Prof. Philip Koopman

Carnegie Mellon University

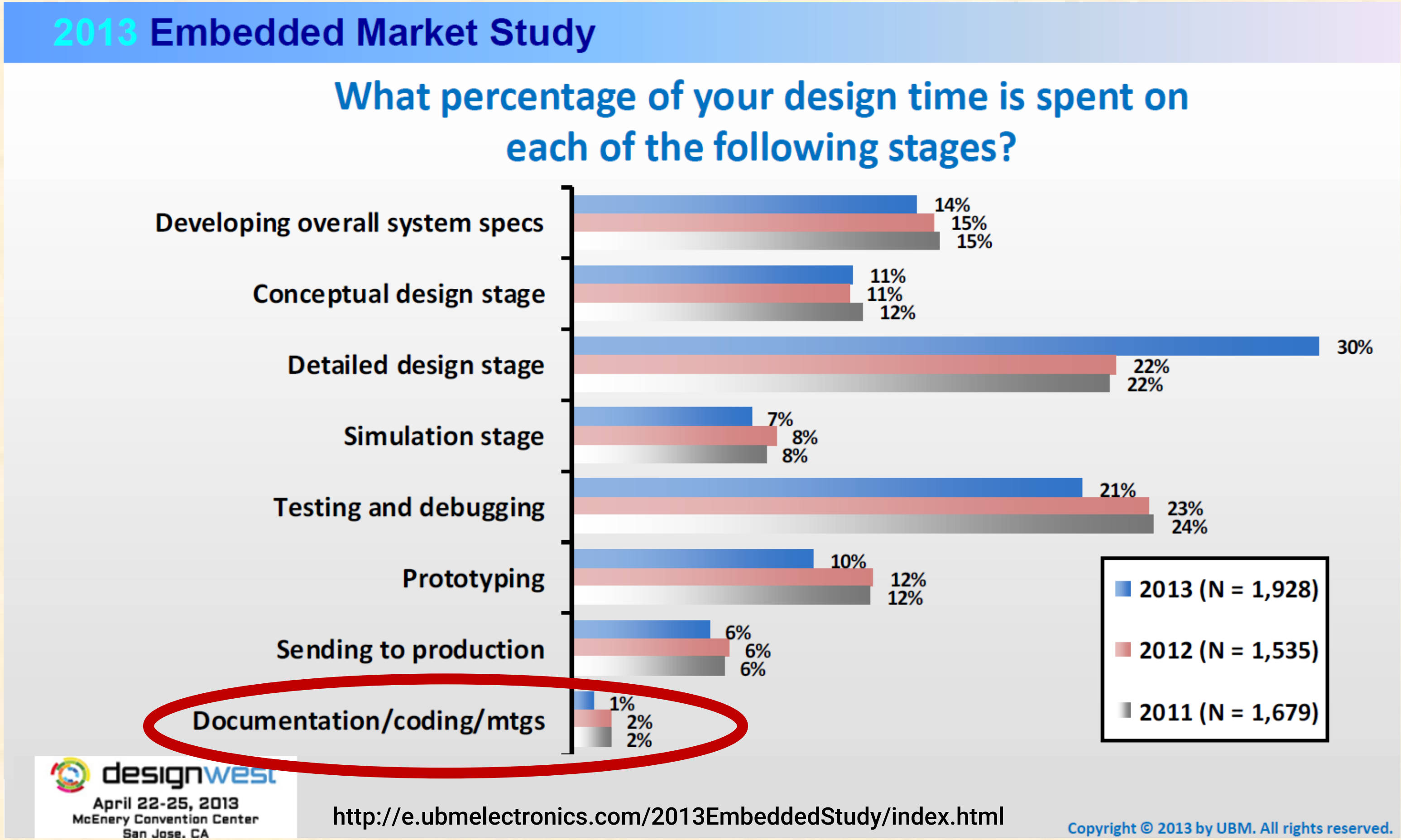
Electrical & Computer ENGINEERING

Software Development Processes

“Without requirements and design, programming is the art of adding bugs to an empty text file.”

— *Louis Srygley*

Coding Is Essentially 0% of Creating Software

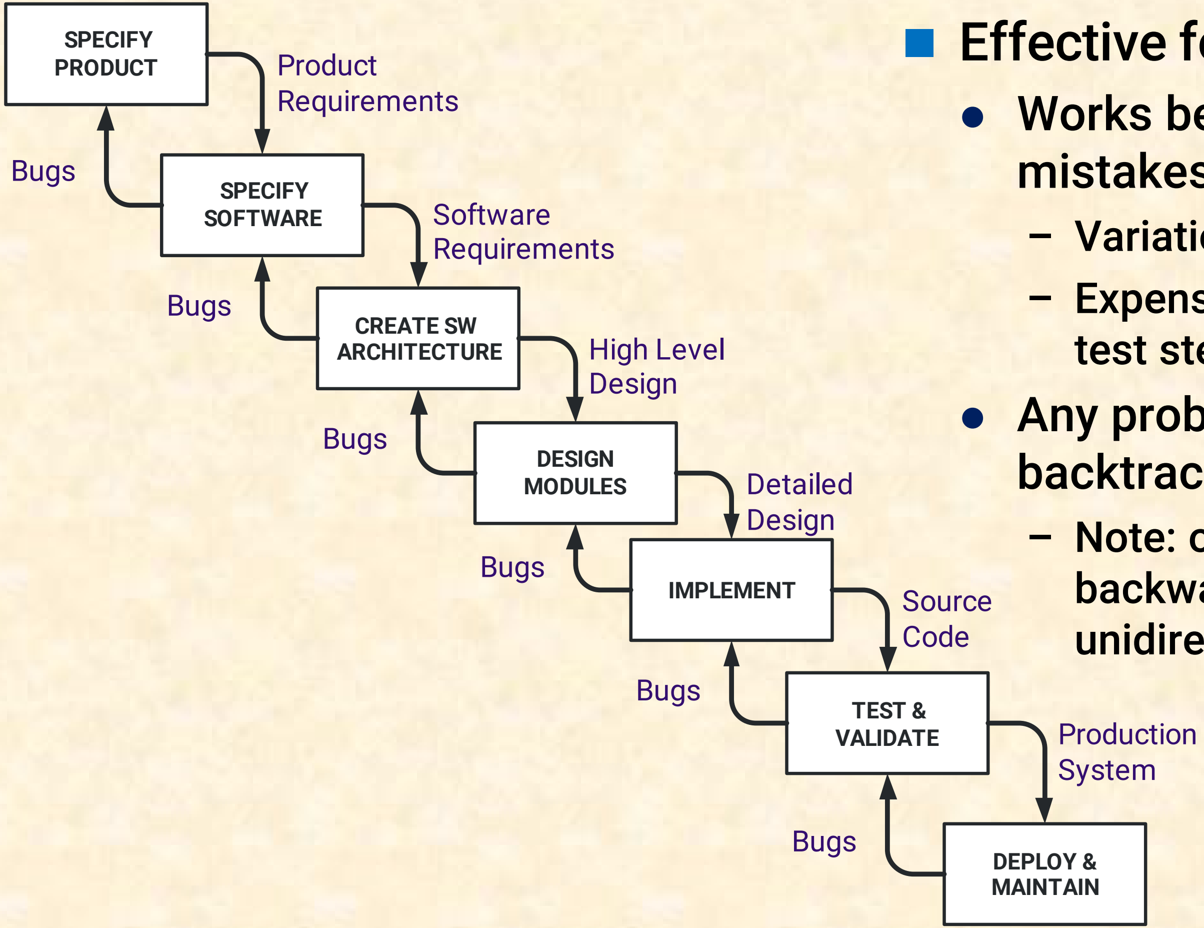


2013 Embedded Market Study

What percentage of your design time is spent on each of the following stages?

http://e.ubmelectronics.com/2013EmbeddedStudy/index.html

Old-School Waterfall Development Cycle



- Effective for well understood domains
  - Works best if you don’t make many big mistakes
    - Variations on existing systems
    - Expensive to fix things that escape to test steps
  - Any problem encountered requires backtracking
    - Note: original waterfall paper had these backward arrows! It was never just a unidirectional process

What We've Learned in 50+ Years of Software

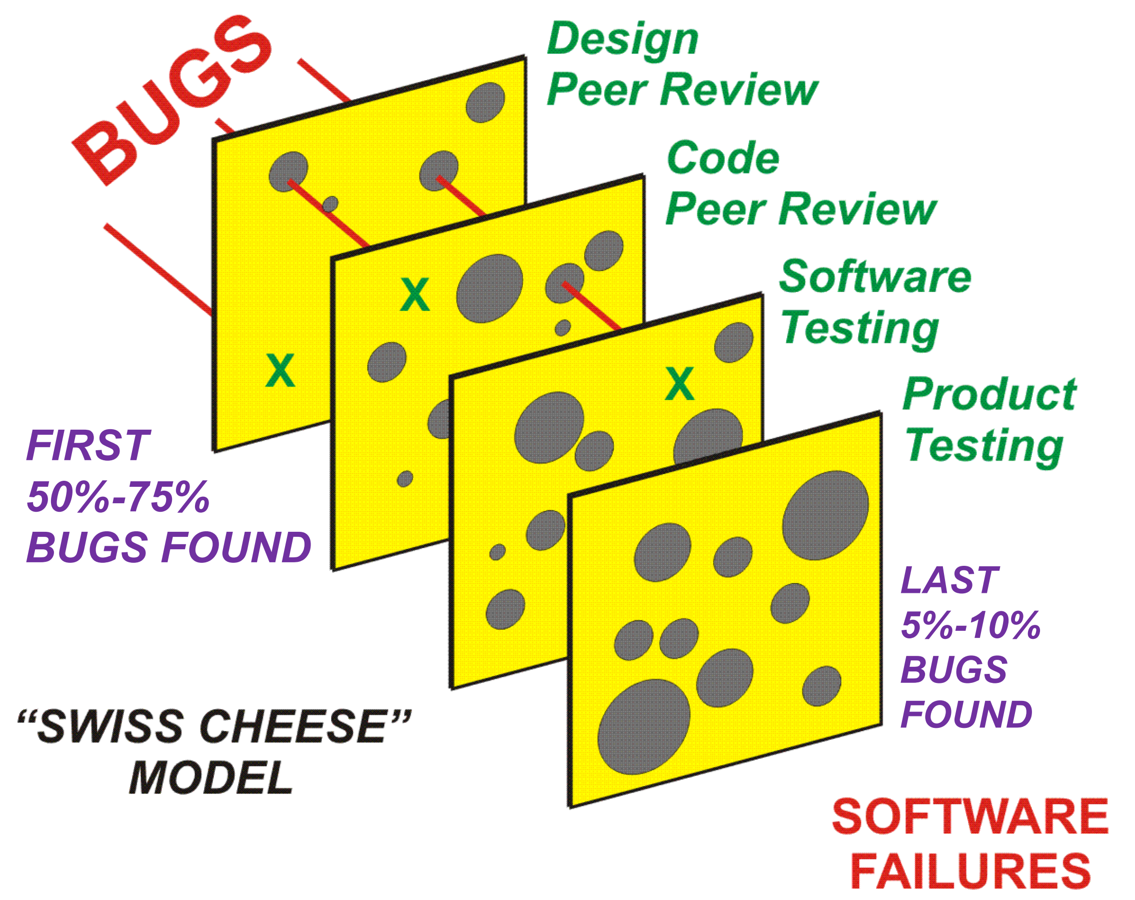
- Dividing up into subsystems is critical
  - Bad architecture will doom a project
- Process formality is a good investment
  - Traceability, formal reviews, etc.
  - Skipping steps costs more in the end
- Requirements change
  - Suggests using an iterated approach
- Finding bugs early is important
  - Traceability from high to low levels
  - Layered testing
  - Peer reviews most cost effective for this

- **If the second half of the project is "debugging" that must mean the first half is "bugging"**

— Jack Ganssle

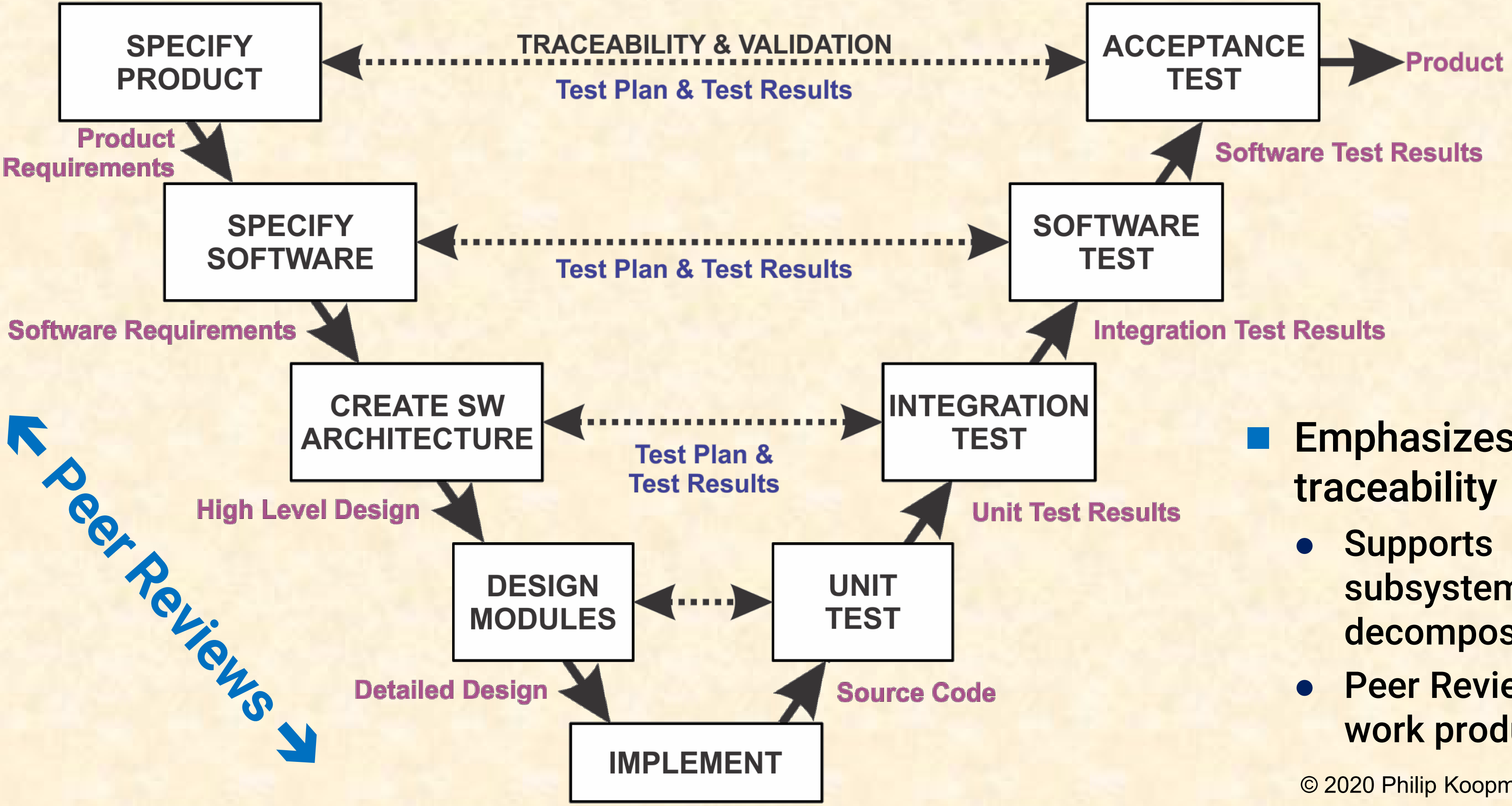
http://www.ganssle.com/rants/ontesting.htm (paraphrase)

Finding Bugs Before Product Test



- **Product Testing**
  - Late & Expensive
  - Many field escapes
- **Software Testing**
  - Unit & Integration test
- **Code Peer Review**
  - Earlier & Cheaper
- **Design Peer Review**
  - Earlier & Cheaper

V (or “Vee”) Development Cycle

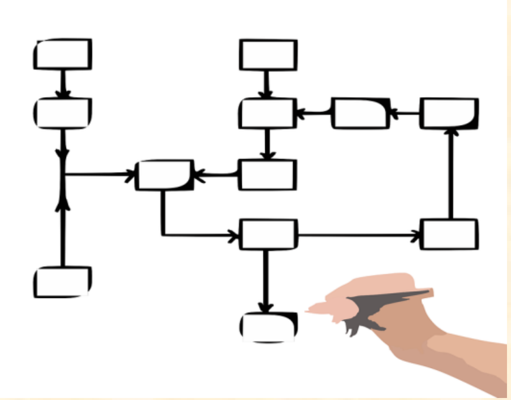


- Emphasizes traceability
  - Supports subsystem decomposition
  - Peer Reviews of work products

A Design Is Not The Code

- Implementation: the code itself
  - Comments describe the implementation; they aren't the design

- Detailed Design (DD)
  - Flowcharts
  - Statecharts
  - Algorithms, control diagrams, etc.



https://pixabay.com/en/flowchart-diagram-drawing-concept-311347/

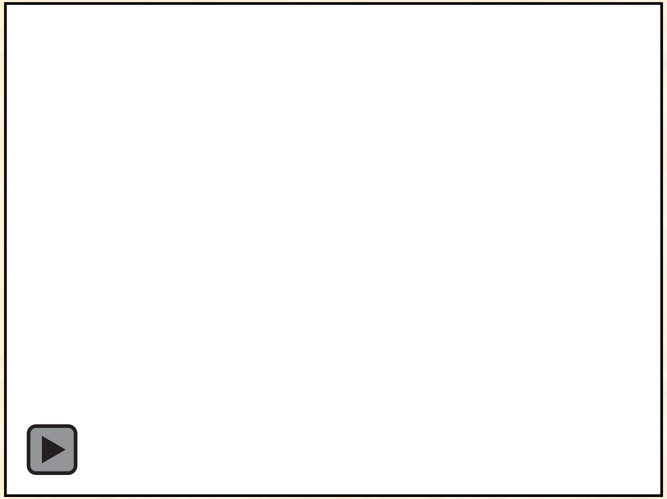
- High Level Design (HLD): architecture, component defs.
  - Pieces of the system (e.g., classes, subsystems)
  - Functional allocation to the pieces
  - Interfaces between the systems

Requirements on Top Left of Vee

- Software Requirements Specification (SRS)
  - Says “what” the software does, not “how” it does it
    - If it’s not in the SRS, the software shouldn’t do it
    - Avoids details unless mandatory due to marketing reqts.
  - Often paired with a Hardware Requirements Spec.

- Product Requirements Specification (PRS)
  - Market-facing product requirements
    - What the system does from a user point of view
  - Point of interface between software group and others
    - Might just be a feature list
    - Might be in form of customer-specified acceptance test

If you think good design is expensive, you should look at the cost of bad design!

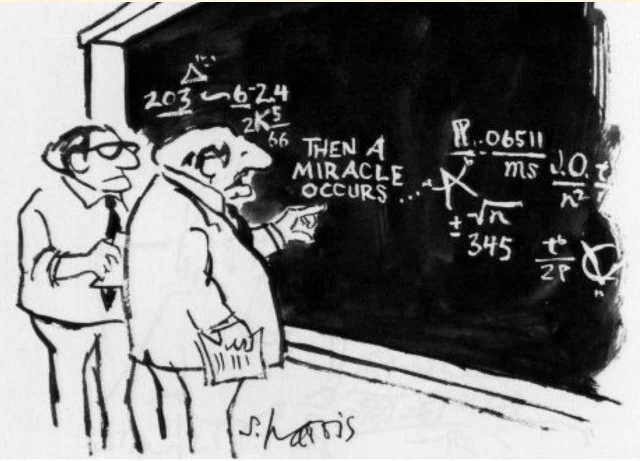


https://youtu.be/j-zczJXSxnw

https://goo.gl/ZVRH9Y

Verification & Validation on Right Side of Vee

- Unit Test: Traces to DD
  - Test individual subroutines, procedures, “modules”
- Integration Test: Traces to HLD
  - Test module interactions (e.g., sequence diagrams)
- Software Test: Traces to SRS
  - Test functionality knowing how software is built
- Acceptance Test: Traces to PRS
  - Test customer-facing functionality
- Other activities:
  - Software Quality Assurance (SQA): did you follow the steps?
  - Peer Reviews: check quality of every step
  - Regression Test: test after bug fix to make sure bugs stay dead



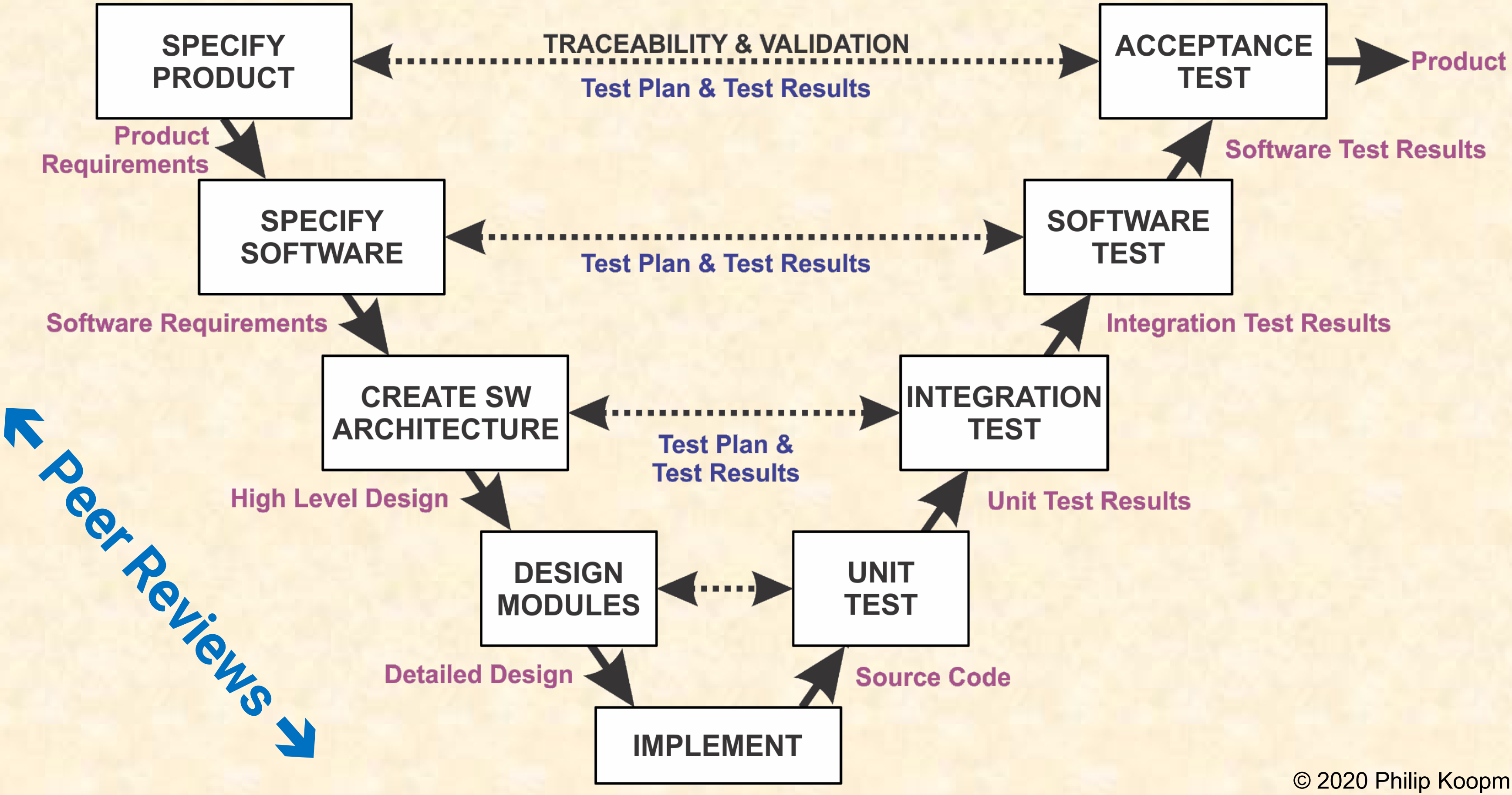
“I think you should be more explicit here in step two.”

from *What's so Funny about Science?* by Sidney Harris (1977)

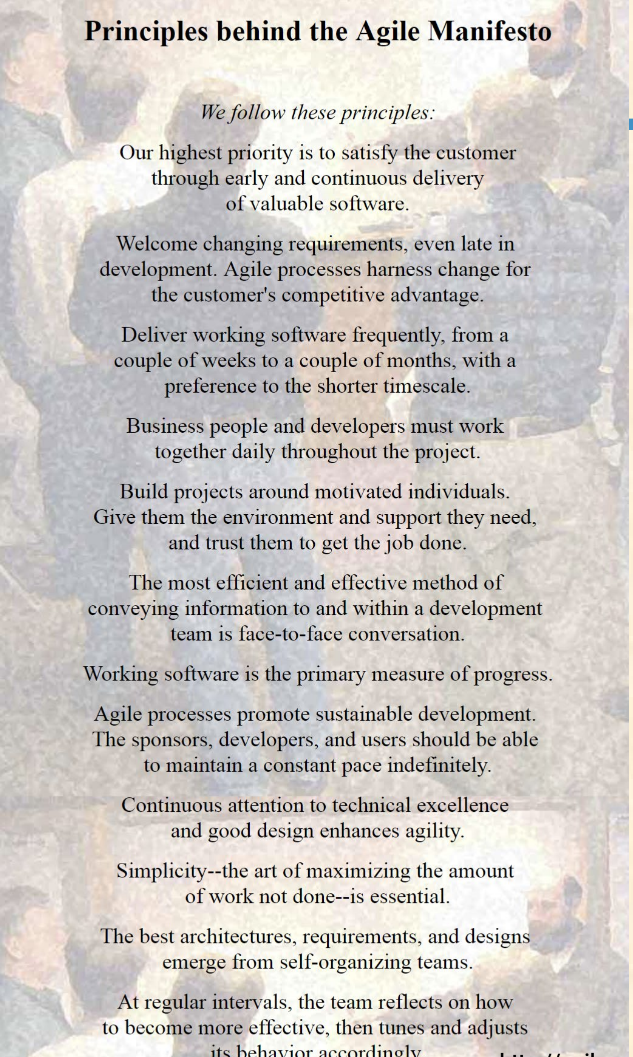
How Much “Paper” Is Enough?

- Old military development saying:
  - Deploy when the paper is heavier than the system. (Even aircraft carriers!)
- Does all this mean you need to be buried in paper? No.
  - Paper required to check process health
    - Be clever about minimizing paper bulk
    - But if code has *no* paperwork, throw the code out
  - Put things on paper as you go through the Vee
    - “Documentation” after writing code is really inefficient
    - If you aren’t going to maintain paper, throw it out

Review: How Do the Pieces Fit Together?



Agile Methods



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://agilemanifesto.org/principles.html

- Agile generally values:
  - Individuals and Interactions over processes and tools
  - Working Software over comprehensive documentation
  - Customer Collaboration over contract negotiation
  - Responding to Change over following a plan
- Example: Scrum
  - Daily “stand up” (“scrum”) meetings for face-to-face collaboration
  - 2-4 week long sprints to incrementally add functionality
    - Each sprint implements items from a backlog
    - Demo at end of sprint; theoretically a shippable product
  - User stories serve as requirements
  - Scrum challenges
    - Geographically split teams with informal communication
    - External dependencies (e.g., other parts of system change)
    - No time for extensive testing, especially embedded hardware

Introduction to Scrum

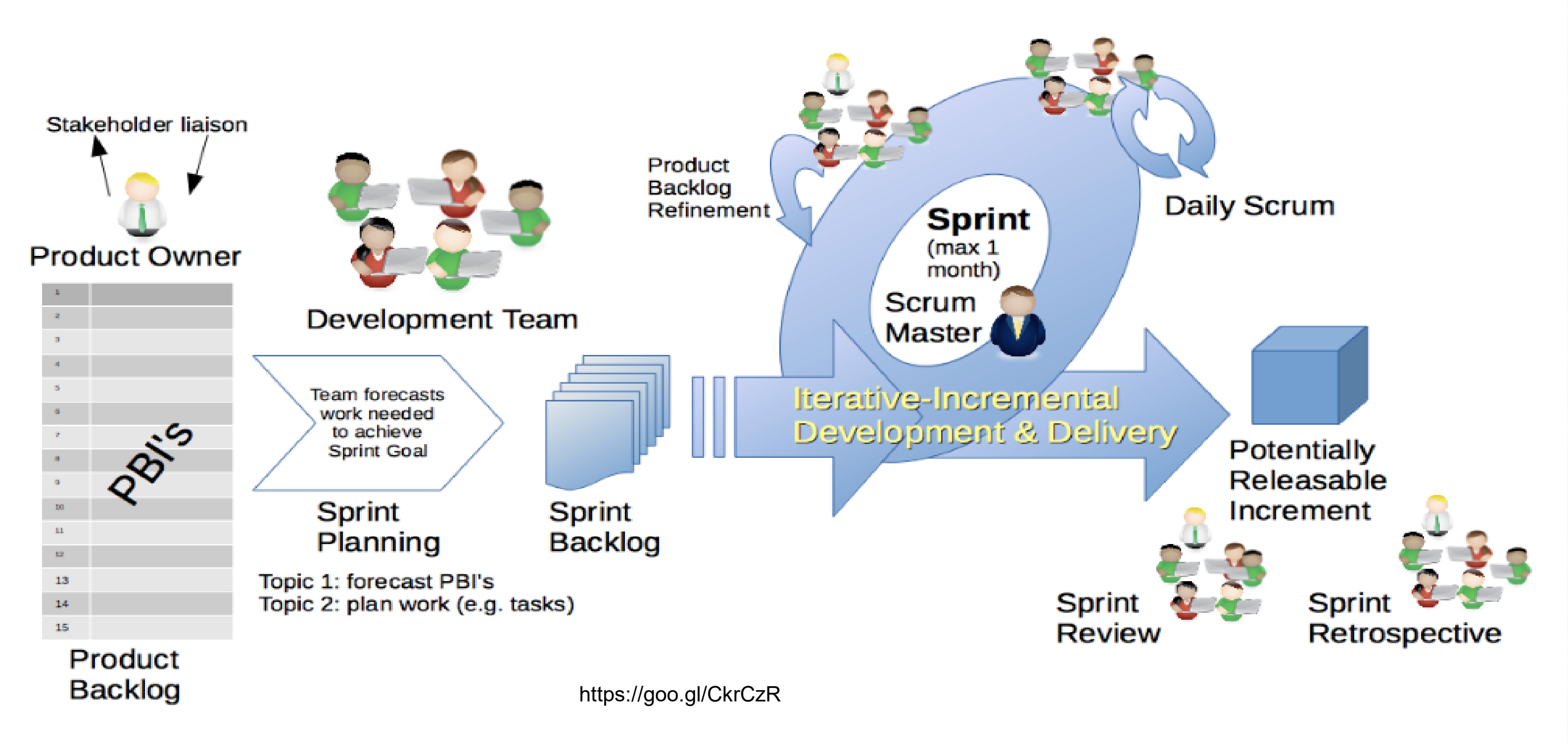
A 7 minute training

Please watch this video:
https://youtu.be/9TycLR0TqFA

by Steve Stedman

Scrum Process Example

- Heavy on implicit knowledge
  - Where are the: requirements, design, test plan, acceptance test



https://goo.gl/CkrCzR

When Is Agile a Good Fit?

Source: Boehm & Turner 2004, Balancing Agility and Discipline

- Agile:
  - Small teams; small products
  - "Everyday" software quality
  - Fast requirements change
  - High-skill experts throughout project
    - Including life-cycle maintenance
  - Developers can handle being empowered; usually senior

- Plan-Driven (waterfall; V)
  - Large teams; large products
  - Mission-critical products
  - Stable requirements
  - High skill primarily in design phase
    - Major versions require expert design
  - Most developers are not empowered; usually junior

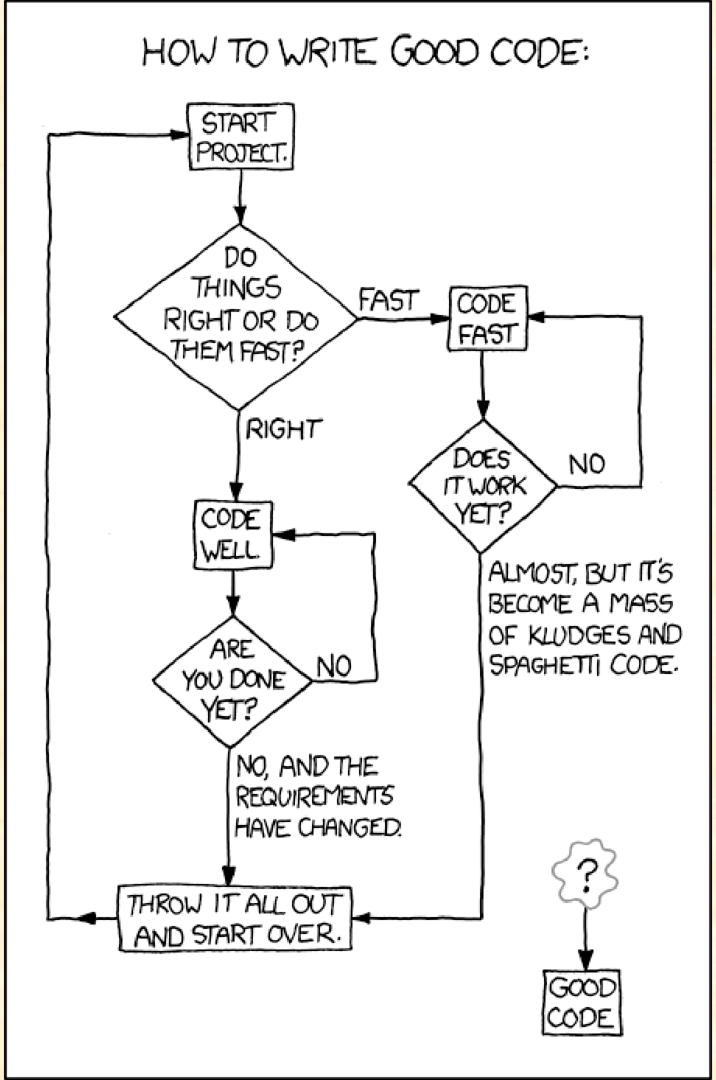
Agile Methods + Embedded (?)

- Significant benefit is that it makes (good) developers happier
  - If done well can help with evolving requirements
  - But, but you need to manage and moderate the risks
- Issue: "Agile" is not just cowboy coding
  - Undefined, undisciplined processes are bad news
  - Yes, Agile teams should follow a rigorously defined process
- Issue: "No-paper" Agile unsuitable for long-lived systems
  - Implicit knowledge is efficient, but evaporates with the team
  - 10+ year old undocumented legacy systems are a nightmare
- Issue: Agile assumes 100% automated acceptance test
  - 100% automated system test is often impractical for physical interfaces
  - Often implicitly assumes that defect escapes are low cost because a new version is 2-4 weeks away
- Issue: Agile typically doesn't have independent process monitoring (SQA)
  - Software Quality Assurance (SQA) tells you if your process is working
  - Agile teams may be dysfunctional and have no idea this is happening
    - Or they may be fine – but who knows if they are really healthy or not?

Best Practices For Software Process

- Follow a defined process
  - Must include all aspects shown on Vee
    - And SQA, Peer Reviews
  - It's OK to rename and reorganize steps
    - All the steps have to get done
    - Common to see "AgileFall" etc.
    - Also common to see bad process dressed up with the latest buzzwords

- Software Process Pitfalls
  - Skipping steps to get to testing faster means more bugs in test
    - Finding bugs is more expensive in testing
  - Using the wrong process for the wrong purpose
    - 3-Week product life and 30 year product life are different situations


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

https://xkcd.com/844/