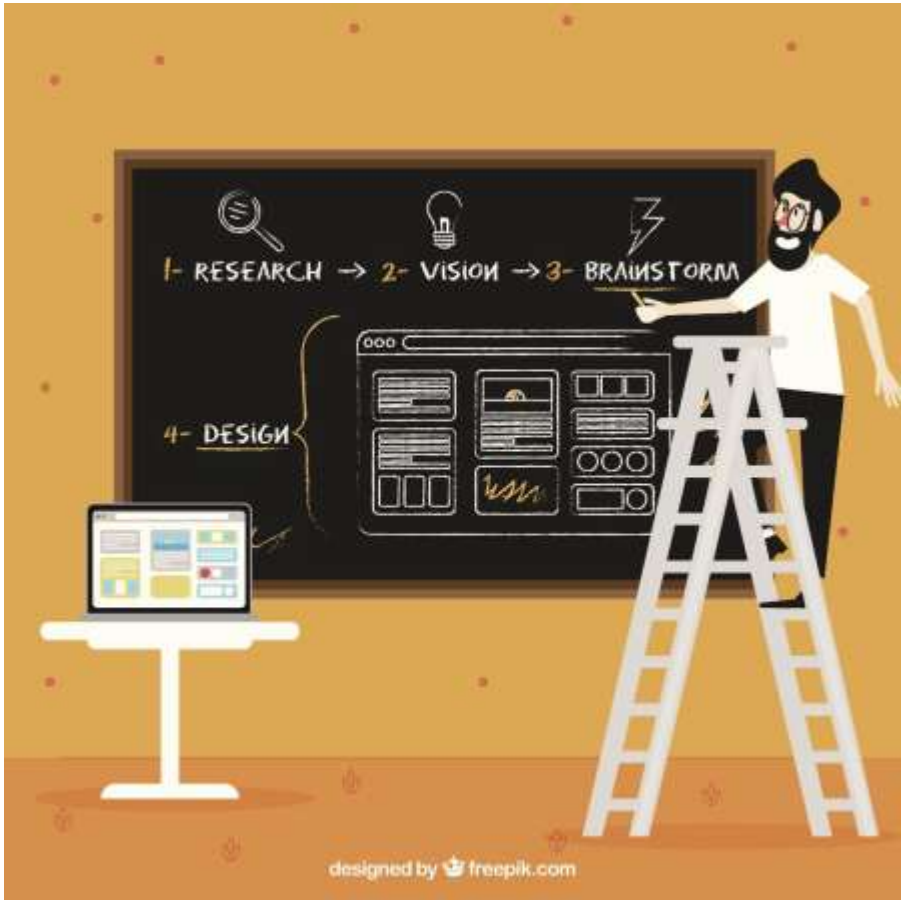
Documenting and Communicating Design Decisions

Dennis Mancl
MSWX Software Experts
dmancl@acm.org

Steven Fraser
Innoxec
sdfraser@acm.org



Designing 2025 Workshop – at ICSE 2025

Who are we?

Steven Fraser
sdfraser@acm.org

- Research center director
- Software best practices
- Open innovation
- Strategic tech transfer
- University-company partnerships
- Engineering learning programs

Dennis Mancl
dmancl@acm.org

- Tech transfer specialist
- Agile software advocate
- Requirements modeling
- Design patterns
- Software quality
- Legacy software techniques

An update on our 2024 Paper

Investing in Design*

Why?

- Share expertise
- Incubate resources
- Catalyze collaboration
- Inspire innovation

How?

- Foster learning
- Support teamwork
- Coach staff

** Steven Fraser, Dennis Mancl: Investing in Software Design, Designing@ICSE 2024: 34-39*

Design: Balancing Discipline and Delivery

Since 2000 – evolution from *process-heavy* to *process-light*

- Agile = a reaction to “High Ceremony Processes”

- ➢ Increase focus on code
- ➢ Avoid “Big Design Up Front”
- ➢ Less effort on low-level design*

* ***Low-level design*** *= <u>documents</u> and <u>models</u> to describe the structure of individual functions and modules*

Light processes

Evolution of Software Products

- Hardware platforms change
- Software “lives” platform-to-platform

Mainframes	Minis + PCs	Web and Smart Devices	Network infrastructure
1960s	1980s	2000s	2020s
Flowcharts	Structure Charts State Machines Data models	Agile Open Source DevOps	Cloud AI Autonomous Systems

How did we get here?

Low-level design models

If low-level design is missing:

- How do we avoid technical debt?
- The challenge of confusing code?

It's connected to a "learning issue" –

- ➢ I learn when someone explains to me
- ➢ I learn when I explain to others

The design process is a collaborative game:

- selecting the correct components
- using design information to drive testing
- building lightweight documentation to help understanding and extending an existing design
- goal: to help our own team – and other teams

** Good structure

** Good communication

It's a learning challenge

Communication challenges

- Talented software developers benefit from sharing information and from personal memory aids

Share!

Can my teammates help?

Aid my memory!

I authored this code,
… but how can I remember?

I wish I had written it down!

Useful design ideas, recorded for the future

- Don’t be afraid to sketch out design ideas early
- Code is linear, invention is not!

Discover and invent

We invent more than just code – we make our own “micro process”

Good tests endure longer than the code…

Tests explain design

Every developer should be able to think and write like an “architect”

Lightweight architecture

Why don’t developers draw diagrams?

Modeling = waste? → The relationship between code and diagrams “quickly drifts into oblivion and usually ends in tears”

Focused modeling = essential → But **abstraction** and communication are absolutely necessary!

- “The code is the truth, but not the whole truth”
- Diagrams can explain: *architecture, patterns, cross-cutting behavior, design rationale, tribal memory*

Grady Booch, “Why don’t developers draw diagrams?”
Keynote talk at SOFTVIS 2010, Oct. 25-26, 2010. https://softvis.wordpress.com/wp-content/uploads/2010/12/gradybooch-whydontdevelopersdrawdiagrams-softvis2010.pdf

Thinking is a waste of time?

Design diagrams, models, and documents

Explain:

Draw selected designs, describe tricky internal mechanism and external contracts

Think:

Take time to "think things through"

Collaborate:

Meet at the whiteboard with friends

Learn:

An opportunity to "learn new tricks"

Explore:

For legacy code, create design pictures as part of "software archaeology"

The value of the design is enriching minds of the current and future development team!

Docs help you think and learn

What's in the paper?

- Three suggestions for lightweight low-level design artifacts
- Old habits to preserve; new habits to instill in young developers

> - Booch's "micro process"
> - Design-focused tests (automated, easy to read)
> - Lightweight architecture docs

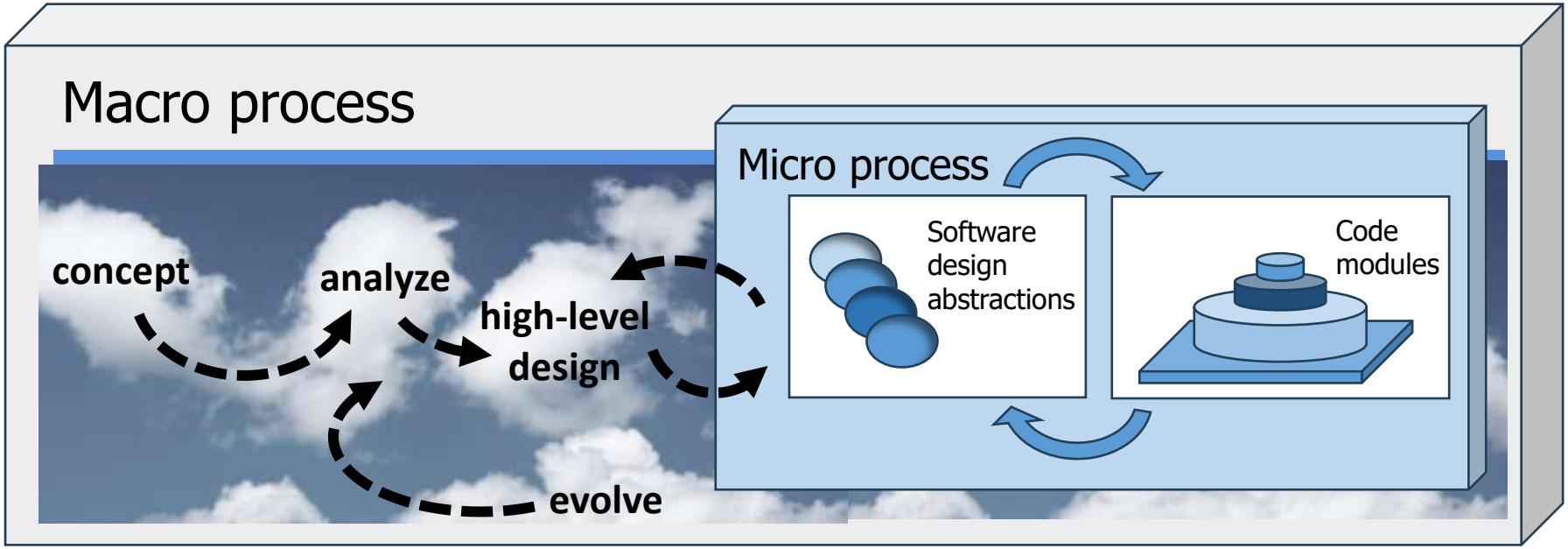
Our advice

Design work by developers

Grady Booch, *Object Solutions: Managing the Object-Oriented Project,* 1996

Design is a set of habits that keep developers organized

- Grady Booch (in the 1990s): “macro process” and “micro process”
- Build design abstractions in each micro process iteration, not just code!
  - “Overlapping waves of **discovery, invention, and implementation**”
  - Within a more nebulous (but necessary) outer loop of concept, analysis, design, evolution

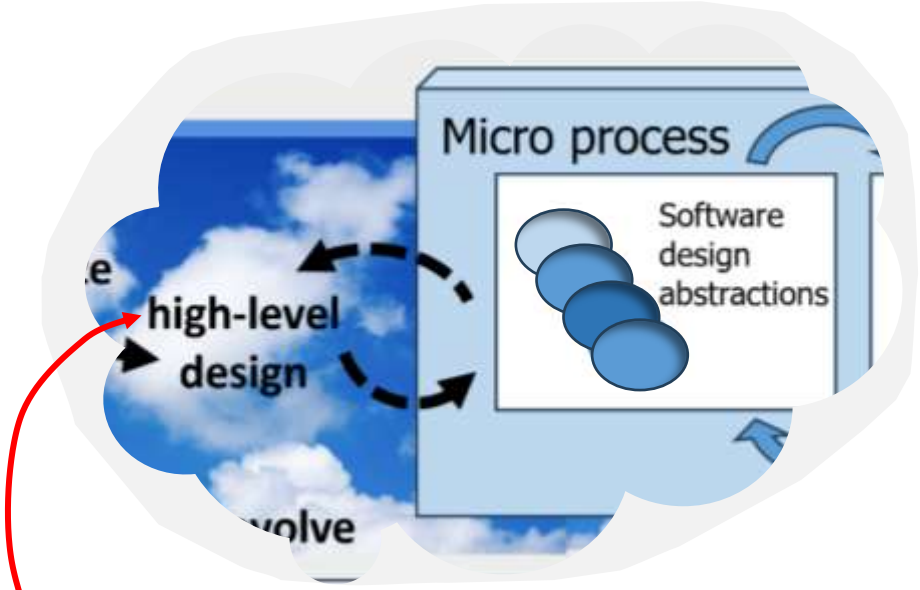


Crossing from abstract to concrete

Keeping models in sync

Change is inevitable

- Need to fix bugs or port to a new platform
- Customers want new features and patches



Use cases and **user stories** are part of the constantly evolving high-level design

Agile projects write many User Stories (part of the high-level design)

- Link new "micro process" design abstractions to new and changed User Stories or Use Cases

Don't go crazy synching the models

- Draw on the whiteboard
- Take a picture of the key changes with your phone
- Upload to your "project database"
- Link to User Stories

Stories are central to evolution

Automated tests for design

Tests that “communicate the design details” (scenarios, constraints, etc.)

Design-focused unit tests – “proud developers” should always make their code crystal clear

- A test that illustrates how great the developer’s code is
  - With good tests, a strong design should be easier to preserve as new code is added

---

- Tests within a single function (algorithm correctness)
- Tests to double-check data integrity after an update
- Tests to check that unauthorized operations are rejected

Tests are design artifacts

Automated tests for design

TDD tests?? No. Design-focused software tests have a different goal than "test-first development" and "acceptance test driven development"

- TDD tests are linked directly to the code…
- Acceptance tests are linked to user stories and the backlog
- Design-focused tests are connected to design decisions

Build design-focused tests as part of the low-level design and coding activities

Connect individual tests to design decisions that are made during the coding process

Tests can probe the internal states within code blocks.

Sometimes a test is better than a picture

Tests are design artifacts

Architecture and risk

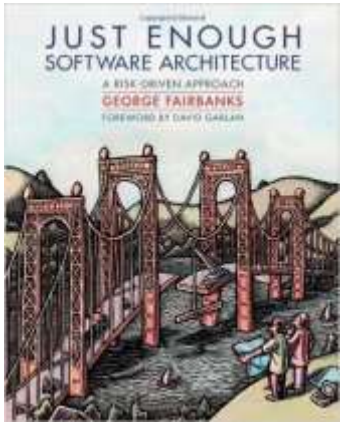
Think about architecture – at a small scale

- Developers are involved in many architectural things
- “Just Enough Architecture” by George Fairbanks
- It’s risk-driven architecture, with small documents

Security	Performance	Handle overload	Leverage activity logs	Recover from outages

Where to invest?

- Selected small architecture “stories”
- Written by developers, not architects



Architecture-in-the-small

Architecture documents in Agile?

- “Architecture stories” on the Product Backlog
- We want everyone to pay attention to scenarios and risks
- “Small” is good… “Architecture Haiku” (Fairbanks)

Backlog should contain:

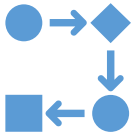
- Visible features ($)
- Hidden arch features
- Defect fixes
- Technical debt

https://philippe.kruchten.com/wp-content/uploads/2012/07/kruchten-110707-what-colours-is-your-backlog-2up.pdf

Summary

Engage

Connect with your software development partners!



Influence

Collaborate

Communicate

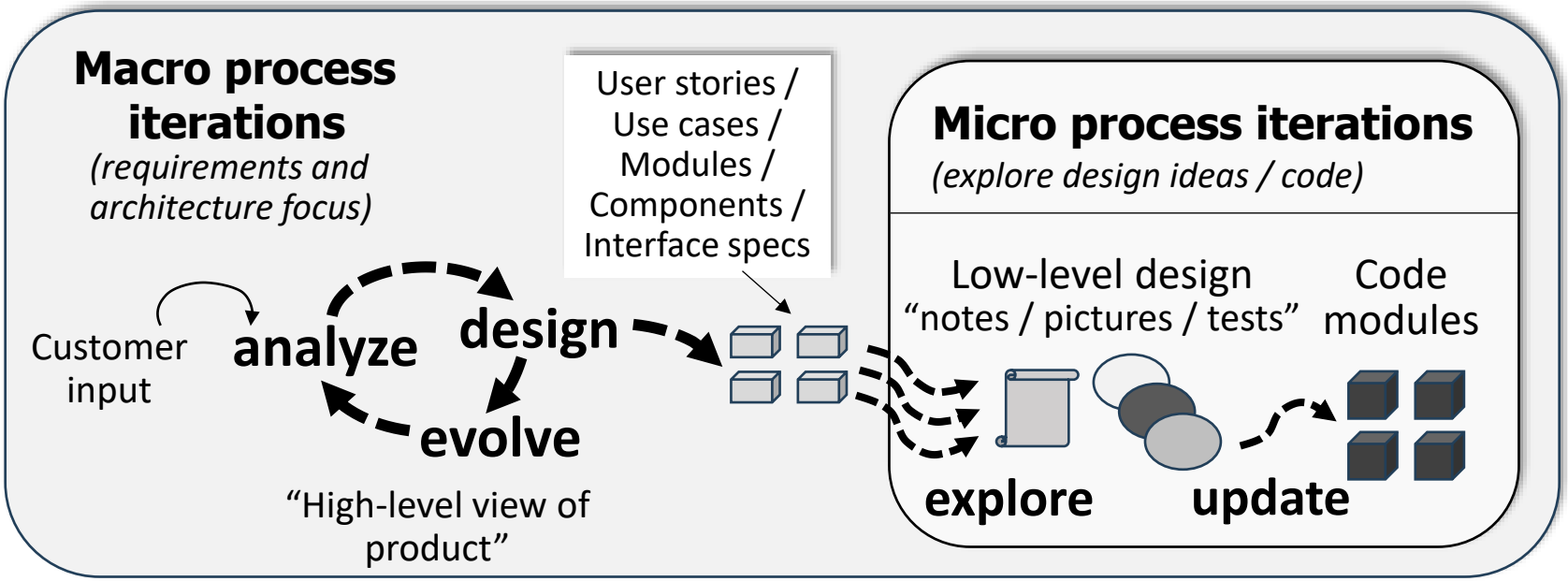
Teach

Thinking skills and an effective micro process

Make the code understandable

Extra slides

Another view of macro/micro processes



Notes = low-level stories, simple architecture view (in text), constraints, invariants
Pictures = simple arch view (in UML or other notations), interfaces, communications
Tests = essential stable module behavior plus story tests
as lightweight as possible

Grady Booch and “Five Habits”

- *Object Solutions: Managing the Object-Oriented Project*, 1996, p. 25
  - In this book, Booch is a champion of “iterative and incremental life cycle”
  - Not quite Agile, but definitely not Waterfall
  - Hopelessly old fashioned? No…
  - Booch thinks of “systems” instead of code
  - And he advocates a culture of communication and modeling

- Booch gives a list of 5 Habits of Successful Object-Oriented Projects:
  - A ruthless focus on the development of a system that provides a well-understood collection of essential minimal characteristics.
  - The existence of a culture that is centered on results, encourages **communications**, and yet is not afraid to fail.
  - The effective use of object-oriented **modeling**.
  - The existence of a strong **architectural vision**.
  - The application of a well-managed **iterative and incremental** development life cycle.