2016-05-01

Architecting Out Software Intellectual Property Lock-In: A Method to Advance the Efficacy of BBP

Berardi, Chris; Cameron, Bruce; Sturtevant, Dan; Baldwin, Carliss; Crawley, Ed

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/53538

Architecting Out Software Intellectual Property Lock-in:

a method to advance the efficacy of BBP

Chris Berardi, Major, USAF

Coauthors:

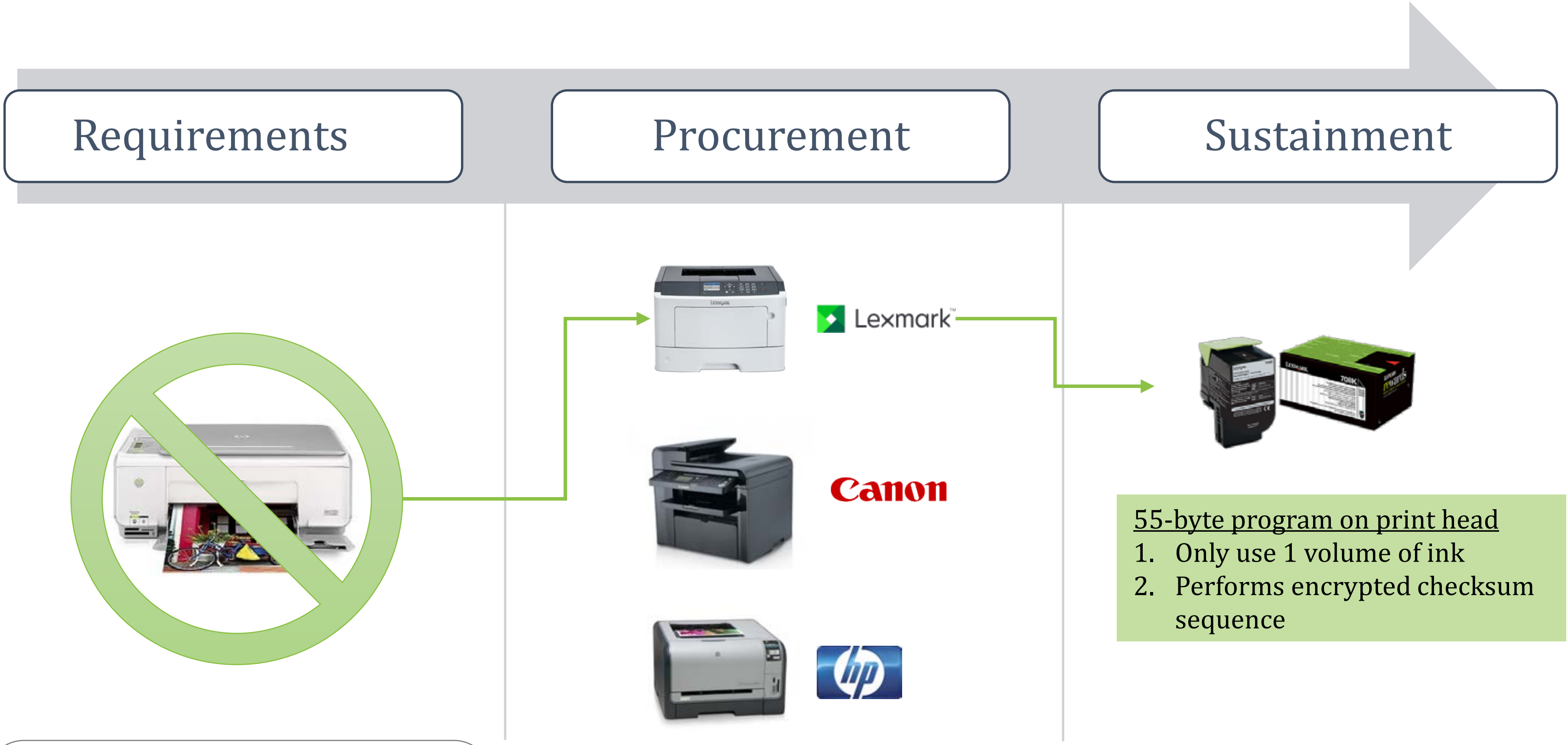
Bruce Cameron, MIT

Dan Sturdevant, Silverthread Inc.

Carliss Baldwin, Harvard

Ed Crawley, MIT

Motivation



55-byte program on print head

1. Only use 1 volume of ink
2. Performs encrypted checksum sequence

Requirement

Printer stops working while printing urgent document (PhD Thesis, presentation slides, etc.)

Procure new printer:

1. Conduct market research
2. Conduct trade analyses
3. Select lowest cost printer

Sustainment Plan:

Utilize third-party parts to save life-cycle costs

- Attempt results in failure

Problem Framing

Intellectual Property (IP) direction in law:

> *"The Secretary of Defense shall require program managers for major weapon systems … to assess the long-term technical data needs of such systems and subsystems and establish corresponding acquisition strategies that provide for technical data rights needed to sustain such systems and subsystems over their life cycle"*

10 U.S.C. § 2320(e)

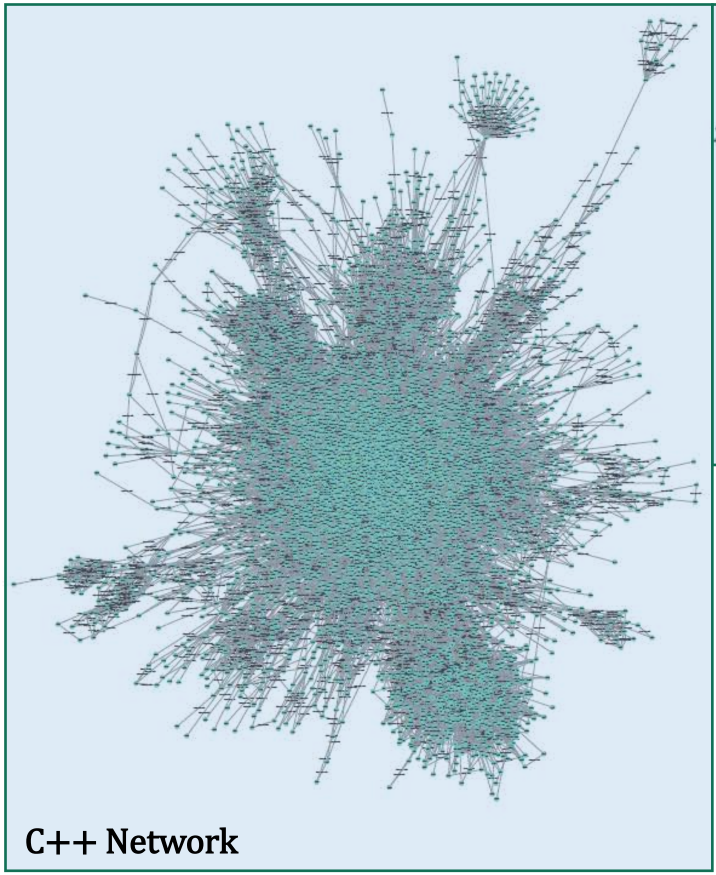
Intellectual Property direction in Policy (BBP 3.0 initiatives):

- Remove Barriers to Commercial Technology Utilization
- Increase the Productivity of Corporate IRAD
- Use Modular Open Systems Architecture to Stimulate Innovation

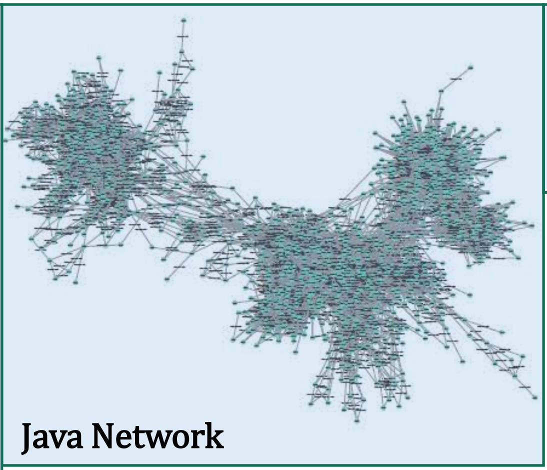
Intent of each is to manage intellectual property and/or avoid traps (lock-in, hold-up, etc.), but no guidance on "how-to"

Software Problem Illustration

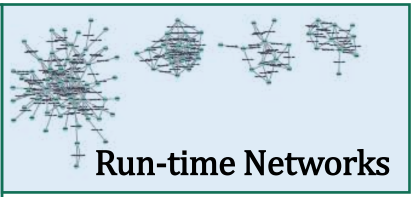
Which pieces of IP are "needed to sustain" the system (flight sim)?



C++ Network

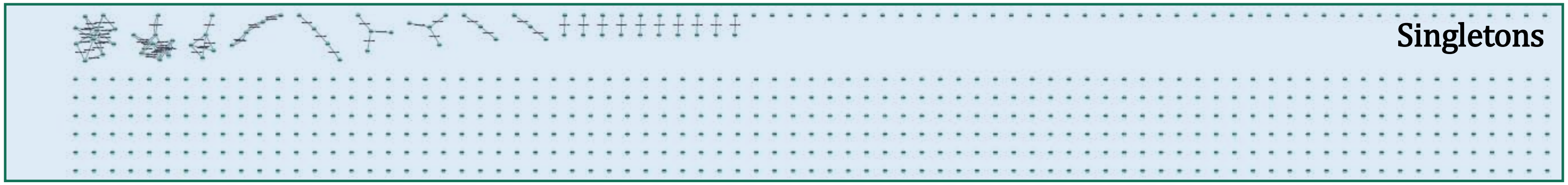


Java Network



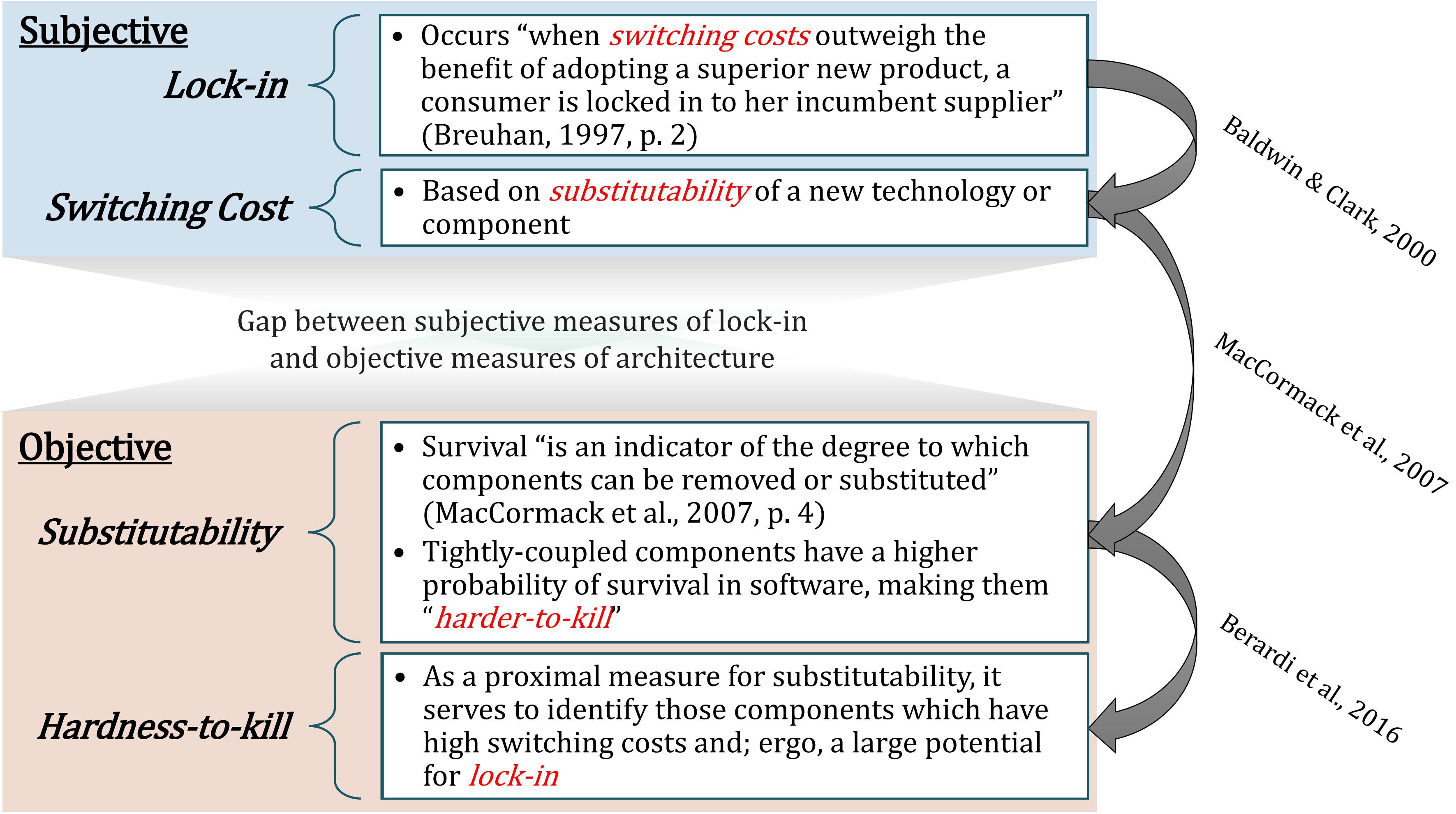
Run-time Networks

Software Architecture Characteristics	
Number of files	6,362
Number of networks	25
Number of cyclic groups	245
Largest cyclic group	665
# of direct dependencies	52,385



Singletons

Theoretical Basis (lock-in)



Subjective

Lock-in

- Occurs “when *switching costs* outweigh the benefit of adopting a superior new product, a consumer is locked in to her incumbent supplier” (Breuhan, 1997, p. 2)

Switching Cost

- Based on *substitutability* of a new technology or component

Gap between subjective measures of lock-in and objective measures of architecture

Objective

Substitutability

- Survival “is an indicator of the degree to which components can be removed or substituted” (MacCormack et al., 2007, p. 4)
- Tightly-coupled components have a higher probability of survival in software, making them “*harder-to-kill*”

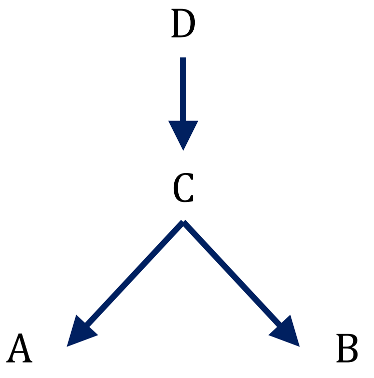
Hardness-to-kill

- As a proximal measure for substitutability, it serves to identify those components which have high switching costs and; ergo, a large potential for *lock-in*

MacCormack et al., 2007 demonstrates files with high Visibility Fan-in and high Visibility Fan-out are statistically significant indicators of hardness-to-kill. However, high VFI is more dominant.

Visualizing Software Architecture

Simple Network* (Direct)



Simple DSM (*M*)

	A	B	C	D
A	■			
B		■		
C	●	●	■	
D			●	■

$$V = \sum M^n , n = 0,1,2,3$$

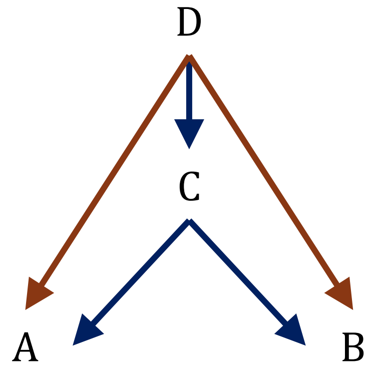
$$M^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M^1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$M^2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$M^3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Simple Network* (Transitive Closure)



Visibility Matrix (Transitive Closure)

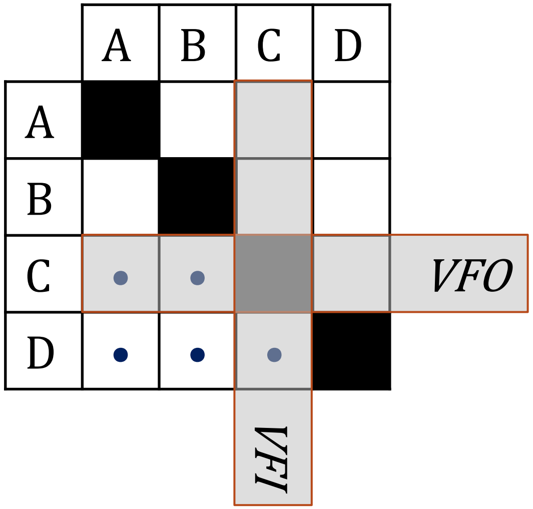
	A	B	C	D
A	■			
B		■		
C	●	●	■	
D	●	●	●	■

*Unit of analysis = source file, dependency type between units of analysis = function call

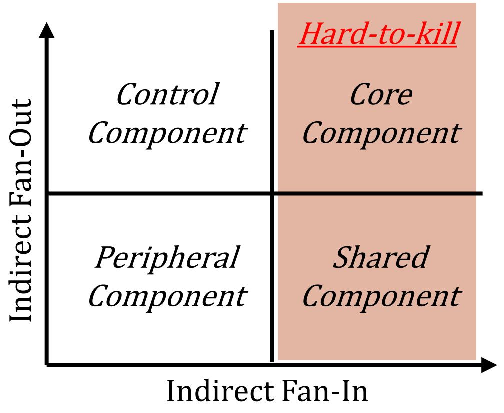
Calculating Metrics / Classifying Files

- *Fan-out Visibility (VFO)* – Sum along rows of visibility matrix and divide by total number of elements:
  - An element with high VFO depends on (or calls functions within) many other files
- *Fan-in Visibility (VFI)* – Sum down columns of visibility matrix, and divide by total number of elements:
  - An element with high VFI is depended upon by many other files (or call functions within it)

Visibility Matrix



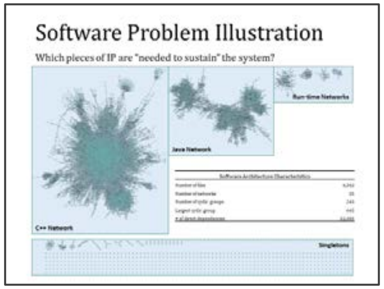
Four Canonical Types of Components



Core Components:	Core files are "seen by" many files and "see" many files.
Shared Components:	Shared files provide functionality to many parts of the system. These files are "seen by" many files, but do not "see" many files.
Peripheral Components:	Peripheral files are neither "seen by" many files nor "see" many files.
Control Components:	These files "see" many other files, but are not "seen by" many files.

Case Study (AF Flight Sim)

Problem

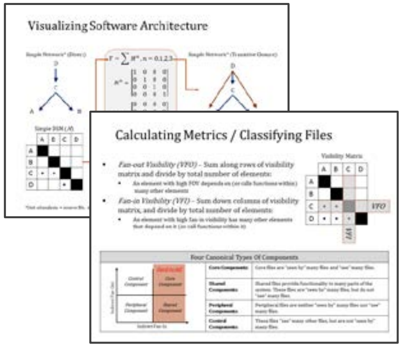


Software Problem Illustration

Which pieces of IP are "needed to sustain" the system?

- Must comply with IP law/policy
- Limited to subjective evaluation
- Limited budget for data rights

Method



Visualizing Software Architecture

Calculating Metrics / Classifying Files

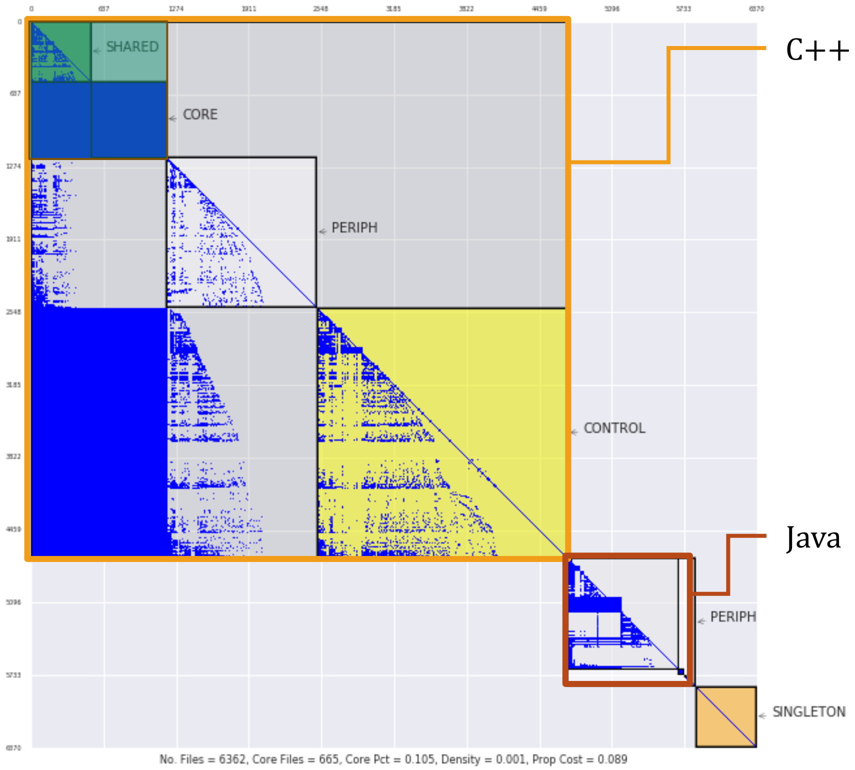
- *Fan-out Visibility (VFO)* – Sum along rows of visibility matrix and divide by total number of elements:
  - An element with high FOV depends on (or calls functions within) many other elements
- *Fan-in Visibility (VFI)* – Sum down columns of visibility matrix, and divide by total number of elements:
  - An element with high fan-in visibility has many other elements that depend on it (or call functions within it)

Four Canonical Types Of Components

Core Components	Core files are "seen by" many files and "see" many files.
Shared Components	Shared files provide functionality to many parts of the system. These files are "seen by" many files, but do not "see" many files.
Peripheral Components	Peripheral files are neither "seen by" many files nor "see" many files.
Control Components	These files "see" many other files, but are not "seen by" many files.

- Objectively measure file-level importance
- Prioritize files based on computed metrics

Solution



No. Files = 6362, Core Files = 665, Core Pct = 0.105, Density = 0.001, Prop Cost = 0.089

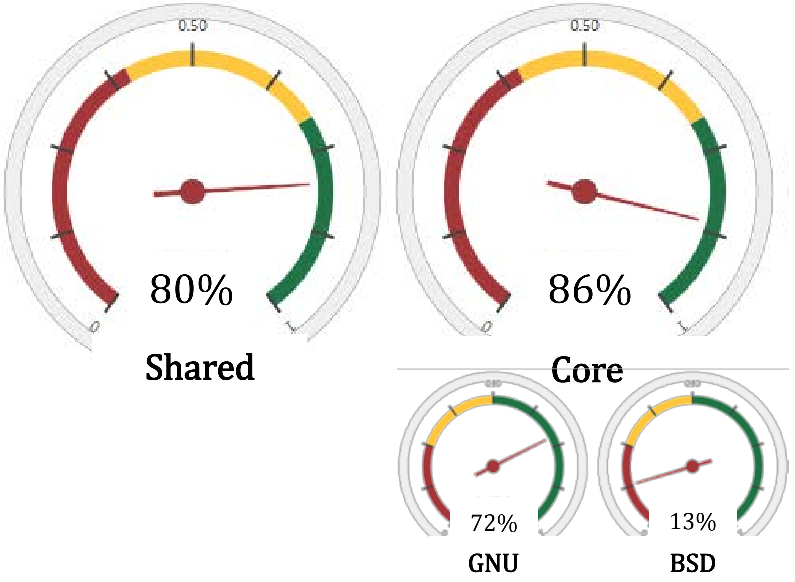
In acquiring the rights to just 18% of files, we argue it increases likelihood of sustained competition because DoD has rights to the subset of files which are hardest to operate the software without

Other Method Applications

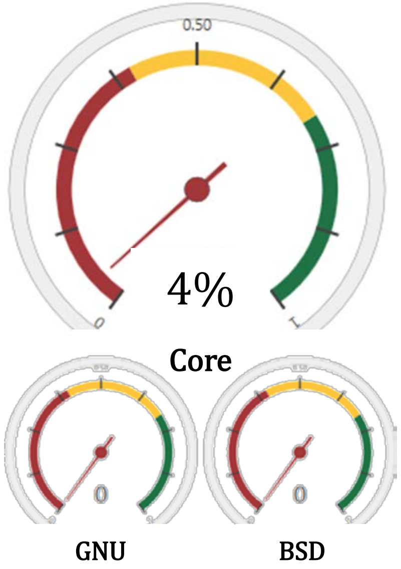
Two additional steps:

1. Scrape copyright information from each source file using RegEx
2. Scraped data is arrayed over visibility matrix

Flight Sim Percent Open



*New Sim Percent Open**



- Metric for assessing "openness"
  - A method to implement BBP *Promote Real Competition*
- Potential uses
  - Source Selection decisions (more open ≈ lower sustainment costs)
  - Used as a KPP: Must not exceed core size of 30% (objective way to regulate software complexity) or Core must contain >50% open source (objectively measured incentive)

Stakeholder Feedback & Way Forward

Feedback from Flight Sim contractor: Results are accurate, "[we] were unable to claim any of code as proprietary nor make business case for sale of the software to USAF given the use of open source code and the full USAF funding since inception." *Flight Sim Contractor PM*

Feedback from AF Senior Leadership: "I only understood 10% of the method, but this area is so vitally important . . . you have my full support" *AF PEO*

Feedback from AF Senior Leadership: "For years I have argued with contractors over the 'rights' to certain pieces of software. Having the information you propose could entirely change the course of the discussion." *AF PEO*

Feedback from Defense Contractor: "I don't like it. This is just another hammer the Government will use to hit us with." *Anonymous*

- Future Work
  - Build inductive theory around *ex-ante* choices to reduce risk of IP lock-in
  - Need additional DoD codebases to further research
    - If interested please email: cberardi@mit.edu

Thank You