



# Always Playable

Continuous Delivery as a game development philosophy

# About myself

2009-2015



2015-2018



2018-Present

## DAREWISE

- Studio opened in 2018
- 30 employees
- AAA Veterans



PROJECT C launched in pre-alpha - <http://project-c.darewise.com>

Let's start a game studio

# Pillars and Philosophy

- High quality games
  - Need to be more efficient to compete
- Fast Iteration
  - Best way to ensure quality in the product
- State of the art tools and processes
  - Quality of life increases quality of the product
- Lean startup philosophy and culture
- Always playable
  - Stability allows iteration
  - Investor or publisher may be in the room right now! What can you show?
- But we start with...
  - ...no team, no time, no budget
  - Must be operational ASAP



# Infrastructure and Tools

# Off the shelf

- Use SaaS tools
  - Managed services = no setup, no maintenance, no outages
  - Use centralized Authentication
  - Not fully customizable but good enough
  - Lease everything, don't immobilize cash
- There's an app for everything
  - Mail and Auth: Office 365 or GSuite
  - HR: Payroll (Payfit), Expenses (Revolut for Business), country-specific...
  - Project Management: Jira, Confluence, Trello, Hansoft...
- Build on top of cloud services
  - Credit programs for startups
  - Workstations as a service = no hardware to buy
- When you outgrow these you'll have the money to build a bespoke solution



# Bread and butter

- Most people work on the Game Client + Server
  - Optimize the user experience for them
- Don't forget satellite projects
  - Online backend services, DevOps scripts, Pipeline Tools...
- Simplest possible project setup
  - Install source control + Get latest
- Self-contained project
  - Include all sub-projects and dependencies
- Do not build fancy tools now
  - Simple DOS/Python scripts if necessary
  - Source control is made to manage dependencies (submodules or equivalent)
  - If you build anything, build tools on top of source control



# Bread and butter

- Source control is the only infrastructure you need
  - Store all critical data in source control and back it up
  - Treat other (local) data stores as temporary
- Source control tools and integrations, use them!
  - Code review
  - Cloud hosting
  - Backup procedures
  - Multi-site solutions for outsourcing
  - Automation
    - Triggers: validate commits and automate processes
    - Continuous Integration and DevOps
- Abuse source control
  - Store all your build artifacts
  - Store all your work files (DCC files, FBX files etc)
  - ...

# Self-contained project



## Notes

- Separate code and game data
- Too large, only sync relevant files
- Too large, only sync relevant files
  - Shared folder is also fine here

## Who needs it?

Coders

All

Artists

Artists

All

Coders

Coders

QA

# Improving Iteration time

# Case Study

Code Iteration: Time before a code change is deployed

	Real studio #1	Real studio #2	Real studio #3	Darewise
Local code change	Hot-Reload	Rebuild time	Rebuild time <10 minutes	UE Hot Reload <1 minute
Gameplay Code to Other Programmer	Rebuild time	Review + Approval + Tests + Rebuild time 1 hour min	Rebuild time <10 minutes	Review + Approval + Rebuild time <2 minutes if lucky
Engine Code change to Gameplay Programmer	Blocked waiting for engine branch integration, stabilisation and tests 1 week average	Blocked waiting for engine branch integration, stabilisation and tests 1 month average	Rebuild time <20 minutes	Unreal full rebuild... <30 minutes ... still waiting
Code change to Non-coder	1 Gameplay Build / wk 2 wk for engine change	Two builds per day	CI Rebuild and submit <15 minutes	CI Rebuild and submit <5 minutes

# Case Study

## Game Data Iteration: Time to test a change

	Real studio #1	Real studio #2	Real studio #3	Darewise
In-editor change	Real-time Edit/Debug while playing in editor	Real-time Play in editor must be stopped	Real-time Play in editor must be stopped	Real-time Edit/Debug while playing in editor
External tool change	Export 1-3 minutes	Export + Restart Editor 1-5 minutes	Auto-reimport <1 minute	Auto-reimport <1 minute
Game data change to Someone else	Instantaneous Sync from editor	Close editor + Sync + Reimport all OR Two builds per day	Instantaneous Sync from editor	Instantaneous Sync from editor
Test on target environment	Always cooked and packaged, just deploy ~5 minutes	Cook + Package + Deploy ~1 hour	Cook + Package + Deploy ~30 minutes	Cook + Package + Deploy ~30 minutes

# Improving Iteration time

- Iteration time is limited by Game Engine and SCC/CI processes
- Unreal has long compilation times and cooking
- Engine and Tools: Optimize iteration time and workflow
  - Use cooking as optimization step, not as mandatory step
  - UX is \$\$\$, invest in UX, great ROI
- Language
  - Low level: optimize for performance (C, C++, Rust, Jai...)
  - High level: optimize for iteration time
    - Visual scripting: empower your designers
    - Can be optimized or hardcoded later
    - Communication tool: bad script better than good doc
- Everything must be automatable
  - Corollary: Automate everything you can

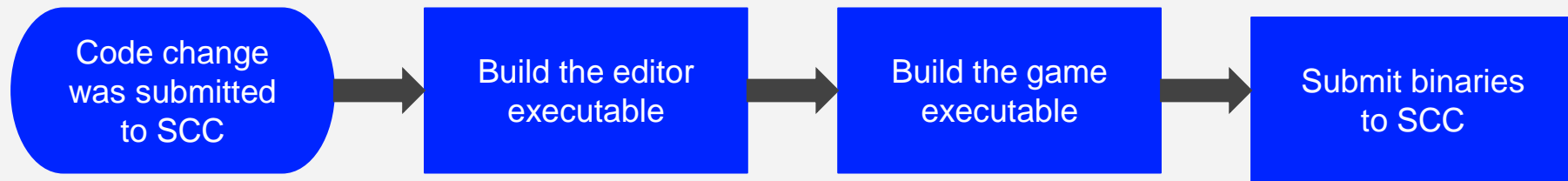
# Improving Iteration time

- Eliminate noise
  - As little context switching as possible
  - Fully featured editor is better than constellation of tools
  - Consistency of UX between tools
  - Source control integrated in editor
  - Minimize use of external tools
- Essential workflows
  - Play in editor
  - Run Client & Server in editor
  - Run Client & Server cooked locally before committing
  - Development Game Server infrastructure
- Local workflow is 99.9% of development
  - Editor and final target as close as possible
  - Bugs should be reproducible locally
  - Bugs that are only present on target environment may be impossible to debug

# Continuous Integration And DevOps

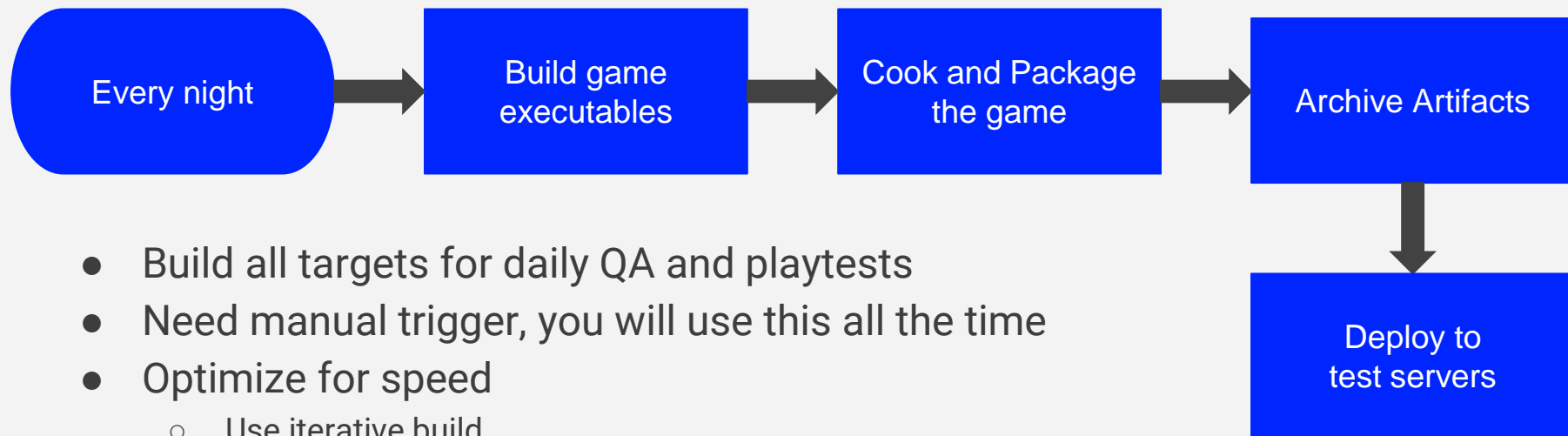


# Fast Iteration Job



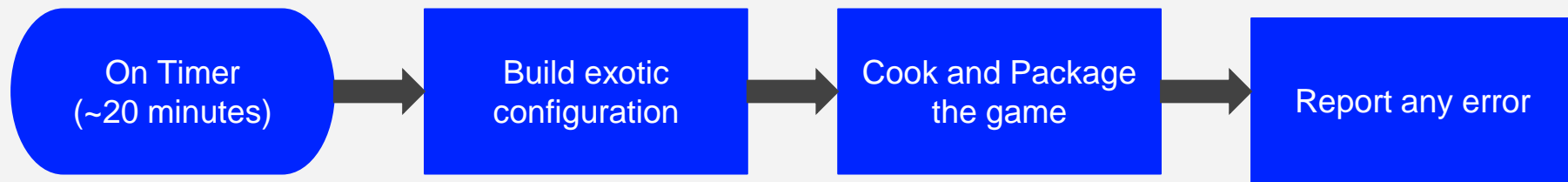
- Most important job
- Main improvement for iteration time
- Do not perform tests before committing
- Optimize for speed
  - Only build required targets for local workflows
  - Use iterative build
  - Clean/Rebuild at night

# Nightly Build Job



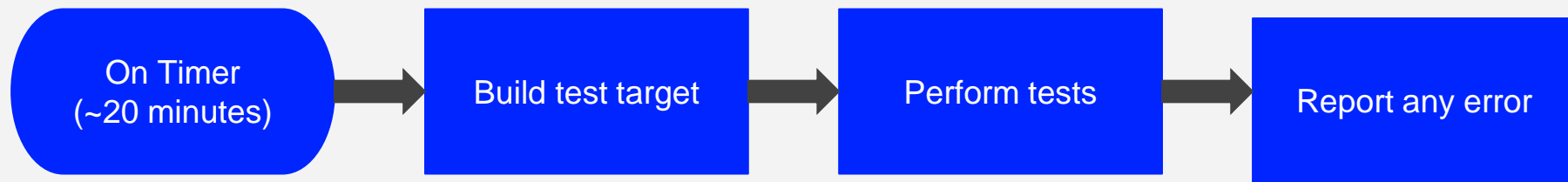
- Build all targets for daily QA and playtests
- Need manual trigger, you will use this all the time
- Optimize for speed
  - Use iterative build
  - Clean/Rebuild every week
- Backup the archive

# Exotic Configuration Job



- Examples: Mac, Linux, Consoles, Debug, Shipping...
- Visibility of errors
  - Email alerts don't work
  - Use a dashboard on a big screen
- Job needs to run often enough but not immediately on change
- Add these jobs even if you don't plan to support configuration
  - If you change your mind you won't regret it
  - Very easy fixes if caught early

# Automated Tests Job



- TDD is hard to apply to gameplay code
- Good candidates for TDD/Unit Tests
  - Core low-level libraries
  - Online micro-services
  - Critical layers and services (database, authentication, payment...)
- Tests should be performed post-commit
- Parallelize test jobs

# More ideas

- Static code analysis
  - Critical if working with c++
  - Treat defects as bugs
- Metrics and reports
  - Server load tests
  - Client performance tests
- Automate Everything...



**KEEP  
CALM  
AND  
AUTOMATE  
EVERYTHING**

# Which tools and languages to use?

- Use whatever works with your source control and hosting
  - Jenkins is industry standard
  - GitLab and TeamCity have good reputation
- Languages
  - Prefer Python to bash/DOS
- Virtualize your build system early
  - You will need to scale
  - On-premises and on the cloud
- Use infrastructure as code
  - Keeps jobs consistent with code
  - History/Diffs/Rollback of CI Jobs
  - Easier to deal with branch specificities
  - Invaluable when codebase gets fragmented

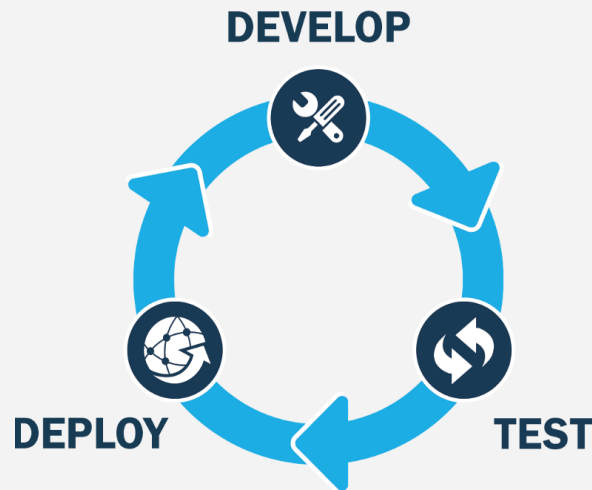


**Jenkins**

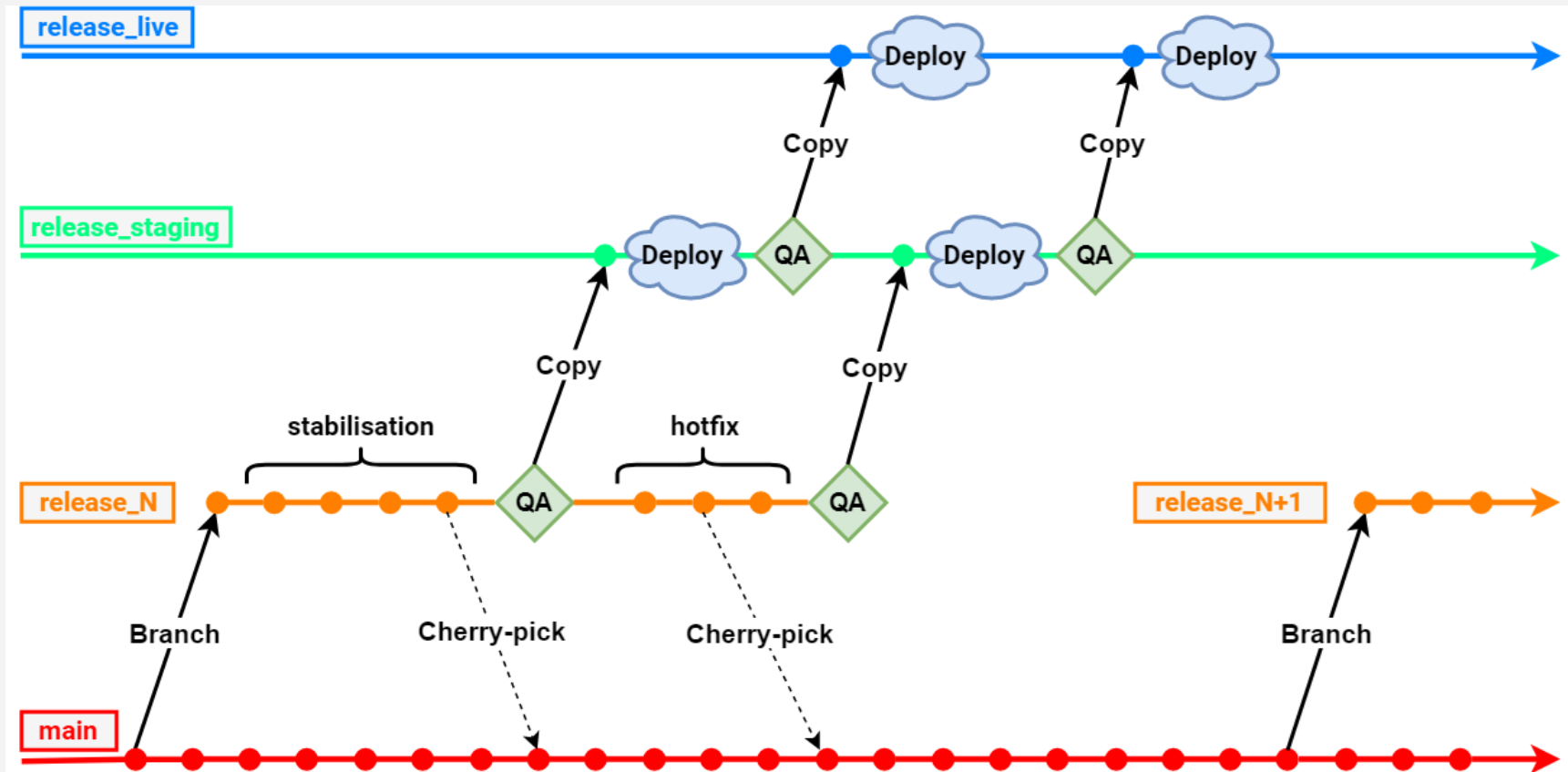


# For multiplayer games

- **Development Servers**
  - Nightly build
  - Playtest build
  - Latest stable build
- **Preparing for release**
  - Development environments will be different from release
  - Dependency with online services
  - Client delivery method (Steam, Launcher VS Shared Folder)
  - Create a staging environment identical to live
  - As early as possible
- **Release process**
  - Automate as much as possible
  - Need human validation steps



# Branching policy





# Which source control to use?

# Which source control to use?

- Surprisingly not obvious question

- You MUST use source control
- No excuses
- Dropbox is not enough

## Version control for game development - issues and solutions ...

<https://gamedev.stackexchange.com/.../version-control-for-game-development-issues-...> ▼

6 answers

Nov 30, 2015 - It's a distributed **version control** system. This allows for us to have our own independent trunk-like area. I can work in my own area and invite you over to view ...

assets - **Source control** for storing everything of game ... 3 answers 24 Mar 2013

architecture - Finding the right directory structure for ... 3 answers 26 Jul 2012

**Version control** with **game development** - When should ... 4 answers 14 Jan 2012

**What Version-control** systems work best with games ... 8 answers 14 Jul 2010

More results from gamedev.stackexchange.com

## Plastic SCM - The version control for game developers

<https://www.plasticscm.com/games> ▼

Plastic SCM is the **version control for game development**. Learn why studios worldwide choose Plastic SCM.

## Which source/version control system do you use for your projects ...

[https://www.reddit.com/.../gamedev/.../which\\_sourceversion\\_control\\_system\\_do\\_you...](https://www.reddit.com/.../gamedev/.../which_sourceversion_control_system_do_you...) ▼

Apr 10, 2018 - 26 posts - 24 authors

TFS lets you use TFS or GIT as a **source control** and still use all the other tools. I use TFS ... I've always used Git for projects; **game dev** or not.

What do you use for **version control** for art files? 3 Oct 2018

How should I go about using **version control** for my project ... 27 Aug 2015

What's a common **version control** setup in large game studios ... 27 Feb 2015

**Which Version Control** Software do you use? 8 Jun 2014

# Git



- Ubiquitous
- Decentralized
- Free and open-source
- Very bad at handling binary files
  - Even with Git-LFS
  - Basic File-locking introduced recently
  - No single-file operations!
- Notoriously bad user experience
  - GUI tools are all lacking
  - Need to use confusing command line
- Best cloud hosting solutions
- Best integrations and ecosystem
- You will need Git to collaborate with 3rd parties



# Perforce

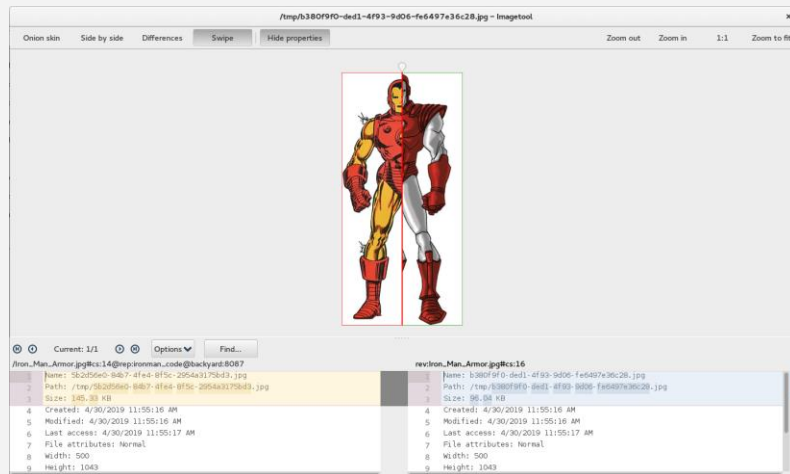
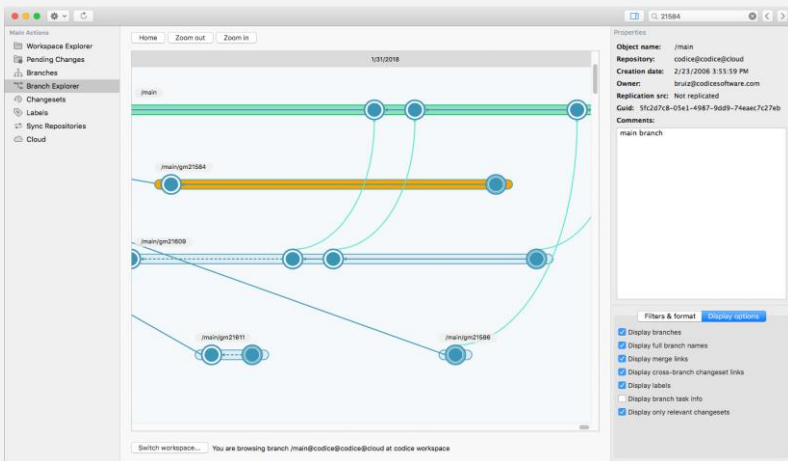
- Industry standard for Games
- Centralized, single-file versioning
- P4 DVCS
- Best for large projects and binary files
- Most powerful GUI tools
- Integrations
  - Smaller ecosystem than Git or SVN
  - Git-Fusion: works with some caveats
  - Helix4Git: Git repositories hosted on P4 server
- Administration
  - Steep learning curve but very powerful. Requires self-hosting.
  - Deep customization: triggers, exclusive locking...
  - Enterprise features: access control, edge servers for outsourcing sites...
- Very expensive

# PERFORCE



# Plastic SCM

- “The version control for Games and big projects”
- Best UX for users and admins
- Two different GUIs
  - Gluon: artist-friendly, file-based with exclusive locks
  - Plastic: for programmers with state of the art toolset



# Plastic SCM



- Semantic merge available as standalone tool
- Major drawback
  - A workspace must be either graph or file-based, not both
  - Separate code and data workspaces works best
- Integrations
  - Smallest ecosystem
  - Native integration with git ...
  - ... with some caveats
- Less customizable than P4
- Much cheaper than P4



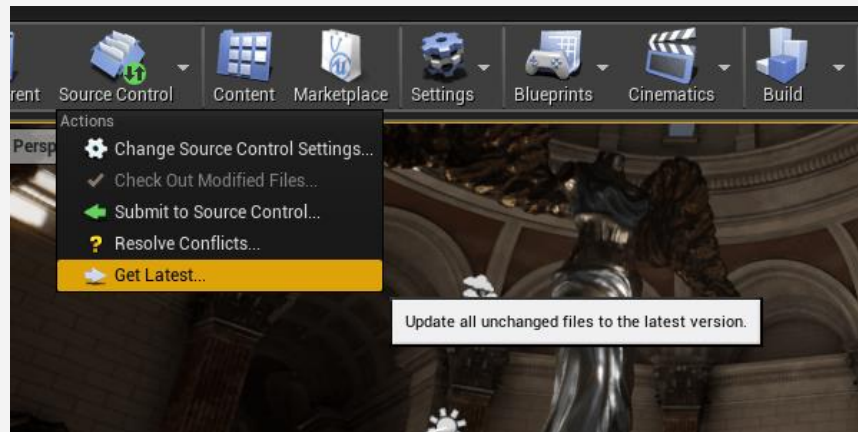
# SVN

- Centralized, single-file versioning
- Free and open source
- Good GUI tools available
- Integrations
  - Integrated in Unity and Unreal
  - Smaller ecosystem than Git, but bigger than P4
- Reportedly bad at handling large projects
  - Perforce is known to perform much better
- Many cloud hosting solutions
- Popular choice for indies
- Perforce beats it in almost every way except cost







# Other

- Mercurial
  - Has Git-LFS equivalent
- Unity Collaborate
  - Git Backend
  - Walled garden
  - No access to backend
  - Cannot be integrated in CI
- GitHub for Unity
  - Unity - Github specific
- GitCentral
  - Unreal specific
  - Centralized
  - File-based
  - Git-LFS backend





# Breakdown

	Centralized	Model	Binary files Scaling	UX	Ecosystem	Cloud hosting	Pricing
Perforce		File-based	++	+	+	- Assembla	5 users free \$\$\$u/year
SVN		File-based	-	-	++	+	Free
Git		Graph	--	--	+++++	++	Free
Plastic		Graph or File But not both	++	++	+	- Plastic cloud	23.25\$/u/mo 595\$ perpetual

# Take-away

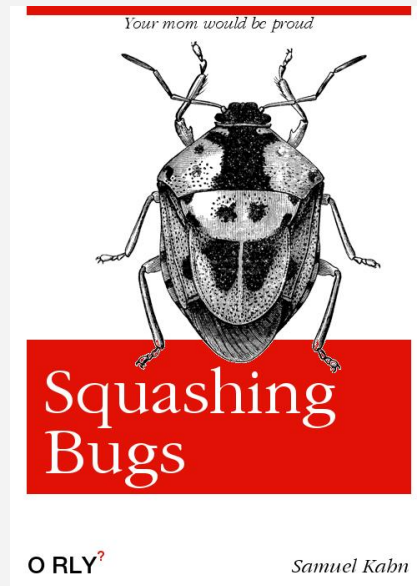
- Use source control!
- Use Perforce if you can afford it
  - Still the best solution today
  - Watch out for Plastic SCM
- Using Git with cloud hosting
  - < 5 users - Azure DevOps
  - <10 GB - GitLab
  - >10 GB - BitBucket
- Contact me if you want tailored advice on this topic
- Using Unreal and on a budget, try GitCentral



# Always Playable

# Always Playable

- The best infrastructure is not enough...
- ... you need culture and discipline
  - Enforce discipline from day one
  - Almost impossible to create this culture later
- Do not let instability accumulate
  - Regression, crash, assert or test failure is treated as blocker
  - High priority bugs planned to be fixed in the sprint
  - Debug day each sprint
  - Boy scout rule: you find it you fix it (or escalate it)
- Peer review everything (code and data)
- Mandatory weekly playtest
  - Is only possible if the game is stable
  - Discover issues earlier and improve the game
  - Increase in overall quality



# Always Playable

- Downsides
  - If you have continuous integration with post-commit tests...
  - ... bugs and crashes will get continuously delivered as well!
  - The gain in iteration time is worth it if you make sure failure is treated immediately
- Slow down production velocity
  - Sacrifice some iteration time to introduce review processes
  - Stability before features slows down tasks...
  - ... but bugfixes take less time now than they will later
- Better overall results
  - Better stability improves iteration time and quality of life
  - Fast iteration time and QoL leads to higher quality result
- Real gains but hard to quantify to team and management
  - Manage team frustration with the processes
- It's up to you now!

# Thank you



[samuel@kahncode.com](mailto:samuel@kahncode.com)



[@kahncode](https://twitter.com/kahncode)