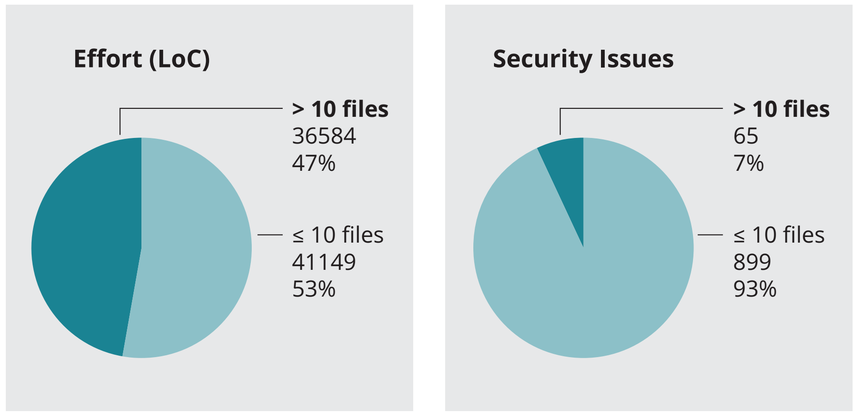
Predicting Security Flaws through Architectural Flaws

Security defects due to implementation and interface dependencies across multiple source code files are difficult and expensive to find and fix. We are evaluating the efficacy of using architectural modular analysis tools to identify security defects and the effect of refactoring on removing security defects.

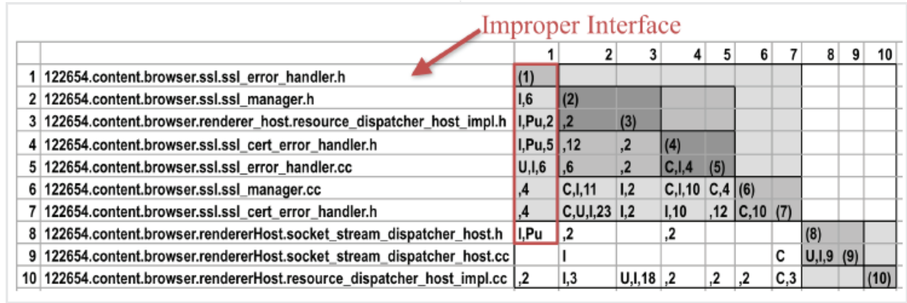
Our project's goal is to use automated architecture analysis to identify, prevent, and mitigate security flaws in code. We are retrospectively analyzing open source software, with revision history and issue lists that include identified security flaws. With this data, we are identifying correlations and building models between the relationship of architectural flaws and security flaws. Future work could use this approach to identify areas of code to refactor for architectural and security improvements.

Statistical Analysis. We evaluated the correlations of the existence of security flaws with the existence of different architectural flaws. We also evaluated the effect of refactoring for reducing security flaws, based on the existence of security flaws before and after major refactoring.

Our analysis has found that some refactoring has significantly reduced security flaws, and that security flaws are commonly present with some architectural flaw types.



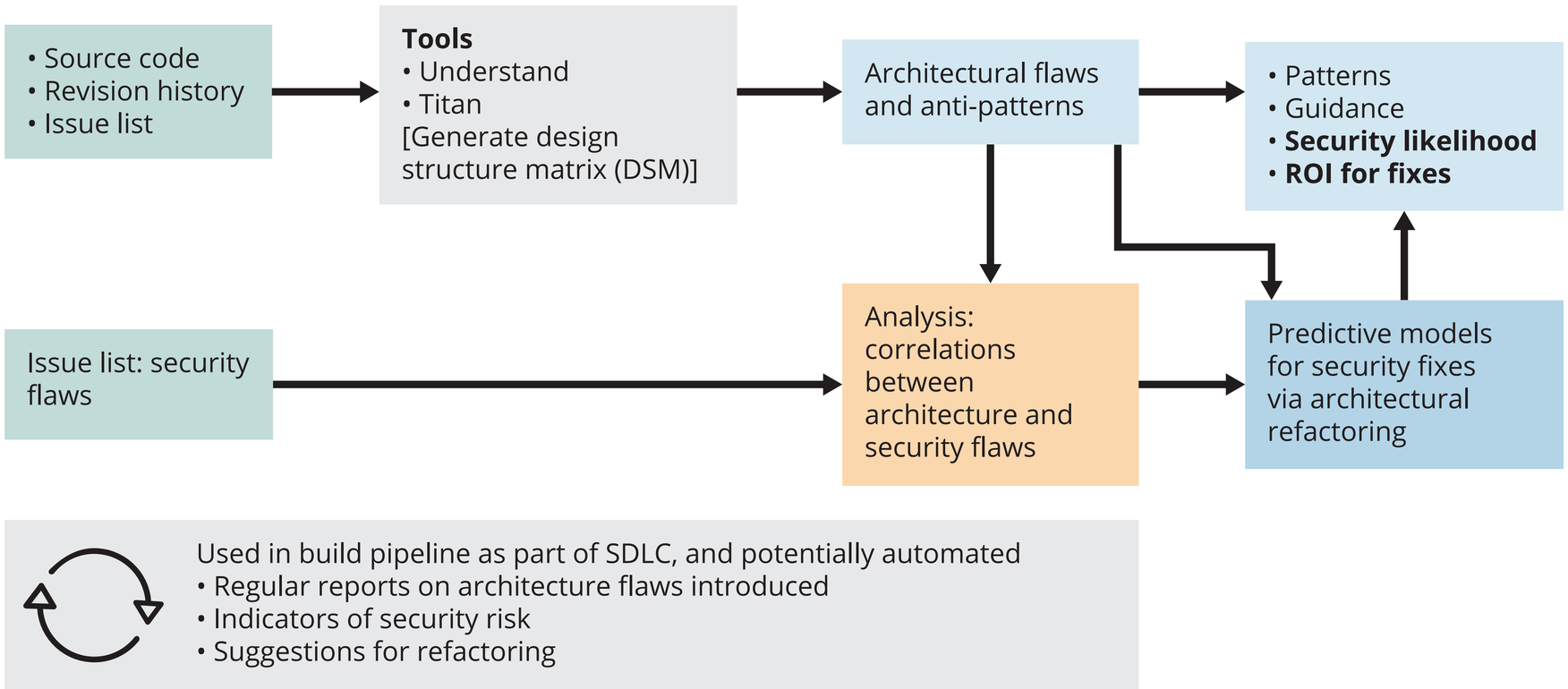
Potential Impact: ~50% of total effort (LoC) to fix security issues came from fixing <10% of the security issues in Chromium.



		1	2	3	4	5	6	7	8	9	10
1	122654.content.browser.ssl.ssl_error_handler.h	(1)									
2	122654.content.browser.ssl.ssl_manager.h	I,6	(2)								
3	122654.content.browser.renderer_host.resource_dispatcher_host_impl.h	I,Pu,2	,2	(3)							
4	122654.content.browser.ssl.ssl_cert_error_handler.h	I,Pu,5	,12	,2	(4)						
5	122654.content.browser.ssl.ssl_error_handler.cc	U,I,6	,6	,2	C,I,4	(5)					
6	122654.content.browser.ssl.ssl_manager.cc	,4	C,I,11	I,2	C,I,10	C,4	(6)				
7	122654.content.browser.ssl.ssl_cert_error_handler.h	,4	C,U,I,23	I,2	I,10	,12	C,10	(7)			
8	122654.content.browser.rendererHost.socket_stream_dispatcher_host.h	I,Pu	,2		,2				(8)		
9	122654.content.browser.rendererHost.socket_stream_dispatcher_host.cc		I					C	U,I,9	(9)	
10	122654.content.browser.rendererHost.resource_dispatcher_host_impl.cc	,2	I,3	U,I,18	,2	,2	,2	C,3			(10)

C = Call; U = Use; I = Include; T = Type; S = Set; O = Override;
Pu = Public Inherit; ,# = # concurrent check-ins

Design Structure Matrix: Presents Relationships among Modules. This example identifies an Improper Interface, as it has files that are related but co-change frequently (unstable), as well as files that co-change frequently but are not related (implicit relationship).



Process for adding statistical analysis of security flaws and architectural flaws to predict yet-to-be identified security flaws, and provide confidence and a security-ROI for refactoring.

Robert Schiela | rschiela@sei.cmu.edu