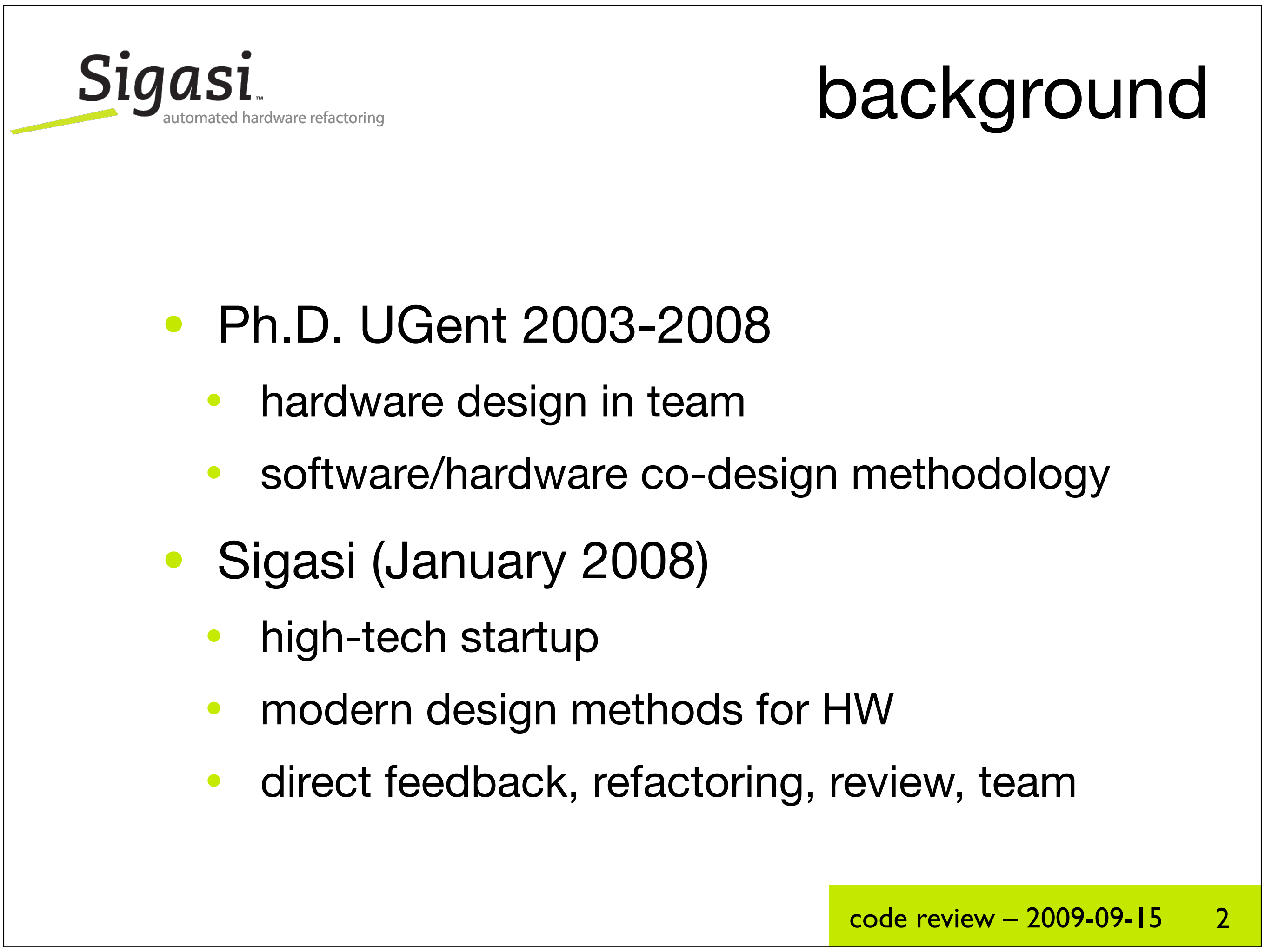
Sigasi™
automated hardware refactoring

code review for hardware designers

Philippe Faes

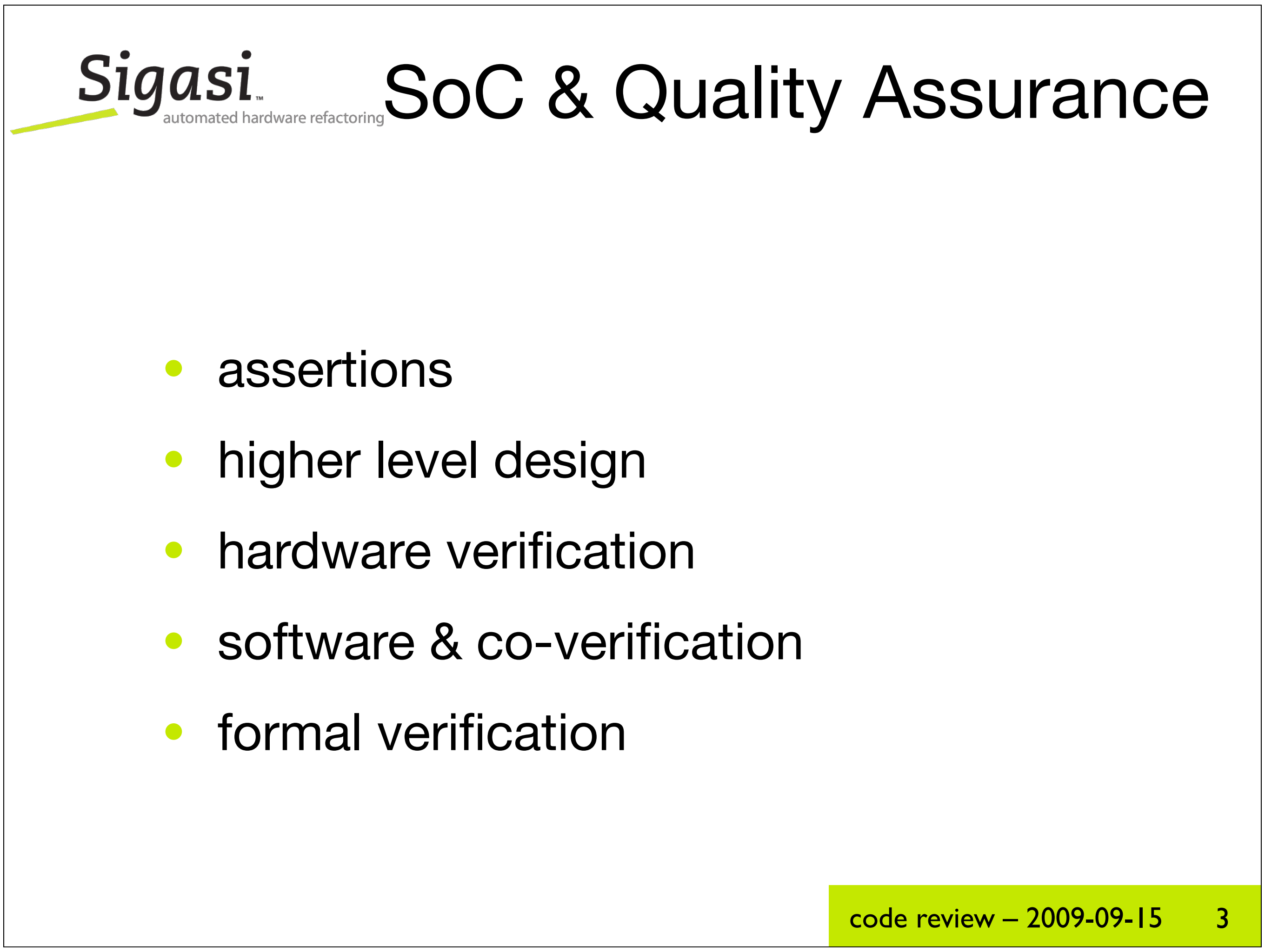


Sigasi™ automated hardware refactoring

background

- Ph.D. UGent 2003-2008
  - hardware design in team
  - software/hardware co-design methodology
- Sigasi (January 2008)
  - high-tech startup
  - modern design methods for HW
  - direct feedback, refactoring, review, team

Philippe Faes, CEO Sigasi
philippe.faes@sigasi.com
http://www.sigasi.com



Sigasi automated hardware refactoring

SoC & Quality Assurance

- assertions
- higher level design
- hardware verification
- software & co-verification
- formal verification

In order to ensure the quality of complex digital systems (especially Systems-on-Chip), several quality assurance techniques have to be used together, including:
* use of in-code assertions
* increasing the level of abstraction (transistors -> gates -> RTL -> ESL/TLM)
* verification of the hardware design, based in simulation and/or emulation
* for systems that include software: HW/SW co-verification (using co-simulation or emulation)
* for critical parts of the system: formal verification.

Even for moderately complex systems, it is impossible to attain full coverage using simulation techniques.
Formal methods are only possible on small, well-defined systems.

Partial solution:
* formal methods only on critical parts
* constrained random testing

But: more QA techniques are required...

Sigasi™
automated hardware refactoring

QA needs inspection

“Code has bugs because engineers do not understand what the code does.
So design, not verification is the real issue.”

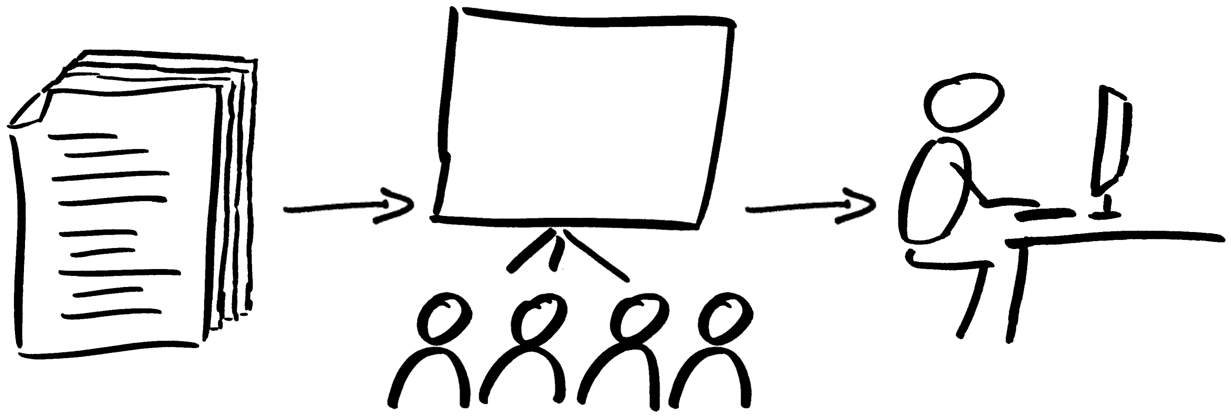
Michael Keating,
Synopsys Fellow
co-author Reuse Methodology Manual

The design and implementation of the code defines the quality of the final product.
Thus, in contrast to traditional black-box testing, inspecting the code is extremely important for code quality.

Before implementing code reviews, you should first consider if code reviews should be first on your list for increasing your QA. Code reviews are just one technique that increase quality and perhaps other techniques will give you a better return on investment.
Are you well organized with regard to revision control, test planning and testing. Do you have coding guidelines? Do you have a system for error reporting and a system. Do you have a means for automated integration (i.e. build scripts)? What about continuous integration (e.g. nightly builds)?

Sigasi
automated hardware refactoring

reviews



Example of a code review:
Before meeting
* four team members get notification of upcoming review meeting
* all four print the code that will be reviewed and read through it, with a pencil and fluorescent marker
* all remarks are collected into a single document
During meeting
* the code is projected on a screen
* the entire list of remarks is processed, each is marked as valid, invalid, major, minor, ...
After the meeting
* the original programmer implements ("fixes") all the valid remarks.

Problems:
* time consuming
* not fun
* insufficient follow-up (not traceable)

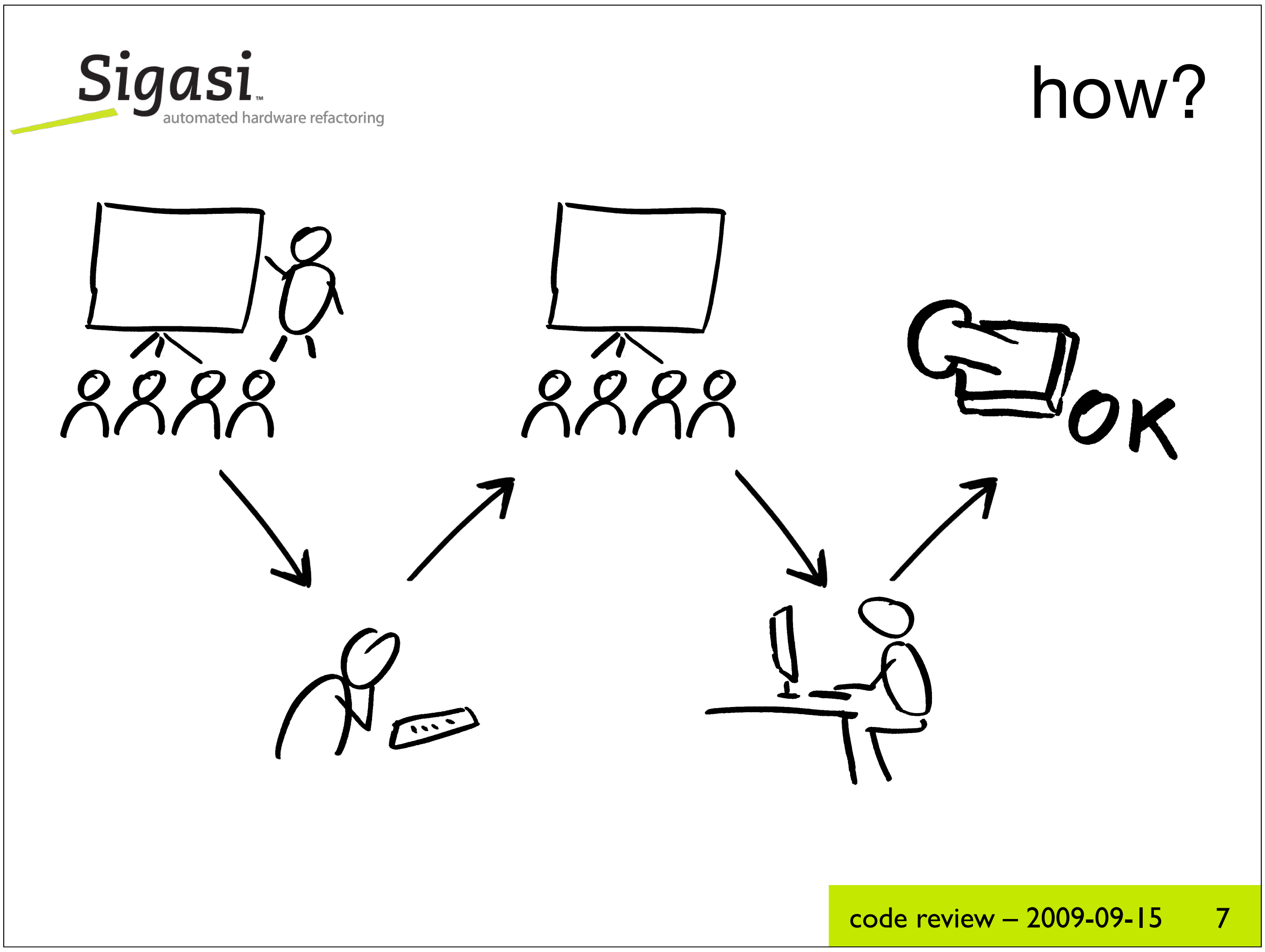
Sigasi™
automated hardware refactoring

why?

- maintainable / reusable
- transferable
- observable
- predictable

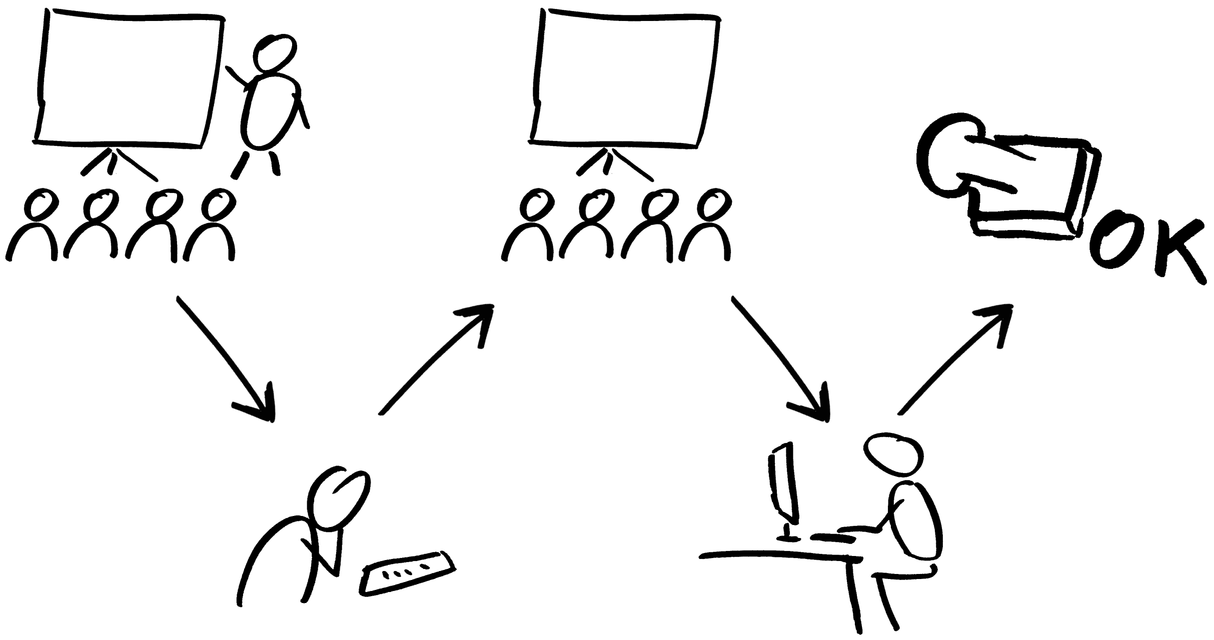
There are several reasons for code review. Ask yourself the following questions:
* Will I be able to build a product in five years, based on today's hardware design. Or alternatively: can I reuse three year old code for today's new killer application?
* Will anybody be able to continue working on this code if the engineer who designed it gets hit by a truck? Or: does anybody understand the code that was designed by John, who has left the company.
* If a 10 month project has been running for 5 months, how can we know if we are on schedule.
* If my hardware design team tells me the project is *almost* finished, what does that mean?

Of course code review will not provide magic answers. Rather, it will *help* solve some important business problems.



Sigasi automated hardware refactoring

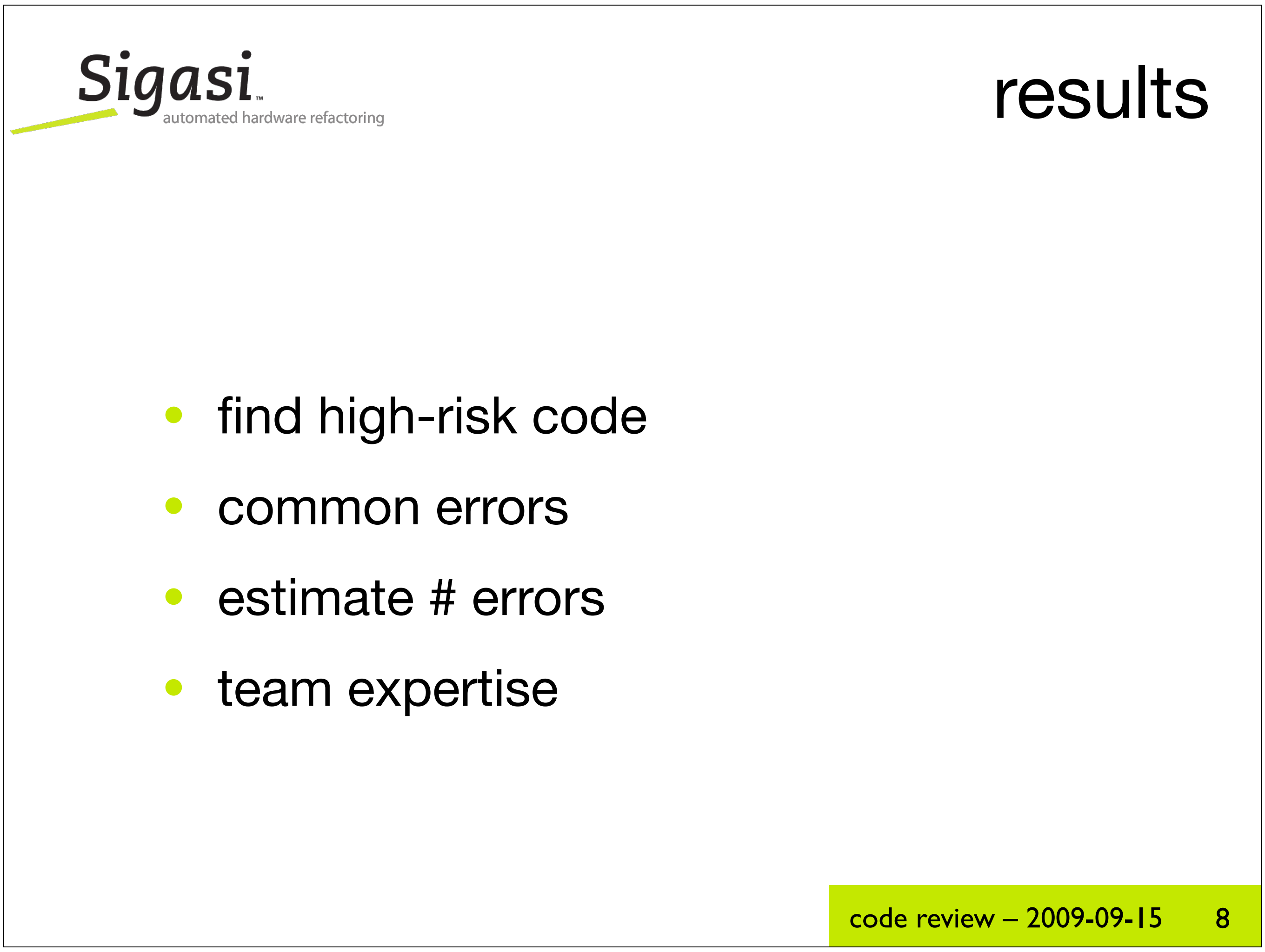
how?



Fagan-style code inpection
* 1976, IBM
* procedure driven
* early computer languages: COBOL, FORTRAN, PL/1

Key participants:
* moderator (people skills!)
* engineer who’s code is to be inspected
* team of reviewers (peers)

Procedure
1. Overview
 engineer explains the context to the team.
 goal is to communicate and educate
2. Preparation
 team reads the code under review
 goal is to educate themselves
3. Inspection
 engineer explains in detail how the code works (line by line) and tries to convince the team of its correctness
 team tries to identify errors
 goal is find errors
4. Rework
 engineer fixes the errors
5. Follow-up
 moderator checks the fixes and decides either to approve or to call for another full review


Sigasi™ automated hardware refactoring

results

- find high-risk code
- common errors
- estimate # errors
- team expertise

As a result of code inspection, critical areas of the code will be identified.
A list of common coding problems will be identified. This helps increase quality of future work.
After reviewing a small part of the code, estimates can be made for the total number of errors in the code. This can trigger further inspection or other QA procedures.

Measured results (Fagan 1976):
* +23% productivity
* finds 2/3 of bugs
* increase predictability

Never use these results to judge individual engineers! (-> killing the goose that lays golden eggs)

As an important result, your team will build and share competence in designing and inspecting hardware.

Sigasi automated hardware refactoring

hardware specifics

simulation vs synthesis

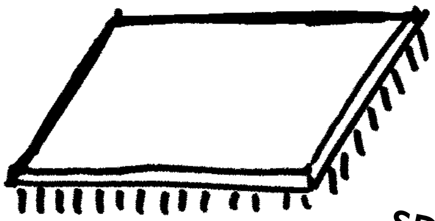
tool support

timing

massively parallel

spatial structure

analog effects

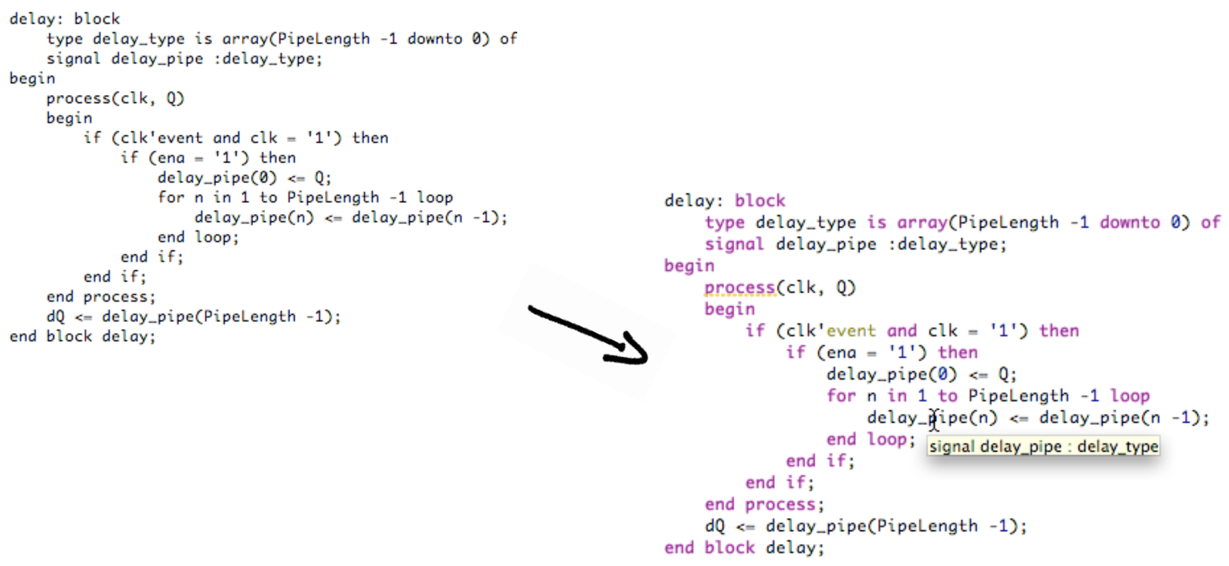


HW and SW are increasingly similar
Still some (important) differences

solutions:
* use the very best tools (tools are cheap compared to the people using them)
* also review synthesis reports (ideally, synthesis warnings should appear inside the source code)
* extra attention (inspection) of areas with increased risk (multiple clocks, asynchronous hardware, ...)
* ...

Sigasi™
automated hardware refactoring

modern tools



```
delay: block
    type delay_type is array(PipeLength -1 downto 0) of
    signal delay_pipe :delay_type;
begin
    process(clk, Q)
    begin
        if (clk'event and clk = '1') then
            if (ena = '1') then
                delay_pipe(0) <= Q;
                for n in 1 to PipeLength -1 loop
                    delay_pipe(n) <= delay_pipe(n -1);
                end loop;
            end if;
        end if;
    end process;
    dQ <= delay_pipe(PipeLength -1);
end block delay;
```

```
delay: block
    type delay_type is array(PipeLength -1 downto 0) of
    signal delay_pipe :delay_type;
begin
    process(clk, Q)
    begin
        if (clk'event and clk = '1') then
            if (ena = '1') then
                delay_pipe(0) <= Q;
                for n in 1 to PipeLength -1 loop
                    delay_pipe(n) <= delay_pipe(n -1);
                end loop; signal delay_pipe : delay_type
            end if;
        end if;
    end process;
    dQ <= delay_pipe(PipeLength -1);
end block delay;
```

Traditionally:
* print code and work through it with pencil or fluorescent marker
* mark review notes on different medium (paper, excel sheet,...)
Disadvantages:
* navigating code = visual inspection / turning pages
* no link between review notes and code (-> manual navigation between error notes and code)
* not easy to answer questions like: where is this signal used, what is the value of this constant, what is the datatype of this variable, is this signal meant to be active-low?
* no link between the fix and the error note

Proposal:
* single tool that displays code and links code with error notes
* support for easy navigation through design
* in-line comments should be easily accessible, even from different files
* integration between revision control and review notes

Sigasi™ automated hardware refactoring

your own procedures

- what is your goal?
- in which phase find problems?
- how will you follow up?

If you want to start off with code reviews, the procedure you design for code inspection should fit the needs of your specific organization.
Some import things to consider:
Is the goal
* only to find flagrant errors?
* to increase the general code quality (transferability, reusability)
* to train the team and establish best coding practices?
In which phase of the review process will you want to find problems?
* during the preparation (= solo activity)
* during the actual review meeting (= team activity)
How will you follow up after the meeting? How will you know that all problems are actually fixed?
Do you have to report (to customers?) about your QA process? (cf. avionics!)

Sigasi™

automated hardware refactoring

Good luck implementing code reviews in your organization.