# SPLASH/ISSTA 2026 Trip Report

Oakland, California, United States

고려대학교 소프트웨어 분석 연구실
신지호
2026.10.05 ~ 2026.10.09

# 1. 들어가며

SPLASH(Systems, Programming, Languages, and Applications)는 프로그래밍 언어 및 소프트웨어 시스템 분야의 학술행사로, OOPSLA, Onward!, Splash-E 등으로 구성된다. 올해에는 소프트웨어 테스팅 및 분석 분야의 ISSTA(International Symposium on Software Testing and Analysis)와 공동 개최되었으며, 정적 분석 분야의 SAS(Static Analysis Symposium), REBASE 트랙, 여러 워크숍도 함께 진행되었다.

이번 SPLASH/ISSTA 2026에는 진행 중인 연구를 포스터로 발표하기 위해 참석했다. 학회 기간 동안 여러 키노트와 연구 발표를 들었으며, 그중 특히 흥미로웠거나 기억에 남는 발표들을 정리하고자 한다.

# 2. 인상 깊었던 발표

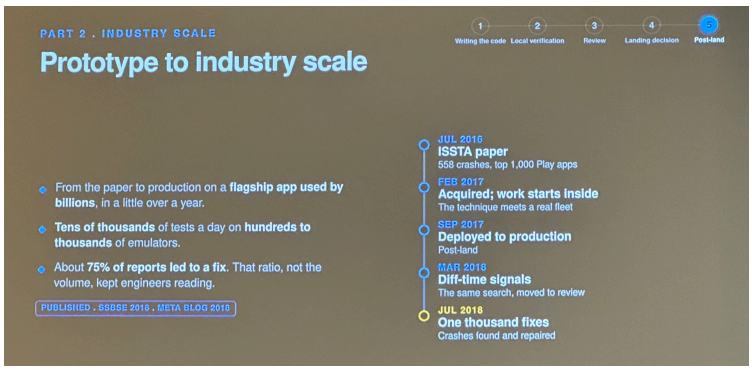
## 2.1. Keynote Talks

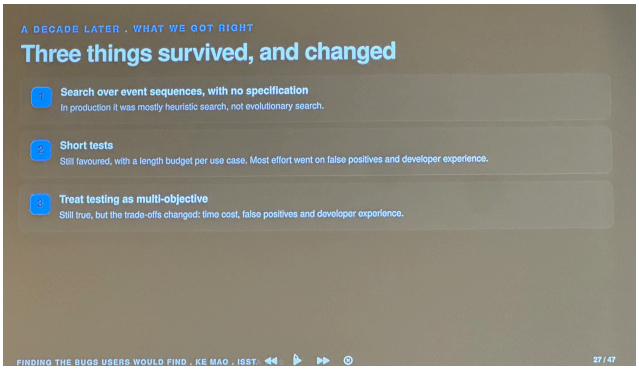
### Beyond Next-Token Prediction: Autonomous Verification and Visual Testing in the Agent-First Era

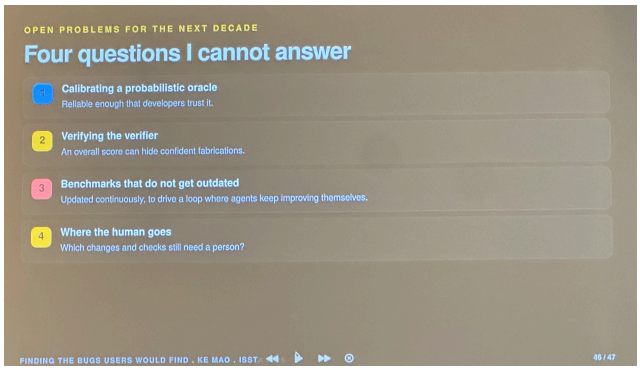
Google Antigravity를 통해 AI 에이전트가 개발 과정에서 수행할 수 있는 여러 작업을 소개했다. 제품 시연이 중심이었지만, 현재 산업계의 에이전트 기술이 어느 정도 수준에 도달했는지 보고 들을 수 있어 흥미로웠다. 기술 설명 외에도 발표자가 전한 몇 가지 이야기가 기억에 남는다. 지신이 아는 많은 프론티어 연구실에서 연구자들이 코드를 들여다보지 않은지 몇 개월이 되었다고 했다. 또한 지적 노동이 대부분 대체될 수 있는 상황에서 자신의 조카들에게는 가장 열정적으로 할 수 있는 일을 찾으라고 조언한다고 했다.

개인적으로는 발표 이후의 질의응답 시간이 더 기억에 남았다. 앞으로 에이전트의 행동을 관찰하고 감독하는 일이 중심이 된다면 인간은 어디에서 지적인 즐거움과 성취감을 느낄 수 있을지에 대한 질문이 있었다. 또한 새롭게 공부를 시작하는 학생들에게 무엇에 집중하고 어떤 역량을 기르도록 가르쳐야 할지, 정교하고 아름다운 코드를 작성하기 위한 연구의 가치가 앞으로 어떻게 달라질지에 관한 질문도 있었다. 명확한 답이 없는 질문들인 만큼 답변도 애매모호하게 이어졌지만, 시니어 연구자들이 에이전트의 발전 속에서 어떤 고민을 하고 있는지 들어볼 수 있어서 좋았다.

### Finding the Bugs Users Would Find: From Sapienz to Autonomous Agents





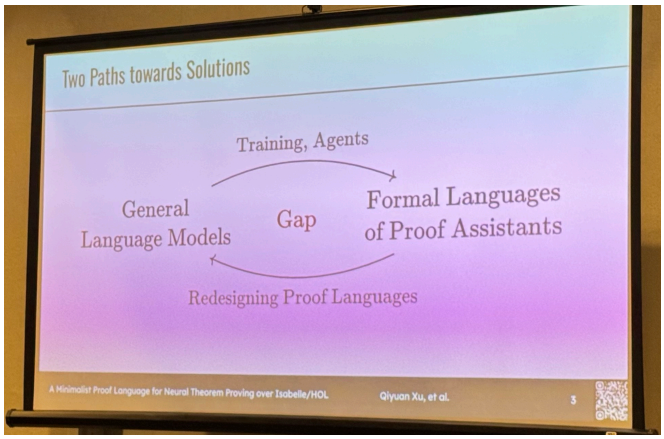




ISSTA 2026 Impact Paper Award를 수상한 2016년 논문 *Sapienz: Multi-objective Automated Testing for Android Applications*가 산업 현장에 적용되어 지난 10년간 어떻게 발전했는지 돌아보는 내용이었다. Sapienz는 Android 애플리케이션의 UI 이벤트 시퀀스를 탐색하면서 코드 커버리지와 결함 발견을 높이고, 실패를 재현하는 시퀀스의 길이는 줄이는 자동 테스팅 기법이다. 특히 발견한 오류를 짧은 이벤트 시퀀스와 재생 가능한 영상으로 제공해, 개발자가 실제로 수정할 수 있는 형태로 전달하는 것을 중요하게 다루었다.

발표는 초기 연구가 산업 규모로 확장되는 과정, 10년이 지난 뒤에도 유지된 아이디어와 달라진 점, 그리고 여전히 해결하지 못한 문제들을 중심으로 구성되었다. 초기의 진화 탐색은 실제 운영 과정에서 휴리스틱 중심의 탐색으로 바뀌었지만, 짧은 테스트를 선호하는 원칙과 여러 원칙을 함께 고려하는 접근은 유지되었다고 한다. 반면 실제 서비스에서는 테스트 실행 비용, 오탐, 개발자 경험의 중요성이 더욱 커졌다. 마지막에는 AI 기반 테스트 오라클의 신뢰성, 검증 결과 자체의 검증, 지속적으로 갱신되는 벤치마크, 그리고 인간의 역할이라는 네 가지 미해결 문제를 제시했다. 한 연구가 논문 발표에 그치지 않고 실제 제품에 적용되어 오랜 기간 발전해왔다는 점이 멋있었다.

## 2.2. Research Talks

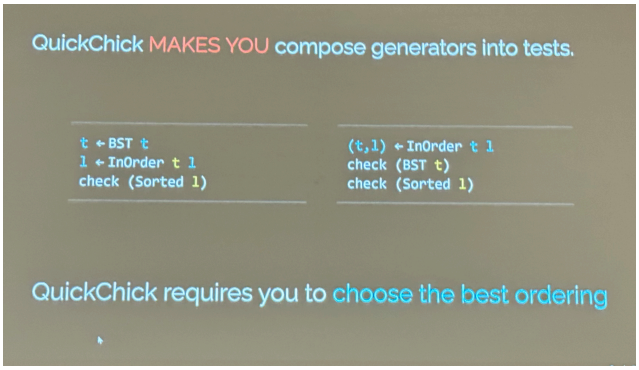
### A Minimalist Proof Language for Neural Theorem Proving over Isabelle/HOL

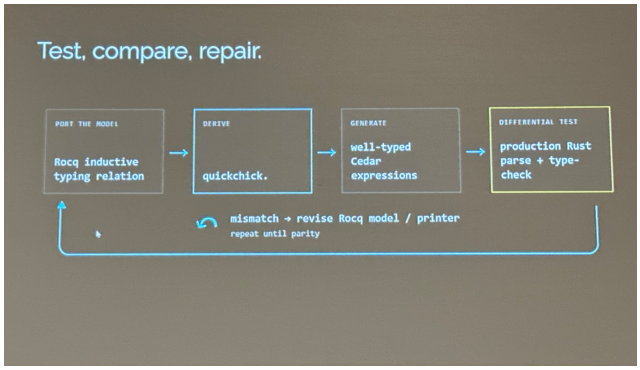


LLM을 이용한 정리 증명에서는 논리적인 추론뿐 아니라 증명 도구의 복잡한 문법과 tactic을 다루는 능력도 요구된다. 세부적인 증명은 자동 증명기에 맡기고 LLM이 고수준 증명 구조에 집중하도록 하는 접근은 기존에도 있었지만, 이 연구에서는 한 단계 더 나아가 LLM이 학습하기 쉽도록 증명 언어 자체를 재설계했다. Minilang은 수학적 증명에서 사용하는 고수준 추론에 대응하는 10개의 핵심 연산으로 구성되며, 각 단계의 세부적인 증명 의무는 Sledgehammer*를 통해 해결한다. 기존 Isabelle 증명 약 34만 개 중 85%를 Minilang으로 변환하여 LLM을 학습시킨 결과 기존 Isar 기반 모델보다 높은 증명 성공률을 달성했다.

발표 말미에는 이 아이디어를 에이전트 방식으로 확장한 후속 연구 *AoA: Theorem Proving Agent over Abstract Syntax Tree of Redesigned Language*도 소개했다. AoA는 증명 스크립트의 텍스트를 편집하는 대신 Minilang의 AST를 직접 조작하는 방식으로, 기존 에이전트보다 토큰 사용량과 도구 호출 비용을 줄였다고 한다. 같은 증명 언어를 모델 학습과 에이전트 시스템에 모두 활용해 좋은 결과를 얻었다는 점이 인상적이었다.

### Testing Theorems, Fully Automatically




정리 증명에 실패했을 때는 증명이 어려운 것인지, 애초에 정리의 명제가 잘못된 것인지 구분하기 어렵다. 따라서 증명을 시도하기 전에 Property-Based Testing(PBT)으로 반례를 찾아보는 것이 유용하다. QuickChick은 이런 테스트를 지원하는 도구지만, 이를 효과적으로 사용하기 위해서는 사용자가 테스트 입력 생성 방법에 대한 여러 결정을 내려야 한다. 특히 귀납적으로 정의된 관계를 만족하는 데이터를 생성할 때, 어떤 값을 먼저 생성하고 어떤 조건을 나중에 검사하는지에 따라 효율이 크게 달라질 수 있다. 예를 들어 타입이 정해진 프로그램을 생성하려면 프로그램을 무작위로 만든 뒤 타입이 맞는지 검사하는 것보다, 주어진 타입을 만족하는 프로그램을 직접 생성하는 편이 효과적이다. 그러나 QuickChick에서는 이러한 순서와 필요한 generator들의 구성을 사용자가 직접 결정해야 하는 경우가 많다.

이 연구에서는 이를 제약 조건이 있는 스케줄링 문제로 바꾸어 자동화했다. 가능한 생성 순서를 탐색하면서 generator가 유효한 데이터를 만들어낼 가능성을 근사하는 density와 변수 간 의존성을 기준으로 효율적인 순서를 선택한다. 불필요한 검사를 피할 수 있는 순서를 우선시하고 비슷한 후보 사이에서는 변수 의존성이 적은 순서를 선호하는 방식이다.

이 발표는 문제 정의부터 해결 방법, 평가까지의 흐름이 아주 깔끔해서 기억에 남는다. 나에게는 한동안 이 연구의 발표자가 세상에서 발표를 가장 잘하는 사람으로 기억될 것 같다.

### Fail Faster: Staging and Fast Randomness for High-Performance PBT

이 연구에서는 PBT 라이브러리가 제공하는 편리한 추상화가 테스트 실행 성능을 떨어뜨릴 수 있다는 문제를 다루었다. PBT에서는 여러 generator combinator를 조합하여 복잡한 테스트 데이터를 생성한다. 그러나 이러한 추상화 계층을 컴파일러가 충분히 최적화하지 못하면서 불필요한 함수 호출이나 closure 및 중간 자료구조의 반복적인 할당으로 인해 추가 비용이 발생할 수 있다. 이를 abstraction overhead 라고 한다.

이러한 비용을 줄이기 위해 staging 기반의 Allegro라는 기법을 제안했다. Staging은 프로그램의 실행을 여러 단계로 나누어, 앞선 단계에서 다음 단계에 실행할 코드를 생성하는 기법이다. 이를 활용해 generator의 조합 구조를 컴파일 시점에 처리하고 실행 시점에는 불필요한 추상화 비용이 발생하지 않도록 특화된 코드를 생성한다. 또한 난수 생성 라이브러리의 구현을 최적화하여 추가적인 성능 개선을 이루었다. 두 방법 모두 기존 generator의 동작을 유지하면서 테스트 데이터 생성 속도를 높임으로써 버그 발견 시간을 단축할 수 있었다.

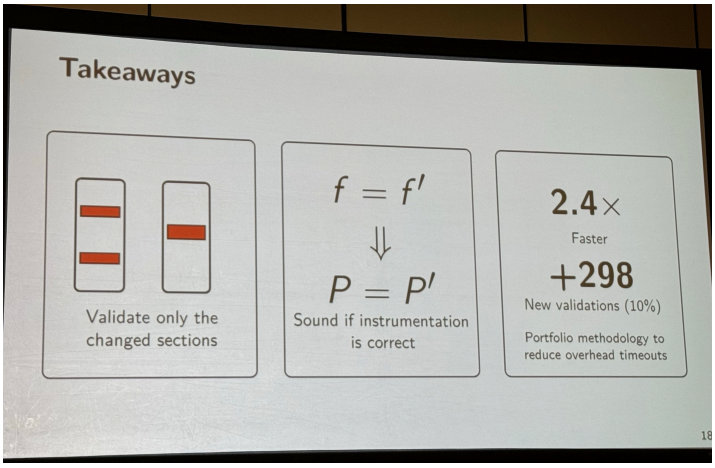
낯선 개념이 많았지만 발표에서 차근차근 설명해 주어서 재밌게 들을 수 있었다. 또 테스트 입력 품질이나 탐색 전략을 바꾸는 게 아니라, 테스트를 생성하는 기반 시스템의 실행 비용을 줄여서 버그를 더 빨리 발견한다는 점이 신선하게 느껴졌다.

### Automated Debugging of Datalog Programs

이번 학기 소프트웨어 테스팅 수업에서 배운 Spectrum-Based Fault Localization(SBFL)을 Datalog 프로그램에 적용한 연구여서 관심 있게 들었다. 일반적인 SBFL은 여러 테스트의 성공/실패 여부와 각 테스트가 실행한 문장들의 정보를 바탕으로 오류가 있을 가능성이 높은 문장을 찾아낸다. 그러나 충분한 커버리지 정보를 수집하기 위해 많은 테스트가 필요하고, 동일한 basic block에 속한 문장들을 항상 함께 실행되므로 서로 구분하기 어렵다는 한계가 있다. 이 연구에서는 Datalog의 특성이 이러한 한계를 완화하는 데 유리하다는 점에 주목했다.

Datalog은 한 번의 실행으로도 많은 fact를 유도하고, 각 fact는 어떤 규칙들을 통해 유도됐는지 나타내는 proof tree를 가진다. 연구에서는 각각의 derived fact를 하나의 테스트 사례로, proof tree에 등장하는 규칙들을 해당 테스트에서 실행된 문장들로 간주하여 SBFL을 적용했다. 이를 통해 각 규칙이 올바른 fact와 잘못된 fact의 유도에 관여했는지 분석하고 오류가 있을 가능성이 높은 규칙을 찾아낸다.

### Spatial and Temporal Decomposition for Faster Translation Validation

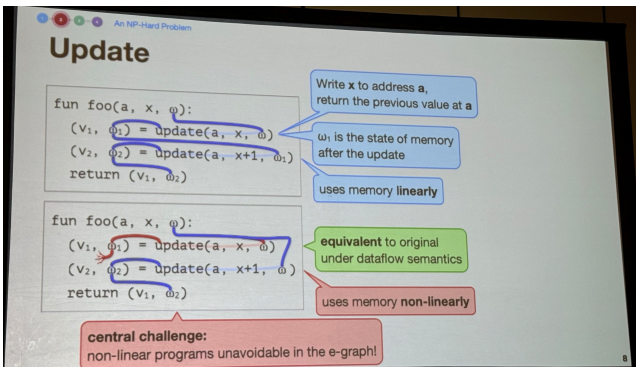


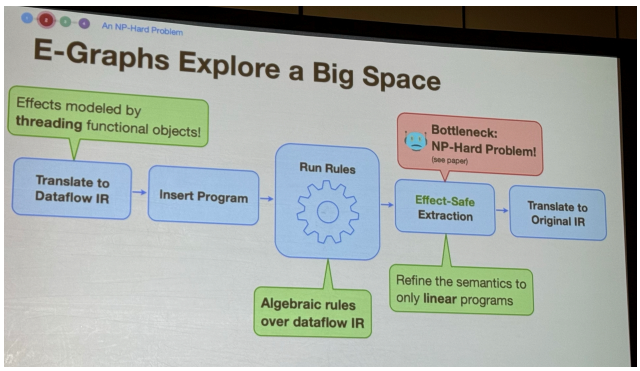

Translation validation은 컴파일러의 최적화 전후 프로그램이 의미적으로 동일한지 검증하는 기법이다. 기존의 Alive2 기반 translation validation는은 최적화 전후의 함수 전체를 대상으로 SMT 제약식을 생성하므로, 최적화와 무관한 코드까지 검증 부담에 포함될 수 있다. 이 연구에서는 이 문제를 해소하기 위해 검증 단위를 공간적·시간적으로 분해해서 비용을 줄이는 DESTIVA를 제안한다.

핵심 기여는 프로그램 전체의 정확성을 보장하면서 검증 범위를 줄이는 방법이다. 공간적 분해를 위해서는 컴파일러에 instrumentation을 추가해 최적화 과정에서 변경된 명령어를 추적하고, 데이터 의존성을 분석하는 program chopping을 통해 검증에 필요한 부분 프로그램을 추출한다. 그리고 추출한 부분 프로그램의 refinement가 전체 프로그램의 refinement를 보장함을 증명했다. 시간적 분해에서는 기존의 end-to-end와 pass-by-pass 검증 사이에서, 각 최적화 패스가 변경한 명령어의 규모를 기준으로 여러 패스를 동적으로 묶어서 검증한다.

두 방법을 결합한 결과 기존 방법들이 타임아웃된 298개 사례를 추가로 검증했으며 검증에 오래 걸리는 사례에서는 pass-by-pass 방식 대비 약 2.4배의 속도 향상을 보였다. 다만 분해 과정 자체에도 비용이 발생하므로 기존 방식과 병행하는 portfolio 접근이 필요하다고 한다.

### Efficient Extraction for Effectful E-graphs



E-graph는 의미가 같은 여러 표현을 함께 관리하여 프로그램 최적화에 활용할 수 있는 자료구조다. 그러나 메모리 접근이나 입출력처럼 실행 순서가 중요한 effect가 포함되면, 단순히 비용이 가장 낮은 표현을 선택할 수 없다. Effect들의 순서도 보존해야 하기 때문이다.

기존에는 이러한 제약을 만족하는 프로그램을 추출하기 위해 integer linear programming(ILP) 기술을 사용했지만 이 과정이 최적화의 병목이 될 수 있었다. 이 연구에서는 유효한 effect-safe extraction을 찾는 문제가 NP-complete임을 보인 뒤 Statewalk DP라는 dynamic programming 기반 알고리즘을 제안한다. 핵심은 문제의 복잡도가 전체 프로그램 크기뿐 아니라 effect 사이의 데이터 흐름 구조를 나타내는 statewalk width에 영향을 받는다는 점이다. 실제 프로그램에서는 이 값이 비교적 작은 경우가 많다는 점을 이용해 기존 ILP 기반 방식보다 훨씬 빠르게 추출할 수 있음을 보였다.

문제의 계산 복잡도를 분석한 뒤 실제 사례에서 활용할 수 있는 구조적 특징을 찾아 효율적인 알고리즘으로 연결했다는 점에서 흥미로웠다.

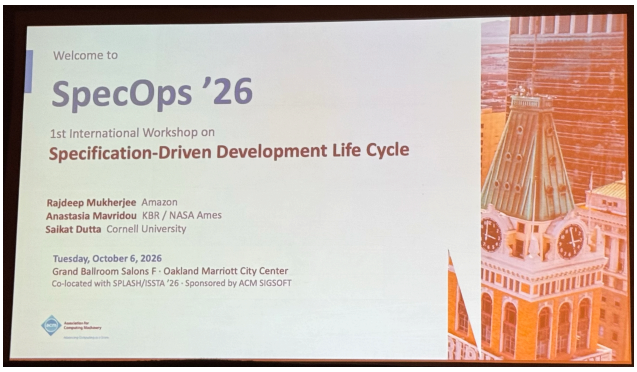
## 3. 포스터 발표 및 SpecOps 워크숍

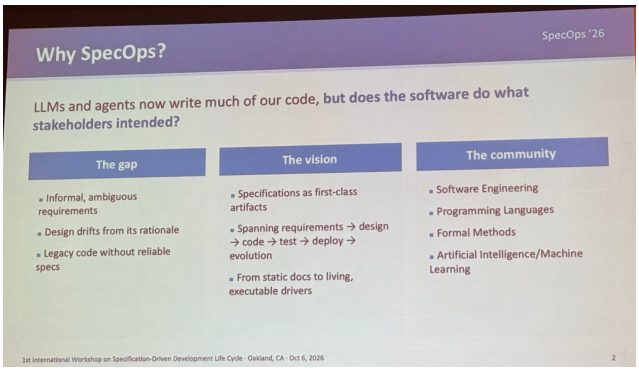
나는 이번 학회에서 진행 중인 연구를 포스터로 발표했다. 포스터 세션을 둘러보니 이미 학회에서 발표된 논문을 소개하는 포스터가 대부분이었고, 진행 중인 연구를 다루는 포스터는 적었다. 여러 참석자들이 찾아와 설명을 듣고 질문을 했는데, 그 중에는 비슷한 분야의 연구를 진행하고 있는 대학원생, 관련 연구를 하는 학생을 지도하는 교수, 자동 정형 검증을 자신의 연구 분야에 적용해 보고 싶어하는 교수 등이 있었다. 모두 격려와 응원의 말을 건네주셔서 감사했다. 한편으로는 이야기를 나누는 과정에서 지금 다루는 연구 주제 외에도 프로그램 검증의 다양한 영역에 대한 이해를 넓힐 필요가 있겠다고 생각했다.



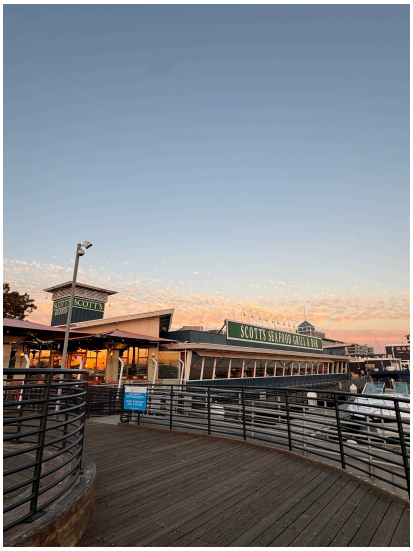


이번 학회에서는 명세 중심 소프트웨어 개발을 주제로 하는 SpecOps(Specification-Driven Development Life Cycle) 워크숍이 처음 개최되었다. AI 에이전트의 발전과 함께 명세의 중요성이 부각되고 있는 상황에서, AI가 생성한 명세의 품질을 평가하거나 명세를 기반으로 테스트를 생성하는 등 산업계에서 명세 중심 개발을 어떤 방식으로 시도하고 있는지 들어볼 수 있는 자리였다.



## 4. 구경








노을 지는 잭 런던 스퀘어. 배가 많았다.



테슬라 로보택시를 이용해봤다.
차가 나에게로 오는 시간이 좀 오래 걸릴 수는 있지만
우버보다 깨끗하고 저렴해서 시간 여유가 있다면 안 탈 이유가 없다.
뒷자석에 앉으면 운전석에 앉아 계신 분의 몸에 핸들이 가려져서
자율주행차에 타고 있다는 사실이 실감 나지 않을 수 있다.

오클랜드 근처에 버클리가 있다.
그래서 UC Berkeley 탐방을 갔다.
캠퍼스가 고대 같다는 의견이 고대 사람들로부터 나왔는데
며칠 후 CS 건물에 다녀오신 분이 보여주신 사진 속
으리으리한 건물을 보고 그곳만큼은 고대가 아니라
백화점이라고 생각하게 되었다.







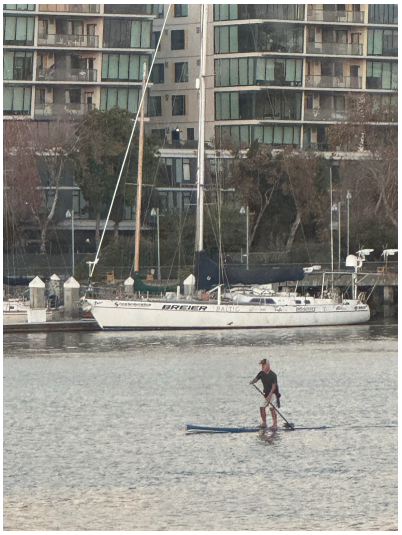

잭 런던 스퀘어에는
아주 타고 싶게 생긴 배가 있는데
오클랜드와 알라메다를 3분 내로
오갈 수 있는 무료 수상 버스이다.

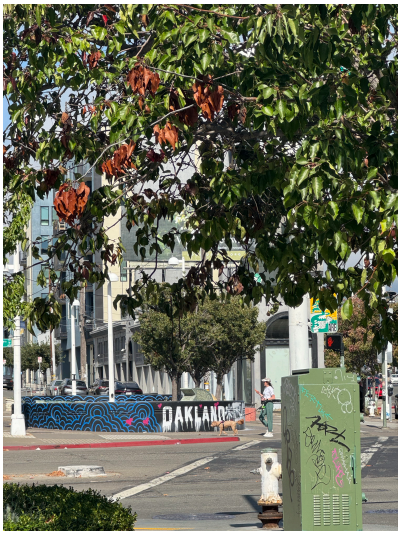
메릿 호수에는 펠리컨이 산다.





연구실 선배들이 요리를 해주셨다.
알리오올리오와 불고기맛 나는 스테이크였다. 맛있었다.








## 5. 마치며

그동안 여러 PL 및 SE 학회에 참석해 봤지만, 개인적으로는 OOPSLA에서의 연구 발표가 가장 재미있는 것 같다. 학회에 참석하면 주 관심 분야의 연구 발표는 당연히 찾아 듣게 되지만, 남는 시간에 잘 모르는 분야의 발표를  별생각 없이 들었다가 흥미로운 연구를 발견하는 순간이 특히 기억에 남는다. 이번 학회에서 그런 즐거운 경험을 여러 차례 할 수 있어서 좋았다.

또 이번에는 학회에서 포스터를 발표하는 새로운 경험도 했다. 방학마다 국내 워크숍에서 포스터를 발표했던 경험 덕분에 큰 어려움 없이 준비하고 마칠 수 있었던 것 같다. 이번에는 교수님과 선배들이 여러 번 맛있는 밥을 사주셔서 학회 기간을 더욱 즐겁게 보낼 수 있었다. 준비 과정에서 도움을 준 연구실 선배, 함께 이번 학회에 참석한 선배들, 그리고 참석 기회를 주신 교수님께 감사하다.