Traditionally, software security assurance has focused largely on discovering bugs after the fact (through (fuzz) testing, static analysis, or code review), with a bit of developer education added in. Our experience at Google shows that in practice, this doesn’t work all that well, especially for certain classes of vulnerabilities such as those related to complex, whole-system flows of untrusted data. A step towards addressing this unsatisfactory state of affairs is to change focus from chasing down instances of implementation-level defects and vulnerabilities, and instead treat the mere potential that a particular type of defect could exist as a design flaw at the application architecture and frameworks level.

Over the past several years we have developed design patterns that, when applied to application architecture, API and framework design, do indeed result in a drastic reduction if not elimination of the potential for certain types of defects to occur in application code.

This talk will briefly summarize our perception of the limitations of traditional approaches to software security. We will then give examples of secure design patterns we have developed, and discuss how we were able to apply them at scale to frameworks and APIs that form the basis of Google flagship products such as GMail, Docs, Search, G+ and many others.
Facebook employs a defense-in-depth approach to product security; we use a range of preventative
and detection-based approaches to help ensure that our Hack/PHP codebase and its myriad
backend services behave as intended. In this context, 'preventative' might refer to secure-by-default
libraries for doing privacy-aware data fetching. 'Detection' might refer to manual review by a
security engineer, automated static analysis before the code is employed in production, runtime
detection (e.g. Invariant Detector; IEEE S&P 2017), or our bug bounty program.

In this talk, I will discuss a static analyzer that we built to surface potential security and privacy
issues in the Facebook.com codebase. We have developed a bottom-up, inter-procedural, abstract interpreter that focuses on security issues that are difficult to prevent
using the type system (i.e., Hack) or secure libraries and frameworks. We designed the tool based
on guidance from Facebook's security engineering teams. When a new class of vulnerabilities is
discovered, we evaluate whether it is amenable to static analysis. If that is the case, we prototype
the new rule, refine it based on feedback from security engineers, and then evaluate the rule against
the whole codebase. In some cases, we are able to generate a patch automatically. Concurrently,
we run this tool on every code change, thus preventing the reintroduction of this type of issue.

I will also describe some of the advances in static analysis that enable the tool to scale to thousands
of changes per day in a codebase that measures tens of millions of lines of code.
For almost two decades, software security practitioners have successfully defined advanced techniques and tools that can effectively be applied to develop secure software. Today, with tens of millions of developers creating code for all kinds of software-enabled devices, mobile apps and cloud services, we need to expand the conversation to how to scale software security. This talk will challenge developers, organizations and technology buyers to change their culture in order to deliver software security at scale. It will lay out a vision for a Software Culture Code ingrained with security. It will also identify the many stakeholders who are shaping today’s software culture. These stakeholders can have a direct or indirect influence over people and organizations creating or using software and change the way we work, leading to the delivery and deployment of more secure software.