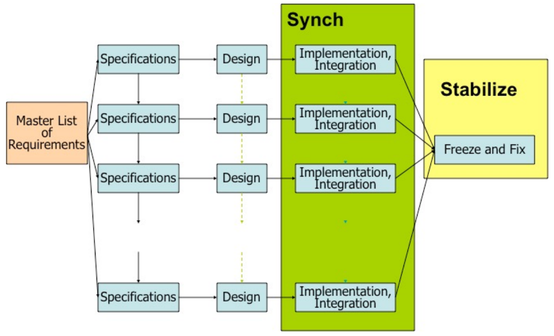






Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance



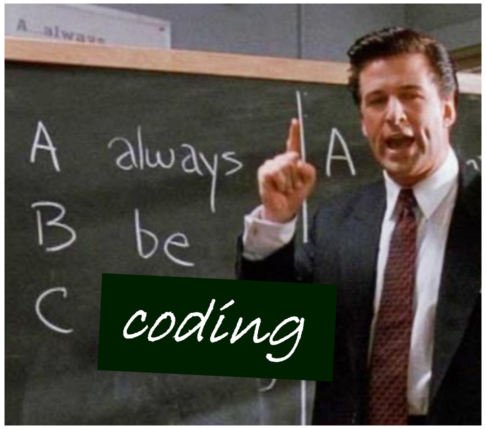


The 5 Cs

If you went to elementary school in Arizona, you would have learned that the AZ economy has traditionally been built upon the 5 Cs: *Copper*, *Climate, Cotton, Cattle* and *Citrus*

In a DevOps world, we have the 5 Cs as well:

1. Continuous Coding
2. Continuous Testing
3. Continuous Integration
4. Continuous Deployment
5. Continuous Delivery



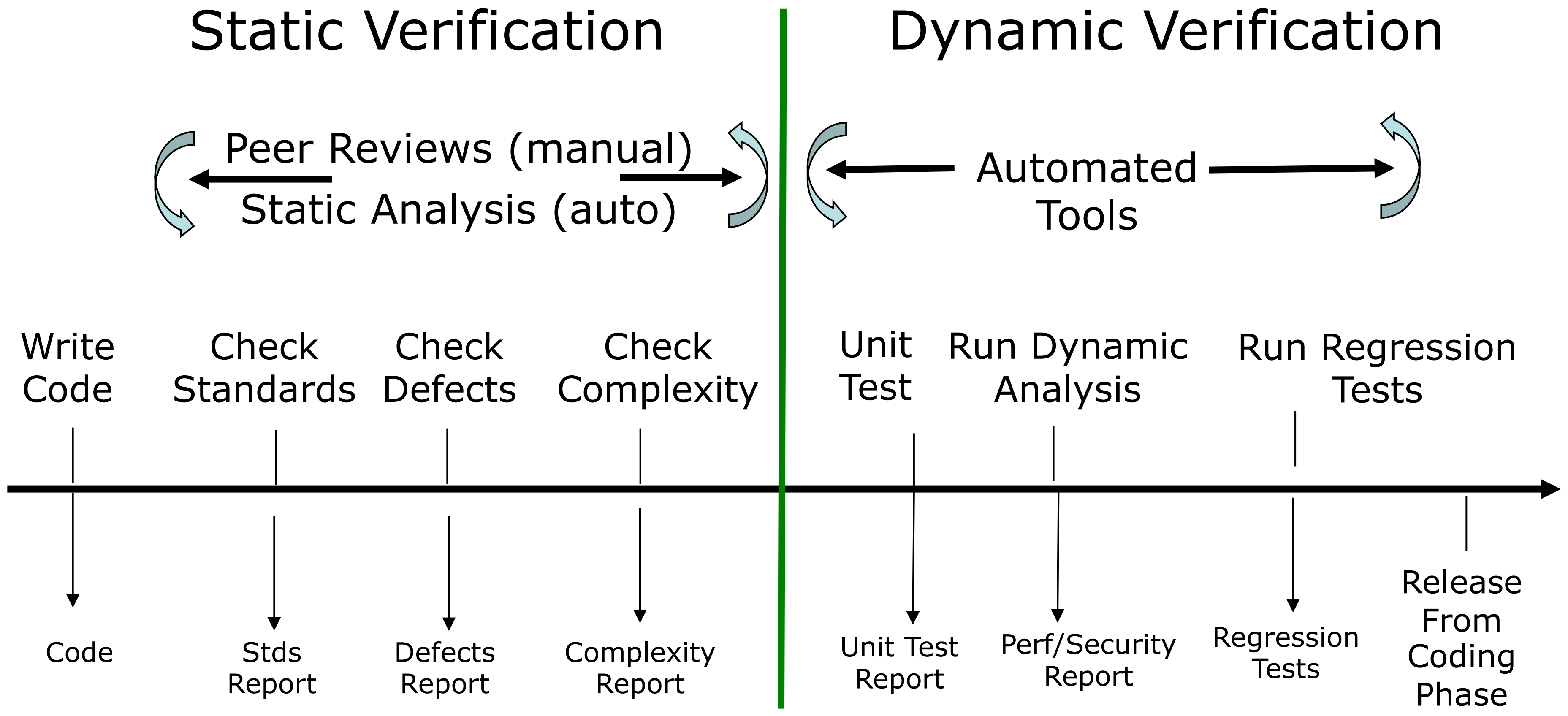
What is different about this kind of *continuous*? It isn't about resource utilization; it is about our *throughput for delivering value.*

Instead of creating phases (walls), we create *pipelines*

walls block flow, pipelines enable it!

Continuous Coding

We want to "Shift Left", so now quality activities done during development just got a whole lot more complicated…



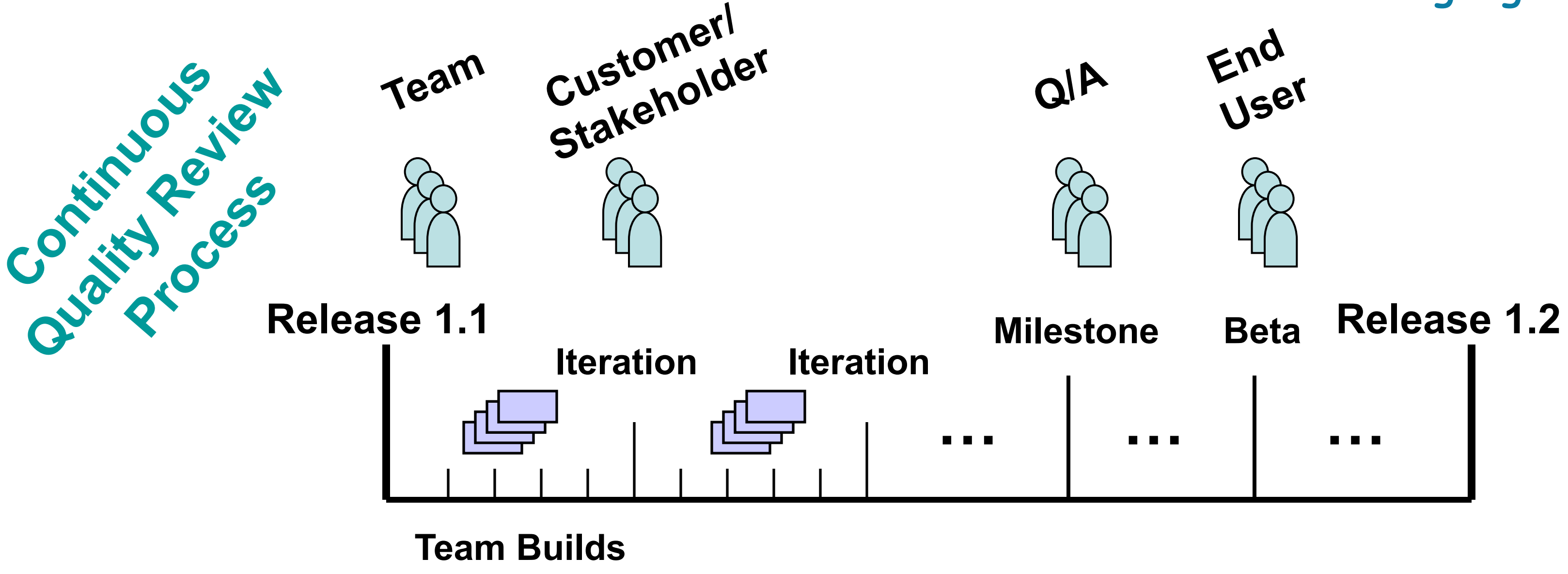
When we talk about implementing "<u>quality policies</u>"

This is what we are talking about – how much of each?

Continuous Testing

- No separate "test" phase – integrate and test continuously
- Features change during release – testing must adapt
- Testing starts on project's Day 1
  - Initial plans, strategies, infrastructure required very early

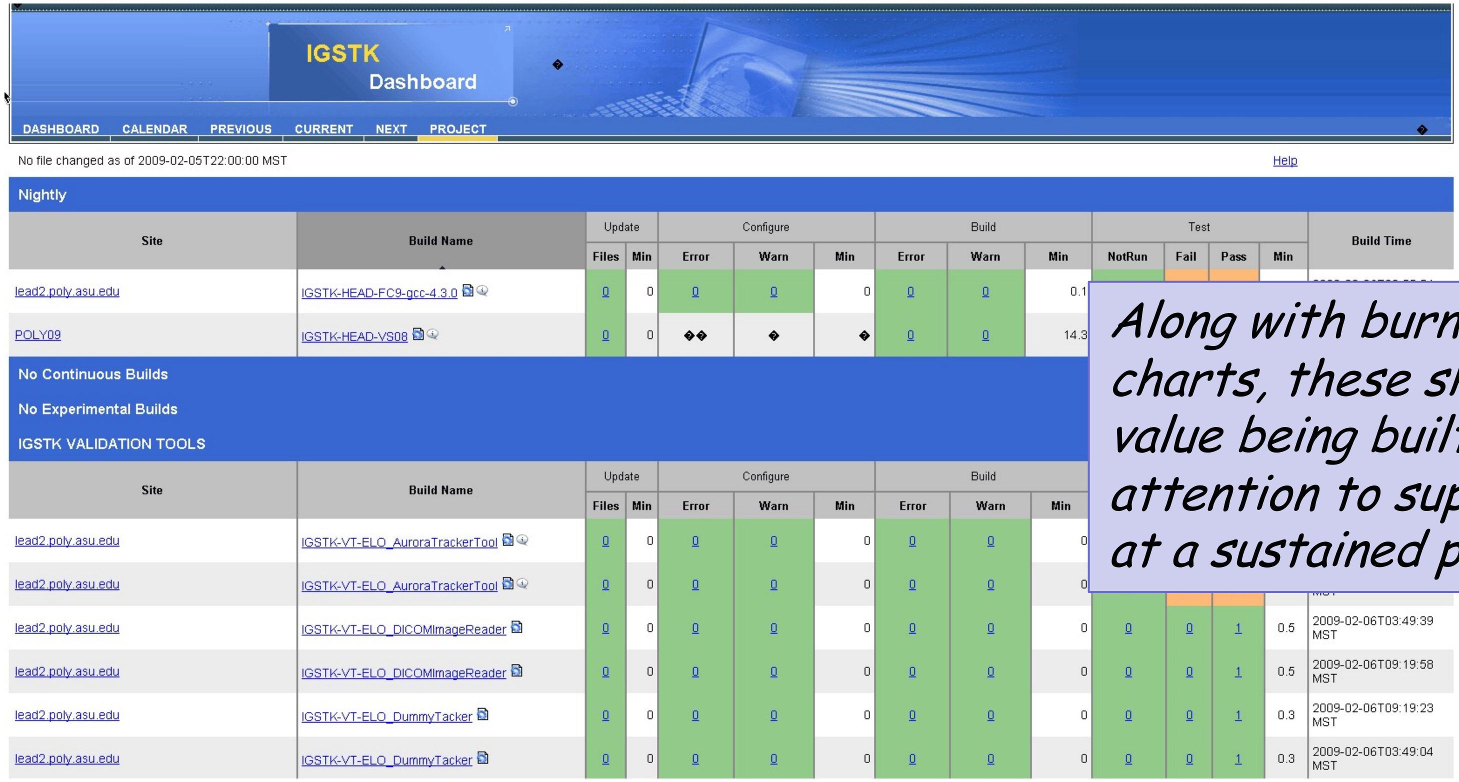
Nightly builds
+ Adaptive planning
+ Continuous integration
= Testing nightmare



Continuous Testing

Unit, System, and Integration tests can be run continuously!

- Requires build/test automation and reporting framework
- Post results to a dashboard for all to see
  - Daily standup in the morning starts by checking if the dashboard is "green"
  - "*WHO BROKE THE BUILD???*" ← don't let this be YOU!



No file changed as of 2009-02-05T22:00:00 MST

Nightly

Site	Build Name	Update		Configure			Build			Test				Build Time
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
lead2.poly.asu.edu	IGSTK-HEAD-FC9-gcc-4.3.0	0	0	0	0	0	0	0	0.1					
POLY09	IGSTK-HEAD-VS08	0	0	��	�	�	0	0	14.3					

No Continuous Builds

No Experimental Builds

IGSTK VALIDATION TOOLS

Site	Build Name	Update		Configure			Build							
		Files	Min	Error	Warn	Min	Error	Warn	Min					
lead2.poly.asu.edu	IGSTK-VT-ELO_AuroraTrackerTool	0	0	0	0	0	0	0	0					
lead2.poly.asu.edu	IGSTK-VT-ELO_AuroraTrackerTool	0	0	0	0	0	0	0	0					
lead2.poly.asu.edu	IGSTK-VT-ELO_DICOMImageReader	0	0	0	0	0	0	0	0	0	0	1	0.5	2009-02-06T03:49:39 MST
lead2.poly.asu.edu	IGSTK-VT-ELO_DICOMImageReader	0	0	0	0	0	0	0	0	0	0	1	0.5	2009-02-06T09:19:58 MST
lead2.poly.asu.edu	IGSTK-VT-ELO_DummyTacker	0	0	0	0	0	0	0	0	0	0	1	0.3	2009-02-06T09:19:23 MST
lead2.poly.asu.edu	IGSTK-VT-ELO_DummyTacker	0	0	0	0	0	0	0	0	0	0	1	0.3	2009-02-06T03:49:04 MST

Along with burndown charts, these show business value being built, with attention to superb quality at a sustained pace

Continuous Integration

What is Continuous Integration?

- Integrate & build the system several times a day
- Integrate every time a task is completed
- Let's you know every day the status of the system

Continuous integration and <u>relentless testing</u> go hand-in-hand

- By keeping the system integrated at all times, you increase the chance of <u>catching defects early</u> (shift-left) and improving the quality and timeliness of your product.
- Continuous integration <u>helps everyone see</u> (transparency what is going on in the system at all times.

*If **testing** is good, why not do it all the time? (continuous testing)*

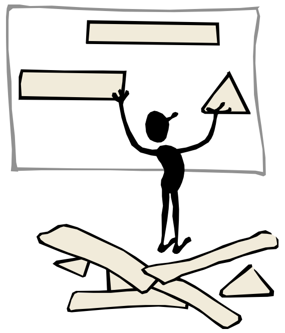
*If **integration** is good, why not do it several times a day? (continuous integration)*

*If **customer involvement** is good, why not show the business value and quality we are creating as we create it (continuous reporting)*

Fowler's 10 Best Practices for CI

http://martinfowler.com/articles/continuousIntegration.html:

1. Maintain a Single Source Repository
2. Automate the Build
3. Make your Build Self-testing
4. Everyone Commits Everyday
5. Every Commit should Build the Mainline on an Integration Machine
6. Keep the Build Fast
7. Test in a Clone of the Production Environment
8. Make it easy for Anyone to get the Latest Executable
9. Everyone can see what's Happening
10. Automate Deployment



How are we doing so far?

"CI & Test" was where Agile got to in roughly the first decade

- Why?
  - Two "neighbors" in our Waterfall/phased model (near in time).
  - Rank-and-file developers probably have a more social familiarity with Testers from QA than they did with anyone else
    - Plebes weren't allowed to talk to customers, and not to ops either
  - Politically, the development team trumped the QA team

DevOps got us the last 2 Cs

- How??
  - Again, a cultural/political revolution, not a technical one
  - Adopting microservices & containers often failed because by themselves they are useful but not game-changing
  - Politically, Dev & Ops were more evenly matched (although I would give Ops the edge)

Continuous Delivery

Back to Martin Fowler:

"Continuous Delivery is … where you build software in such a way that the software can be released to production at any time."

You're doing continuous delivery when: (Thoughtworks)

- Your software is deployable throughout its lifecycle
- Your team prioritizes keeping the software deployable over working on new features
- Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them
- You can perform push-button deployments of any version of the software to any environment on demand

This gets us to the point where Dev, QA, and Ops are all working together to create a *production-ready product*

- But it is not deployed yet, it is just *deployable*
- Deployment waits for another (usually manual) trigger

Continuous Deployment

In its simplest form, it is the automating of the “push-button” human step of Continuous Delivery

But there is more to consider…

- Continuous Deployment requires an ability to take binary packages and deploy them to a myriad of operating environments automatically
  - No human operator needs to login and do something (run a script)
  - If the 1st line of your deployment automation is “git clone” then you are doing it wrong!
- Continuous Deployment requires Continuous Delivery

Strategies for Continuous Deployment

- Naïve – just roll out to all nodes and environments at once
- Rolling deployments – incremental rollouts to batches of nodes with multiple service in synch
- Blue-Green – maintain 2 environment copies and employ two-bin
- Canary deployments – incremental but by user base not by node
  - Partition nodes by users, or services by users

Putting it all together

Dr. Gary's Law of Delivery Pipelines:

$$\sum_{k=1}^{k=5} \mathtt{C}_k > \{ \mathtt{CC}, \mathtt{CT}, \mathtt{CI}, \mathtt{CDe}, \mathtt{CD} \}$$

Yes, the sum is in fact greater than its parts!

The CI/CD/CDE "pipeline" discussed in the literature and in the practice is where modern organizations are today

- Automation infrastructure is *enabling*
  - This is at every step of the C*, even coding to some extent
  - Modern cloud infrastructures and tooling platforms process this pipeline automatically on *your trigger events*
    - Events: *time, commit, feature*
    - Are you at the point where you no longer have a human in the loop?
- Culture is *necessary*
- *What is next?*