## Recap
- Logistics
- Why program analysis?
- A few first examples

## Today
- Manual program analysis: Code review
- Terminology and important concepts
- Static vs. dynamic analysis
- Paper discussion
  - Static and dynamic analysis: synergy and duality
  - Lessons from Building Static Analysis Tools at Google

## Code review/inspection
Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection
Code review/inspection

Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

Anything that could be improved in this (Java) code?

double foo(double[] d) {
    int n = d.length;
    double s = 0;
    int i = 0;
    while (i<n)
        s = s + d[i];
    i = i + 1;
    double a = s / n;
    return a;
}

Code review/inspection

Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n)
        sum = sum + nums[i];
    i = i + 1;
    double avg = sum / n;
    return avg;
}

Anything that could be improved in this (Java) code?

static OSStatus
SSLVerifySignedServerKeyExchange(...) {
    OSStatus err;
    ...
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    err = sslRawVerify(ctx, ctx->peerPubKey, dataToSign, dataToSignLen, signature, signatureLen);
    if (err) {
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify returned %d
        " , (int)err);
        goto fail;
    }
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}

Anything wrong with that code?
static OSStatus SSLVerifySignedServerKeyExchange(...) {
    OSStatus err;
    ...
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    err = sslRawVerify(ctx, ctx->peerPubKey, dataToSign, dataToSignLen, signature, signatureLen);
    if (err) {
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify returned %d\n", (int)err);
        goto fail;
    }
    fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
    }

What are the key differences?

Static vs. dynamic analysis

Static vs. dynamic analysis: overview

Static analysis
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

* Some static analyses are unsound; dynamic analyses can be sound.

Anything wrong with that code?

Apple's "goto fail" bug: a security vulnerability

Form 4 groups, define the following terms, and give examples related to program analysis:
1. Soundness and completeness
2. Abstract domain
3. Conservative analysis (and precision vs. accuracy)
4. Precision vs. Recall (and FP/FN/TP/TN)

Terminology and important concepts

Static vs. dynamic analysis

Reason about the program without executing it.
Build an abstraction of run-time states.
Reason over abstract domain.
Prove a property of the program.
Sound* but conservative.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis: overview

Static analysis
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

\[ y = x++ \]

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis: overview

Static analysis
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

Dynamic analysis
- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

* Some static analyses are unsound; dynamic analyses can be sound.
**Static analysis: examples**

**Compiler: type checking**

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum / n;
    return avg;
}

**Rule/pattern-based analysis (PMD, Findbugs, etc.)**

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum / n;
    return avg;
}

**Dynamic analysis: examples**

**Software testing**

double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum / n;
    return avg;
}

A test for the avg function:

@Test
public void testAvg() {
    double[] nums = {1.0, 2.0, 3.0};
    double actual = Math.avg(nums);
    double expected = 2.0;
    assertEquals(expected, actual, EPS);
}

**Static vs. dynamic analysis**

**What are the key challenges?**


Static vs. dynamic analysis: challenges

Static analysis: choose good abstractions
- Chosen abstraction determines cost (time and space)
- Chosen abstraction determines accuracy (what information is lost)

Dynamic analysis: choose good representatives (tests)
- Chosen tests determine cost (time and space)
- Chosen tests determine accuracy (what executions are never seen)

Google: Why developers don’t use static analysis?
- Not integrated into the developer’s workflow.
- Reported issues are not actionable.
- Developers do not trust the results (FPs).
- Fixing an issue is too expensive or risky.
- Developers do not understand the reported issues.
- Issues theoretically possible but don’t manifest in practice.

Google: effective false positive
- We consider an issue to be an “effective false positive” if developers did not take positive action after seeing the issue.
- If an analysis incorrectly reports an issue, but developers make the fix anyway to improve code readability or maintainability, that is not an effective false positive.
- If an analysis reports an actual fault, but the developer did not understand the fault and therefore took no action, that is an effective false positive.

Do you agree with this characterization?
Is effective false positive rate an adequate measure?