CSCI 435/535 Software Engineering

Lecture 1 · Course introduction

Oscar Chaparro Fall 2026
· William & Mary

Today

Course logistics — how this semester works

The research study — Danny Otten

What is software engineering, and where does it sit in computing?

Software in the wild — the systems we actually build

Why this course is built the way it is

Instructor and TA

Oscar Chaparro

Instructor — you can call me just Oscar

oscarch@wm.edu

Office: ISC4 2325

Office hours: Tue & Thu, 10:45 AM–12:20 PM

ojcchar.github.io

Mustahid Hasan

Teaching assistant

mhasan02@wm.edu

Office hours: Mon & Wed, 11:00 AM–12:00 PM

Office: ISC4 2362

No appointment needed

The SEA Lab

Software Evolution and Analysis Lab

The **SEA** Lab builds techniques, tools, and methodologies that help developers understand, design, build, and maintain high-quality systems.

Undergraduates are welcome — ojcchar.github.io/lab

Oscar Chaparro

Antu Saha

Nathan Wintersgill

Nadeeshan De Silva

Mehedi Sun

Mustahid Hasan

What we work on

Goal: assist developers to better design, build, and evolve high-quality software.

Bug reporting and triage

Better reports, duplicate detection, reproduction steps

Bug localization and repair

Finding the faulty code, and fixing it automatically

Program comprehension

How developers read and understand unfamiliar code

Testing and analysis

Generating tests, mobile app analysis, quality assurance

AI and ML for software

Models and agents applied to development tasks

Software artifacts

Mining bug reports, code, and online discussions

Course logistics

The course at a glance

LECTURES

Tuesdays & Thursdays
12:30–1:50 PM

LOCATION

Integrated Science Center (ISC) 1353 · in person

PREREQUISITES

CSCI 301 Software Development · CSCI 312 Programming Languages

FORMAT

Lectures, tech showcases, a semester-long team project, and project presentations

READINGS

No required textbook. Chapters, papers, and docs posted per class

CROSS-LISTED

CSCI 435 undergraduate · CSCI 535 graduate

What you'll be able to do

- Explain the concepts, processes, and tools of modern software engineering
- Elicit, analyze, and specify requirements for a realistic system
- Design and communicate software architectures, with design rationale
- Implement and test systems using modern frameworks and CI workflows
- Collaborate in a team using agile methods, issue tracking, and version control
- Integrate agentic AI assistants responsibly, transparently, and on the record

The project

Six teams, each contributing to one real open-source project all semester.

Sprint 0

Team setup, learning the system, one meaningful change

Sprint 1

Plan the backlog, build, release

Sprint 2

Same pattern, larger scope

Sprint 3

Same pattern, larger scope

Sprint 4

Final release, demo, report, and presentation

Every sprint ends with a release: issues, pull requests (PRs), code, tests, a live demo, AI logs, a report, and a reflection survey.

Open source projects

Actual Budget

Personal finance

Excalidraw

Collaborative whiteboarding

Gitea

Self-hosted Git service

Medusa

Headless e-commerce

Chatwoot

Customer engagement

Discourse

Forum and community

Cal.diy

Scheduling and booking

Outline

Team wiki and knowledge base

Teams submit their top preferences; we match as closely as we can. Details on the Projects page.

Presentations and showcases

Lightning talks

Two rounds. Short check-ins on your project's status, decisions, and plans.

Sep 22 and Oct 20 · 3% each

New-tech showcases

Two rounds of short, walkable demo sessions on a modern SE tool or technology.

Oct 6 and Nov 5 · 5% each

Final presentation

Each team presents its finished project in full, with a live demo.

Dec 1 and Dec 3 · 9%

Grading

Component	Weight
Final exam (cumulative)	15%
Quizzes / participation	5%
New-tech showcases (2 rounds)	10%
Lightning talks (2 rounds)	6%
Final project presentation	9%
Sprint 0 — setup and system learning	9%
Sprint 1 — development and release	10%
Sprint 2 — development and release	12%
Sprint 3 — development and release	12%
Sprint 4 — development and release	12%

55%

of the grade is sprint work on your team's project

No curves on any component or on the total.

Bonus up to 3% — for a pull request merged upstream, an exceptional showcase, or notable contribution to the project community.

Full scale and policies in the Syllabus. A ≥ 95, A− ≥ 90, B+ ≥ 87, B ≥ 83.

Policies you should know

Quizzes

Short, graded, in class. Complete at least 70% across the semester for full credit on participation.

Late work

A 2-hour grace period. After that, 10% off for every day late.

Attendance

Not directly controlled, but students who miss class have had more trouble with the project and the exam.

Exam

One cumulative final, Dec 9, 9:00 AM–12:00 PM. Closed book, notes, and electronics.

CSCI 535: what's different

Same format, same schedule, same grading weights. However, **the bar within those weights is higher**.

- Each sprint sets a task minimum for graduate students at least 10% above the undergraduate one
- Presentations and showcases must show greater technical depth
- On a team with a graduate student, that student normally serves as team lead

Key dates

Sep 4	**Last day to add/drop**
Sep 22	Sprint 0 lightning talks
Oct 6	New-tech showcase, round 1
Oct 8–11	**Fall Break — no class**
Oct 20	Sprints 1–2 lightning talks
Oct 26	**Last day to withdraw**

Nov 3	**Election Day — no class**
Nov 5	New-tech showcase, round 2
Nov 17 & 19	In-class project working sessions
Nov 25–29	**Thanksgiving Break — no class**
Dec 1 & 3	Final project presentations
Dec 9	**Final exam, 9:00 AM–12:00 PM**
Dec 15	Project due

Tools and communication

csci-435-se.github.io	Course site — schedule, syllabus, slides, readings
github.com/CSCI-435-SE	GitHub organization — all team repositories
csci-435-se.zulipchat.com	Course Q&A, announcements, team discussion — use channels, not DMs
blackboard.wm.edu	Grades and official announcements

Academic integrity and AI logs

Agentic AI tools are a required part of the workflow, not just permitted.

Unrestricted for development: coding, debugging, testing, documentation, design.

The team owns the project. Every member is expected to understand how it works.

AI logs

Prompts, responses, the tool used, and metadata — pushed to your team repository, one folder per log.

Reference each log from its issue, so the work is traceable.

Verifying AI output is your responsibility. Its errors are yours.

Prompt → copy → paste, with no review or understanding, violates the Honor Code.

How to do well in this course

Contribute regularly and visibly — commits, reviews, issues, all semester, not at deadlines

Read what's posted before each class; the quizzes assume you did

Review and understand (almost) every line of AI-generated code your team ships

Bring what you're hitting on your project into class discussion

Reach out early when something blocks you — your team, Mustahid, or me

Support and well-being

If something is making coursework hard, tell me early.

The college also has people whose job this is:

Student Accessibility Services

sas@wm.edu · (757) 221-2512

Academic Wellbeing

wm.edu/academicwellbeing — tutoring and coaching

Counseling Center

(757) 221-3620 — free, confidential

The research study

Evaluating the Impact of Generative AI Tools on Software Engineering Education

PRESENTED BY

Daniel Otten

What is software engineering?

You've all written software. What makes it engineering?

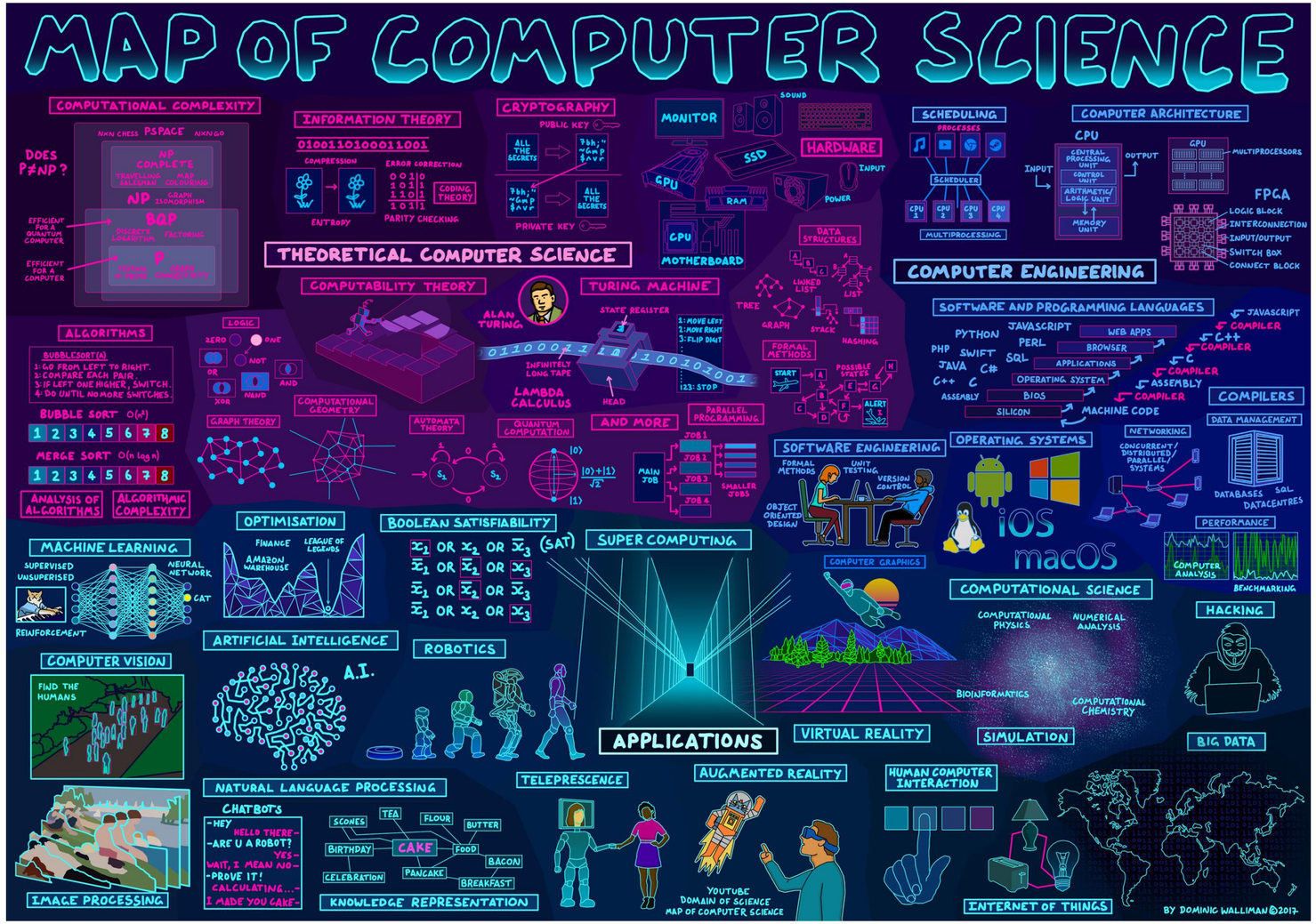
Software engineering within computing

Theory of computation	Algorithms and data structures	Programming languages	Systems and architecture	**Software engineering**	Human–computer interaction	Data, ML, and AI
What is computable at all	How efficiently	How we express it · CSCI 312	What it runs on	Building it, and keeping it working · CSCI 435/535	Who uses it	What it learns from

more theoretical — more applied

Software engineering is not a subfield of any one of these. It sits at the applied end and borrows from all of them.

SE builds on all of computing



“Map of Computer Science” — Dominic Walliman, Domain of Science · used with attribution

What is software?

Computer programs and their documentation and artifacts.

It includes issues, conversations, meetings, diagrams, sketches, requirements documents, specifications, system data, and other recorded knowledge about the system.

Software is not just code.

Not software: the hardware it runs on, and the people and organization around it.

Knowledge that is never recorded matters too, but there is nothing tangible to hand to anyone.

Two different questions

COMPUTER SCIENCE ASKS

Can this be computed, and how efficiently?

One program, one author, one correct answer
Requirements are given in the problem statement
Success is proof, or a passing test suite
The program is finished when it runs

SOFTWARE ENGINEERING ASKS

How can we build it, ship it, and keep it working for ten years?

Many authors, over years, most of whom have left
Requirements are unclear, contested, and changing
Success is measured against cost, time, and quality
Shipping is the beginning of the expensive part

Three definitions for SE

> “The systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software.”
>
> U.S. Bureau of Labor Statistics

> “The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.”
>
> IEEE Standard

> “An engineering discipline that is concerned with all aspects of software production.”
>
> Ian Sommerville

A WORKING DEFINITION

Practices, techniques, systems, tools, and artifacts to design, produce, and maintain high-quality, cost-effective software systems — given the resources actually available.

Money, people, time, knowledge, and tools.

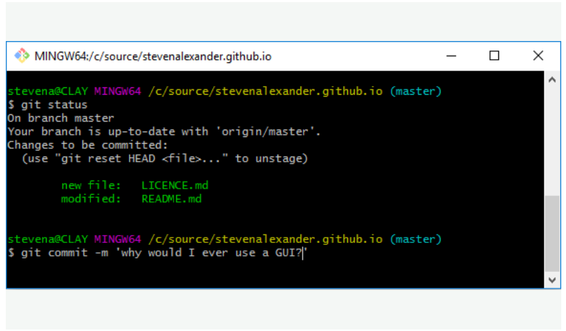
The software engineering spectrum

Understanding unfamiliar codebases	Requirements analysis	Planning development tasks	Implementing code changes
Software design	Software architecture	Data modeling	Automated testing
Code quality analysis	Debugging	Refactoring	Collaborative development workflows
Issue report management	Build setup	Deployment	Communicating technical information

Software in the wild

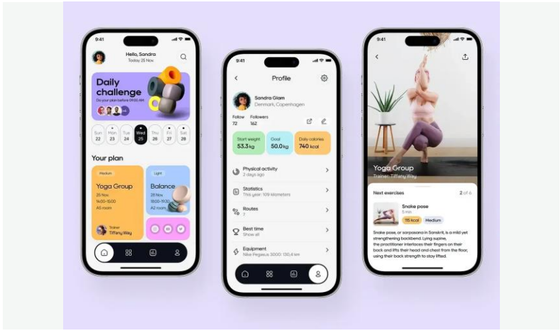
The systems we actually build, and why each one is hard in its own way

Many kinds of software systems



Terminal tools

No UI to hide behind

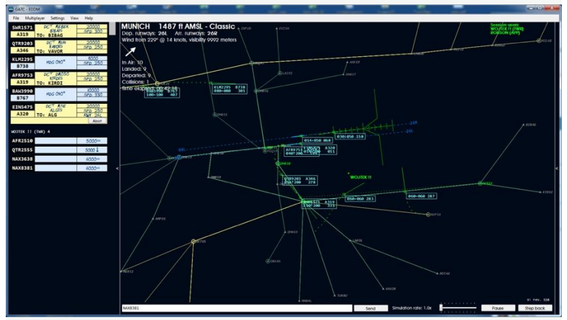


Mobile apps

Battery, network, and a store review in the way

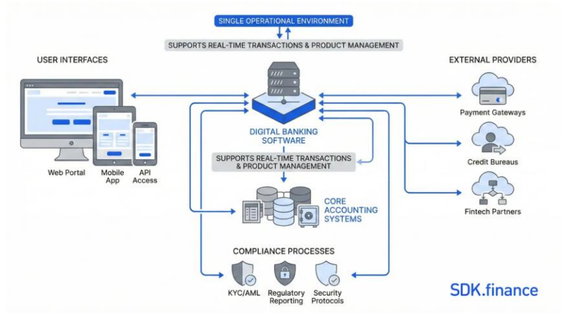
Embedded

Fixed memory, no update path, ships inside hardware



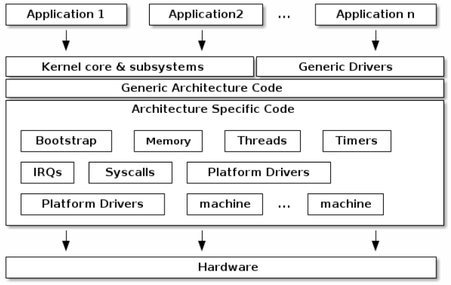
Mission-critical

Failure costs money, or lives



Distributed systems

Parts fail independently; the network is not reliable



Operating systems

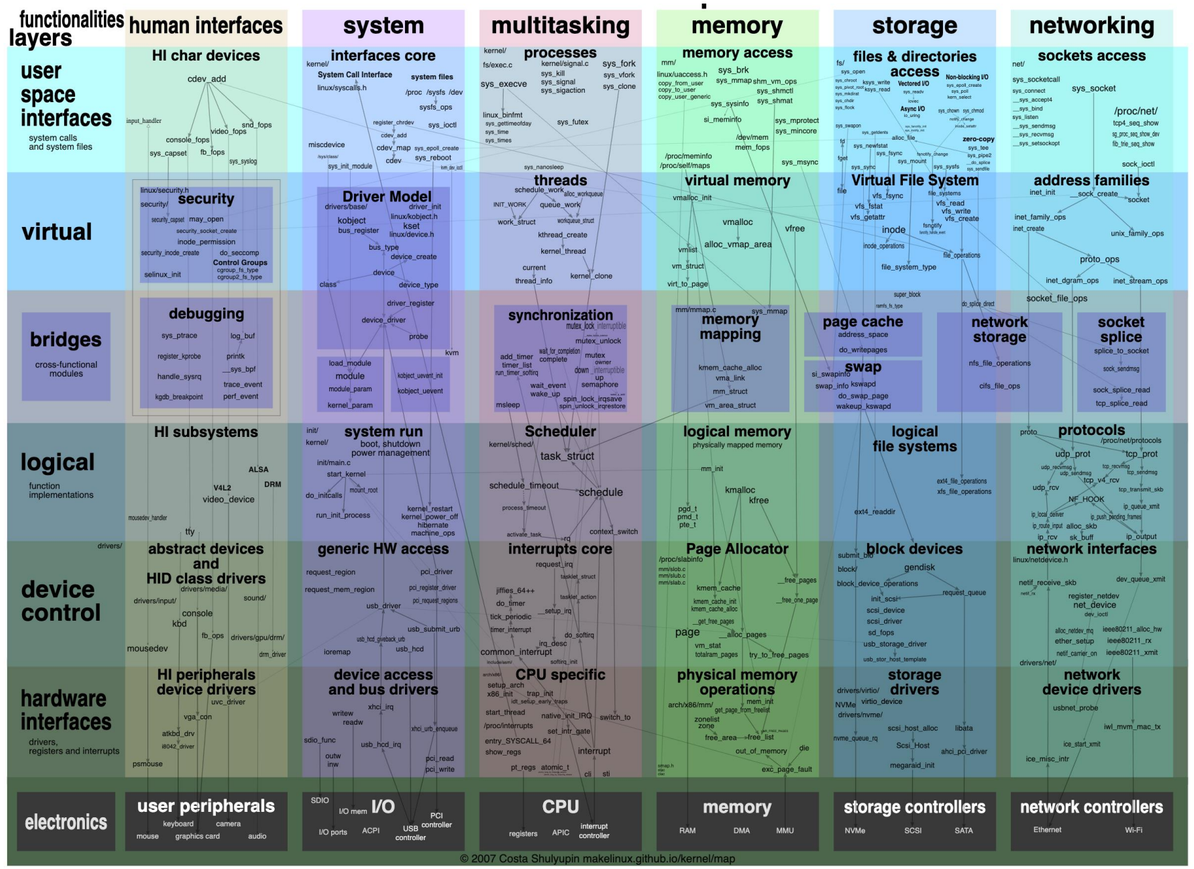
Decades old, thousands of authors, cannot break anything

Libraries and frameworks

Your users are other programmers; the API is forever

Different stacks, different constraints — the same engineering principles apply to all of them.

Scale: the Linux kernel



40M+

lines of source, across about 80,000 files

Over a thousand developers contribute to a typical release.

A new release lands every nine or ten weeks.

No single person has read it. It ships anyway.

When software must not fail

Therac-25, 1985–1987

A race condition let a radiation therapy machine deliver massive overdoses. Six known accidents, several deaths.

Ariane 5 Flight 501, 1996

Reused inertial-reference code converted a 64-bit float to a 16-bit integer. Overflow, then self-destruct, 37 seconds after launch.

We don’t start from scratch

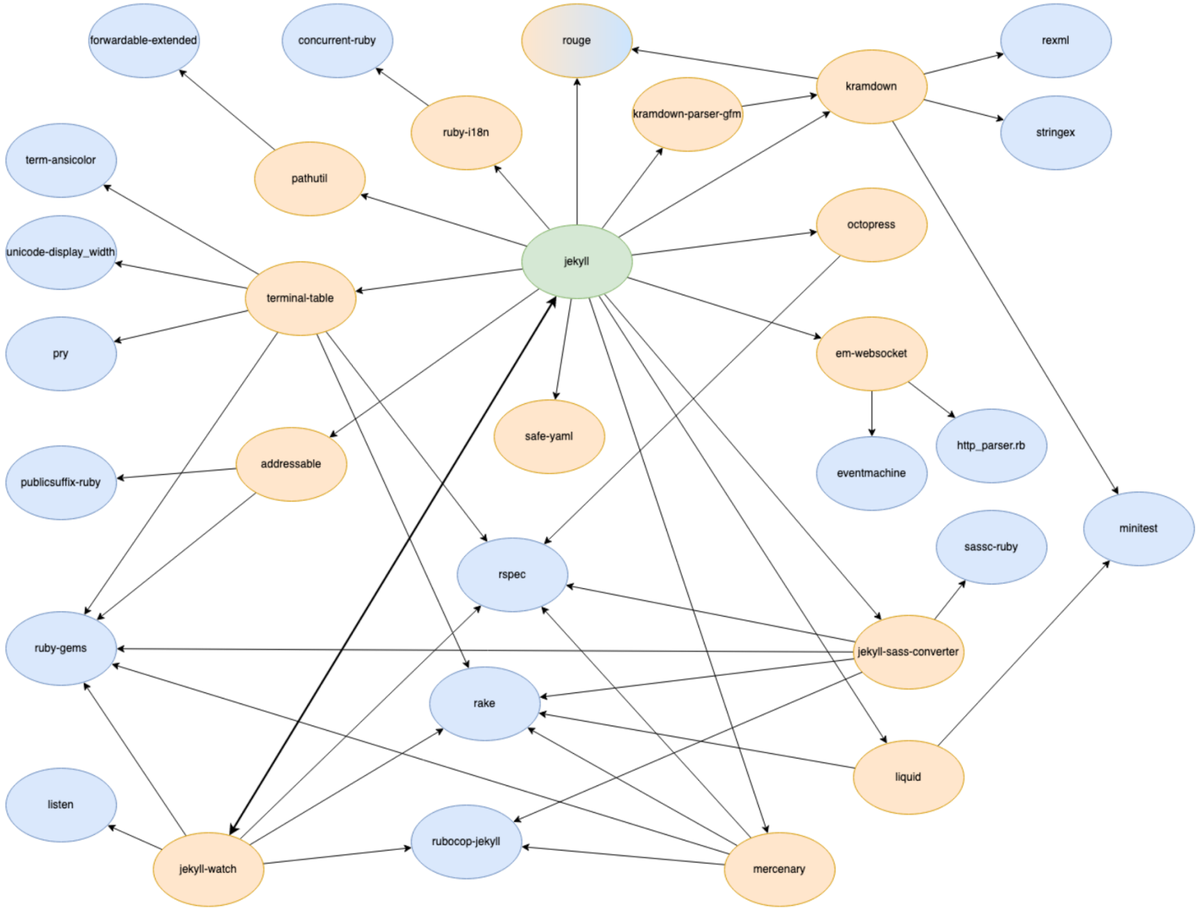
3M+

packages published on npm alone

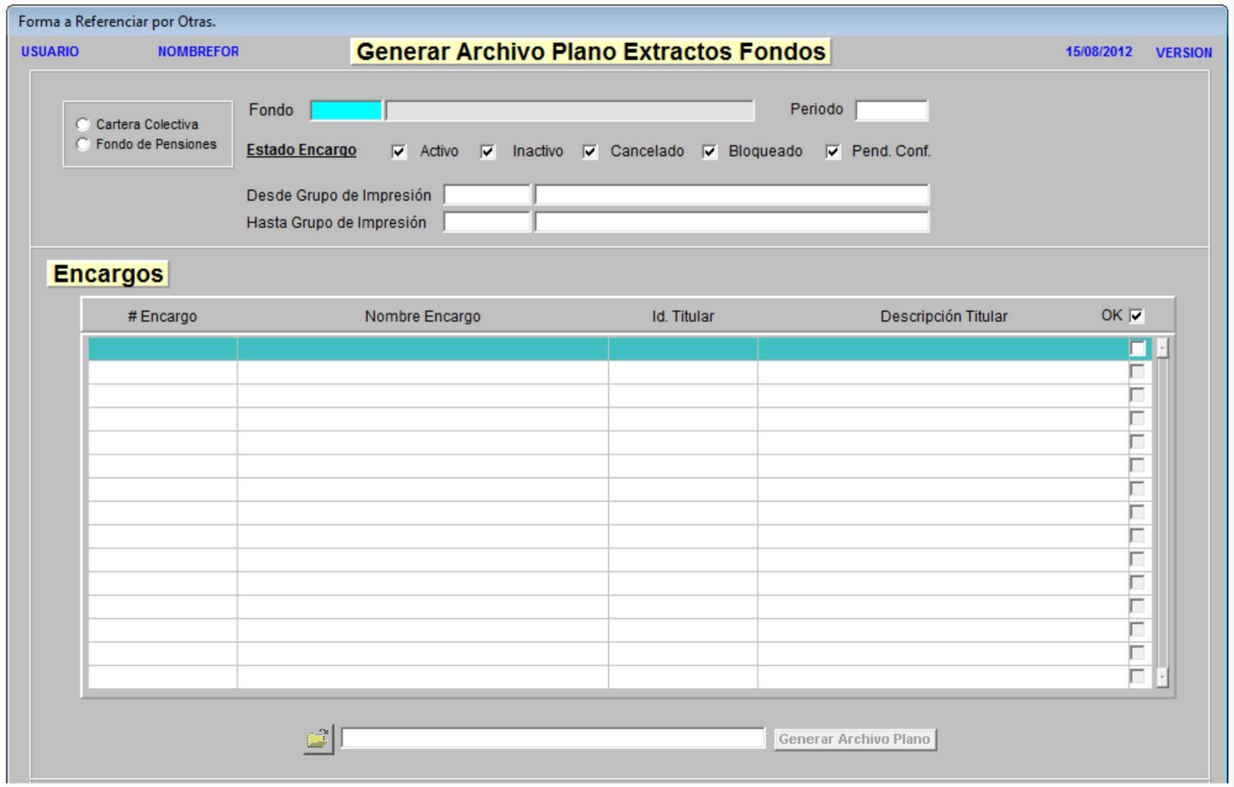
A modern application is mostly code its team did not write. Open source is what made that normal.

You will spend far more of your career reusing, extending, and maintaining existing systems than starting new ones.

Every dependency is also a decision: a maintainer you trust, a licence you accept, a vulnerability you inherit.



A system I maintained



SIFI — a financial information system used by multiple banks

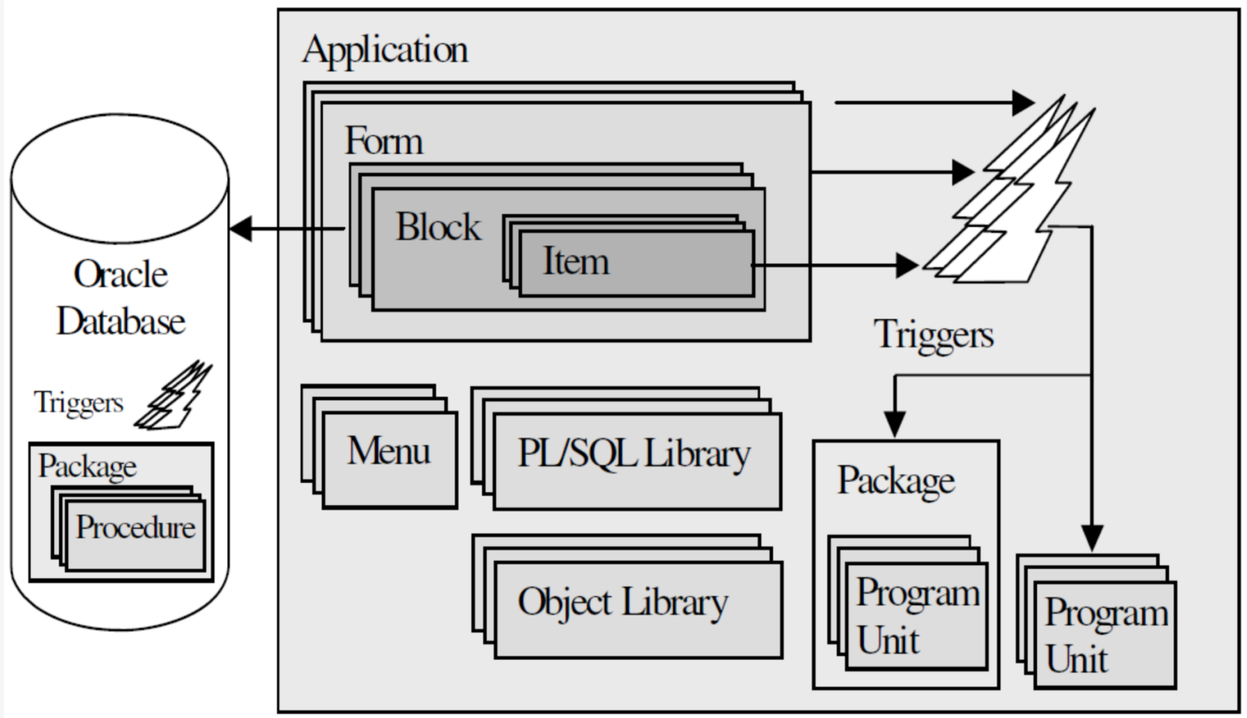
My first job, 2009. The system was already 15 years old.

Oracle Forms 6i and PL/SQL — obsolete when I arrived.

Tightly coupled, undocumented, no traceability between modules.

Nobody on the team had built it. All of us maintained it.

Oracle Forms' architecture

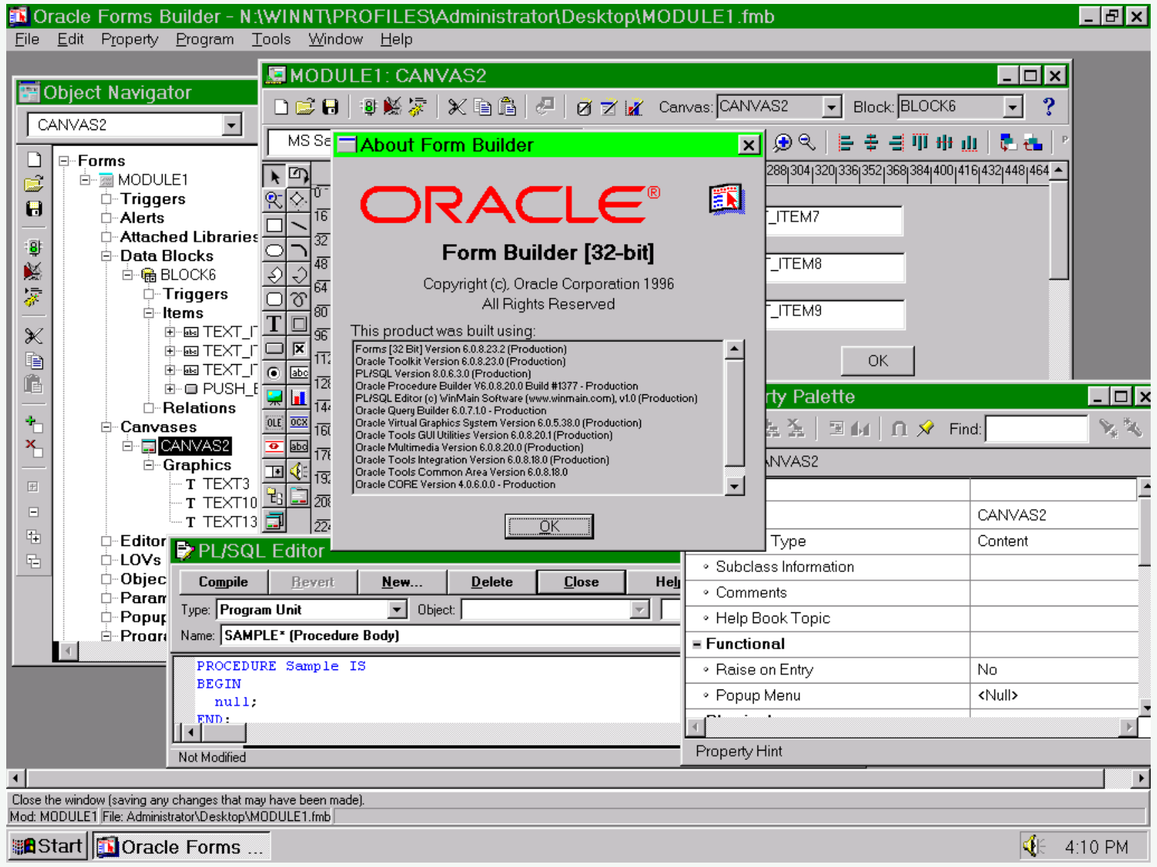


Two tiers: the form runs on the client and talks directly to the database.
A form contains blocks; blocks contain items.
Behavior is attached as triggers at every level — form, block, item.
Shared code lives in PL/SQL and object libraries.
More logic again lives in database packages and database triggers.

Business logic sits in the UI triggers and in the database, with no layer in between.

Why it was hard to work with

- Implemented with an obsolete technology
- Components were extremely tightly coupled
- Code and architecture decay for over 20 years
- Naming code conventions were not easy to understand
- Very limited technical and business documentation
- No traceability information between modules and artifacts
- Knowledge remained in people's heads



Inside SIFI

```
10276      if ( v_Hay_DiAp ) then                        -- Soli_2036 - 17 noviembre 2003
10277        --
10278        -- Actualiza SF_TDIAP, con Numero de Comprobante Generado.
10279        --
10280        update sf_tdiap
10281           set diap_te_NroCom  = p_NroCom,
10282               diap_te_FecMov  = FecMov
10283         where diap_Movi      = i.mvtm_Movi
10284           and diap_te_Rengln = i.mvtm_Rengln;
10285        --
10286      end if;
10287      --
10288      -- Ini.Soli_7184
10289      --
10290      if ( i.mvtm_fd_Oper is not null  and  i.mvtm_Cias_Dest is not null  and  i.mvtm_Orpa is not null ) then
10291        --
10292        -- Liberacion de Reserva por Instruccion de Pago
10293        --
10294        --FD_QInstr_Pago.Libera_Reserva( i.mvtm_Cias_Dest, i.mvtm_fd_Oper, v_InPg );
10295        -- Soli 7540, se modifica el valor por el cual se realiza la liberacion de recurso
10296        if (i.mvtm_inpg is null) then
10297          --
10298          open c_inpg( i.mvtm_Cias_Dest, i.mvtm_VigOrPa, i.mvtm_OrPa, i.mvtm_PgPr );
10299          fetch c_inpg into v_InPg, v_bpop;
10300          if ( c_inpg%NotFound ) then
10301            v_InPg := null;
10302          end if;
10303          close c_inpg;
10304          --
10305        else
10306          --
10307          v_InPg := i.mvtm_inpg;
10308          --
10309        end if;
10310        --
10311        if ( i.mvtm_sf_Plan is not null and i.mvtm_sf_Fondo is not null ) then
10312          --
10313          -- Toma el porcentaje parametrizado para el Plan/Encargo.
10314          --
10315          open c_plan_CE( i.mvtm_sf_Fondo, i.mvtm_sf_Plan );
10316            fetch c_plan_CE into r_pl_CE;
10317          close c_plan_CE;
10318          --
10319          if ( r_pl_CE.plan_RConEsp = 'S' ) then
10320            Valor_Conesp  :=  i.mvtm_valor * nvl( r_pl_CE.plan_porc_RConEsp, 0) / 100;
10321          else
10322            Valor_Conesp  := 0;
10323          end if;
10324          --
10325          -- Ini.Soli_7929
10326          --
10327        else
10328          --
10329          Valor_conesp := te_fvalor_ce(i.mvtm_tpmv, i.mvtm_enti, abs(i.mvtm_valor));
10330          --
```

10,276

the line number, in a single file

Commented in Spanish, dated 2003, referencing a change request nobody could still find.

Systems that grow without software engineering discipline end up here.

Facts about software

Society depends on software more every year	…and it is still frequently unreliable and of poor quality.
Software is treated as an engineering product	…but it is more complex than almost anything else humans build.
Maintenance is treated as low-status work	…but 75–95% of what software costs is spent there.
Software changes for business and technical reasons	…and a system that has stopped changing is effectively dead.
AI is changing how software gets built — a new paradigm	…and we don’t yet know the best ways to use it.

Essential properties of software

Václav Rajlich, *Software Engineering: The Current Practice*

Complexity

More complex than one mind can hold. We manage it by decomposition, hierarchy, and abstraction.

Changeability

Easy to change — a few keystrokes, no walls to tear down. Changing it *correctly* is the hard part.

Discontinuity

A small change in the input can produce a huge change in the output.

Conformity

It must conform to human institutions it does not control.

Invisibility

No inherent geometry — there is nothing to look at.

Why this course is built this way

You engineer an existing codebase

Toy assignments teach syntax. They do not teach what it is like to work inside someone else's large, evolving system.

You work in sprints

Software is never finished, so the course repeats the same loop five times: plan, build, test, review, release, reflect.

You use AI, and you account for it

Agentic tools are professional practice now. AI logs keep using them compatible with understanding your own system.

Next class

WE'LL COVER

A brief history of software engineering
Software processes: waterfall, iterative, agile
Incremental change and the staged model

BEFORE TUESDAY

Read the Syllabus and the Schedule
Look through the eight projects and form a preference
Answer the introductory survey