SOFTWARE ENGINEERING

Agile Software Development (Part 3 — XP, Lean, Kanban) and DevOps

Course ID 06016410, 06016321

Nont Kanungsukkasem, B.Eng., M.Sc., Ph.D.
nont@it.kmitl.ac.th

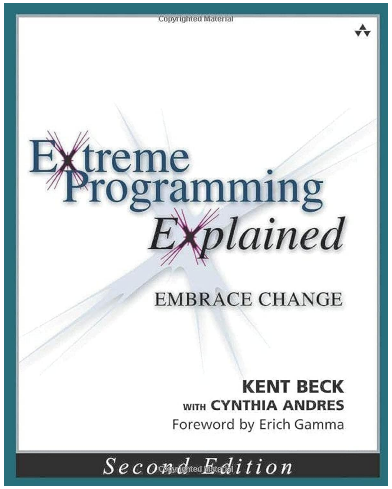
Extreme Programming (XP)

- One of the most widely used approaches to agile software development, originally proposed by *Kent Beck*

- The basic idea of extreme programming is to take to an extreme each of several known good practices
  - "The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible, like testing and reviews. The second time there was a lot more on the line. I thought, "Damn the torpedoes, at least this will make a good article," [and] asked the team to crank up all the knobs to 10 on the things I thought were essential and leave out everything else." — Kent Beck

Extreme Programming (XP)

- XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements. (The 1st edition)
- Then, in the 2nd edition,
  - XP is lightweight
  - XP is a methodology based on addressing constraints in software development.
  - XP can work with teams of any size
  - XP adapts to vague or rapidly changing requirements.



Extreme Programming (XP)

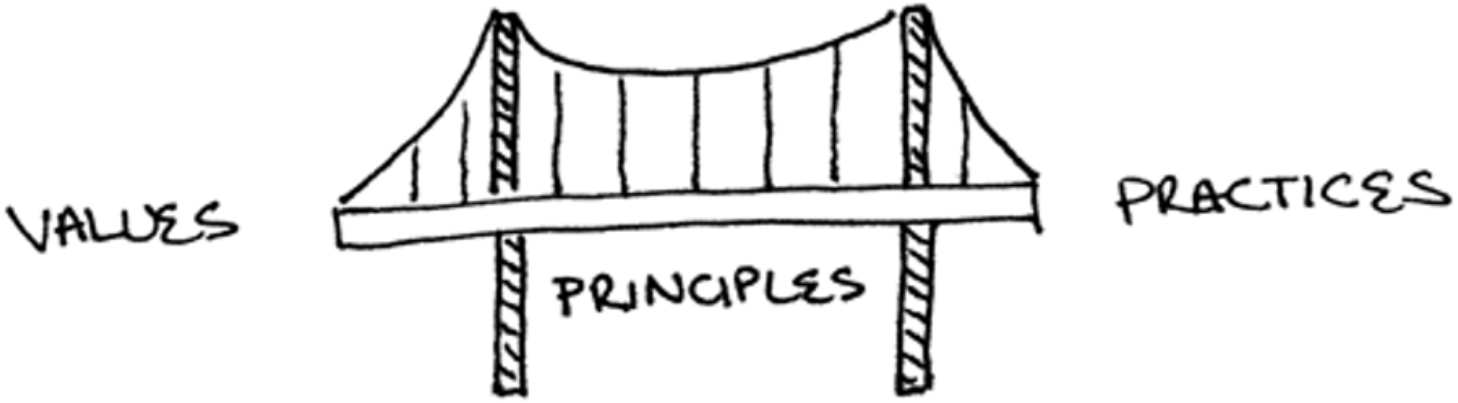
- XP is a path of improvement to excellence for people coming together to develop software. It is distinguished from other methodologies by:
  - Its short development cycles, resulting in early, concrete, and continuing feedback.
  - Its incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.
  - Its ability to flexibly schedule the implementation of functionality, responding to changing business needs.

Extreme Programming (XP)

- Its reliance on automated tests written by programmers, customers, and testers to monitor the progress of development, to allow the system to evolve, and to catch defects early.
- Its reliance on oral communication, tests, and source code to communicate system structure and intent.
- Its reliance on an evolutionary design process that lasts as long as the system lasts.
- Its reliance on the close collaboration of actively engaged individuals with ordinary talent.
- Its reliance on practices that work with both the short-term instincts of the team members and the long-term interests of the project.

Extreme Programming (XP)

- XP is giving up old, ineffective technical and social habits in favor of new ones that work.
- XP is fully appreciating yourself for total effort today.
- XP is striving to do better tomorrow.
- XP is evaluating yourself by your contribution to the team's shared goals.
- XP is asking to get some of your human needs met through software development.



XP Values

- **Communication**
  - When you encounter a problem, ask yourselves if the problem was caused by a lack of communication.
  - What communication do you need now to address the problem?
  - What communication do you need to keep yourself out of this trouble in the future?
- **Simplicity**
  - When you need to change to regain simplicity, you must find a way from where you are to where you want to be.
  - Improving communication helps achieve simplicity by eliminating unneeded or deferrable requirements from today's concerns.
  - Achieving simplicity gives you that much less to communicate about.
- **Feedback**
  - Change is inevitable, but change creates the need for feedback.
  - XP teams strive to generate as much feedback as they can handle as quickly as possible.

XP Values

- **Courage**
  - Doing something without regard for the consequences is not effective teamwork.
  - The courage to speak truths, pleasant or unpleasant, fosters communication and trust. The courage to discard failing solutions and seek new ones encourages simplicity.
  - The courage to seek real, concrete answers creates feedback.
- **Respect**
  - If members of a team don't care about each other and what they are doing, XP won't work.
  - If members of a team don't care about a project, nothing can save it.
- **Others**
  - Other important values include safety, security, predictability, and quality-of-life.

XP Principles

- **Humanity**
  - Basic safety: freedom from hunger, physical harm, and threats to loved ones. Fear of job loss threatens this need.
  - Accomplishment: the opportunity and ability to contribute to their society.
  - Belonging: the ability to identify with a group from which they receive validation and accountability and contribute to its shared goals.
  - Growth: the opportunity to expand their skills and perspective.
  - Intimacy: the ability to understand and be understood deeply by others.
- **Economics**
  - Time value of money: a dollar today is worth more than a dollar tomorrow.
  - Software development is more valuable when it earns money sooner and spends money later.
  - If I can redeploy my media scheduling program for a variety of scheduling-related tasks, it is much more valuable than if it can only be used for its originally intended purpose.

XP Principles

- **Mutual benefit**
  - It's about searching for practices that benefit me now, me later, and my customer as well.
  - Extensive internal documentation of software is an example of a practice that violates mutual benefit. I am supposed to slow down my development considerably so some unknown person in a potential future will have an easier time maintaining this code. I can see a possible benefit to the future person should the documentation still happen to be valid, but no benefit now.
  - XP solves like this:
    - I write automated tests that help me design and implement better today. I leave these tests for future programmers to use as well.
    - I carefully refactor to remove accidental complexity, giving me both satisfaction and fewer defects and making the code easier to understand for those who encounter it later.
    - I choose names from a coherent and explicit set of metaphors which speeds my development and makes the code clearer to new programmers.

XP Principles

- **Self-similarity**
  - Try copying the structure of one solution into a new context, even at different scales.
  - Just because you copy a structure that works in one context doesn't mean it will work in another. It is a good place to start, though.
- **Improvement**
  - To do the best you can today, striving for the awareness and understanding necessary to do better tomorrow.
  - The principle of improvement shows in practices that get an activity started right away but refine the results over time.
  - The history of software development technology shows us gradually eliminating wasted effort.
- **Diversity**
  - Teams where everyone is alike, while comfortable, are not effective.
  - Teams need to bring together a variety of skills, attitudes, and perspectives to see problems and pitfalls, to think of multiple ways to solve problems, and to implement the solutions.
  - Two ideas about a design present an opportunity, not a problem.

XP Principles

- **Reflection**
  - Good teams don't just do their work, they think about how they are working and why they are working.
  - They don't try to hide their mistakes, but expose them and learn from them.
  - Reflection moments:
    - Official: do pair programming and continuous integration.
    - Non-official: conversation with a spouse or friend, vacation, non-software-related reading and activities, shared meals and coffee breaks.
  - Reflection comes after action.
  - Learning is action reflected.
- **Flow**
  - Flow is delivering a steady flow of valuable software by engaging in all the activities of development simultaneously.
  - It suggests that for improvement, deploy smaller increments of value ever more frequently.

XP Principles

- **Opportunity**
  - To reach excellence, problems need to turn into opportunities for learning and improvement, not just survival.
  - Turning problems into opportunities takes place across the development process. It maximizes strengths and minimizes weaknesses.
- **Redundancy**
  - The critical, difficult problems in software development should be solved several different ways.
  - Defects are addressed in XP by many of the practices:
    - pair programming
    - continuous integration
    - sitting together
    - real customer involvement
    - daily deployment
  - While redundancy can be wasteful, be careful not to remove redundancy that serves a valid purpose.

XP Principles

- **Failure**
  - Failure is not a waste if it imparts knowledge.
  - Knowledge is valuable and sometimes hard to come by.
  - Failure may not be avoidable waste.
- **Quality**
  - Projects don't go faster by accepting lower quality.
  - Pushing quality higher often results in faster delivery.
  - Quality isn't a purely economic factor. People need to do work they are proud of.
  - A concern for quality is no excuse for inaction.
    - If you don't know a clean way to do a job that has to be done, do it the best way you can.
    - If you know a clean way but it would take too long, do the job as well as you have time for now. Resolve to finish doing it the clean way later.

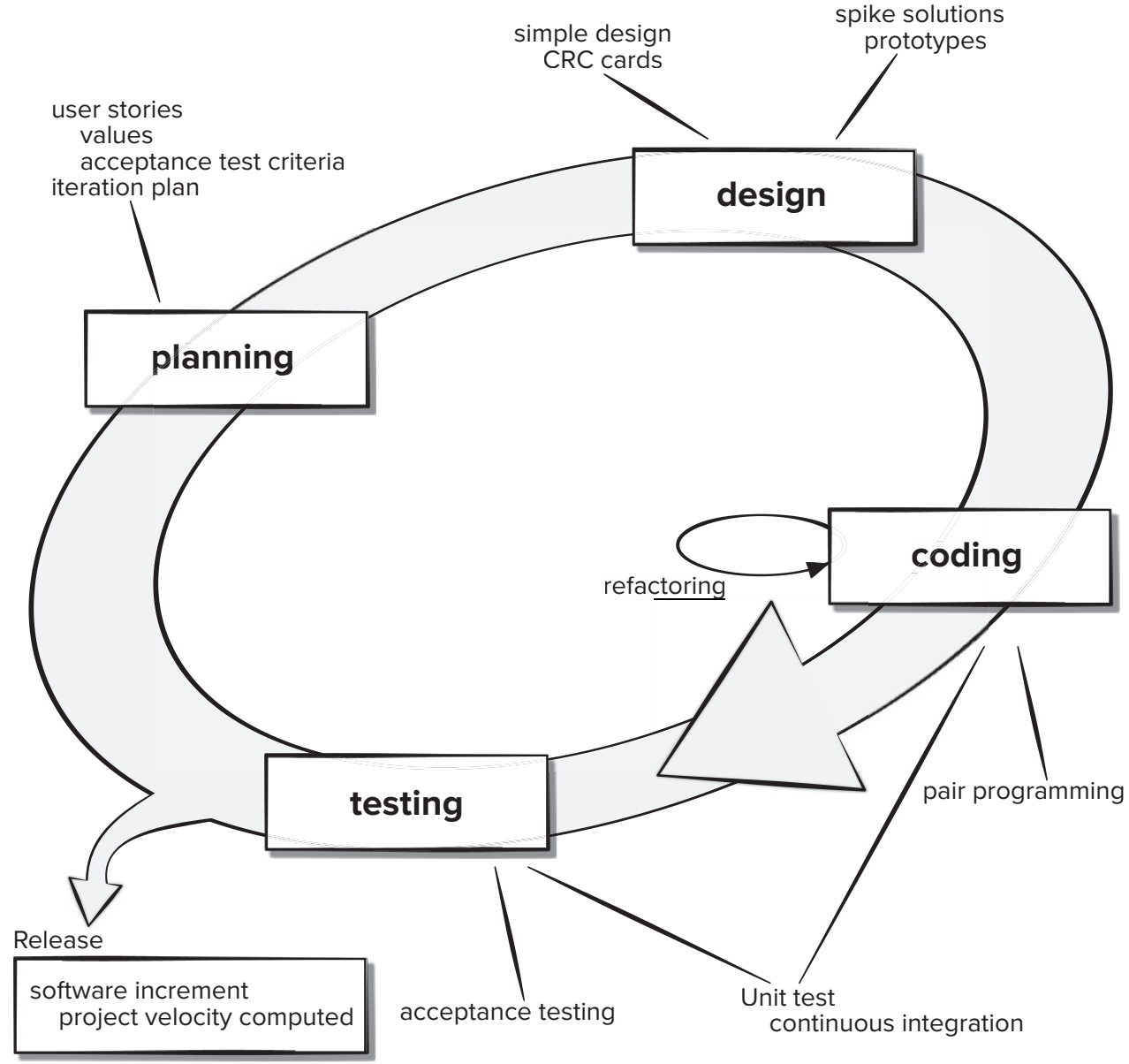
XP Principles

- **Baby steps**
  - It's always tempting to make big changes in big steps.
  - Baby steps acknowledge that the overhead of small steps is much less than when a team wastefully recoils from aborted big changes.
- **Accepted responsibility**
  - Responsibility cannot be assigned, it can only be accepted.
  - If someone tries to give you responsibility, only you can decide if you are responsible or if you aren't.
  - With responsibility comes authority.
  - Misalignments distort the team's communication.

XP Practices

- Primary Practices
  - Sit together
  - Whole team
  - Informative workspace
  - Energized work
  - Pair programming
  - Stories
  - Weekly cycle
  - Quarterly cycle
  - Slack
  - Ten-minute build
  - Continuous integration
  - Test-first programming
  - Incremental design
- Corollary Practices
  - Real customer involvement
  - Incremental deployment
  - Team continuity
  - Shrinking teams
  - Root-cause analysis
  - Shared code
  - Code and tests
  - Single code base
  - Daily deployment
  - Negotiated scope contract
  - Pay-per-use

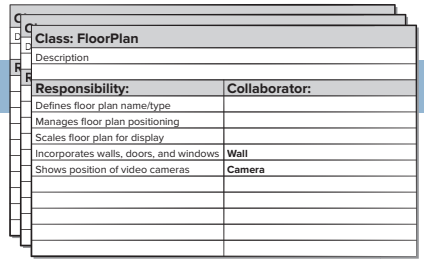
The Extreme Programming process



XP Practices in each activity

- XP Planning
  - Planning game
  - Listening leads to the creation of a set of “stories” (also called “user stories”)
  - The customer assigns a value (i.e., a priority)
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - Once a basic commitment is made for a release, the XP team orders the stories that will be developed in one of three ways:
    1. all stories will be implemented immediately (within a few weeks),
    2. the stories with highest value will be moved up in the schedule and implemented first, or
    3. the riskiest stories will be moved up in the schedule and implemented first.
  - After the first increment “project velocity” is used to help define subsequent delivery dates for other increments

XP Practices in each activity

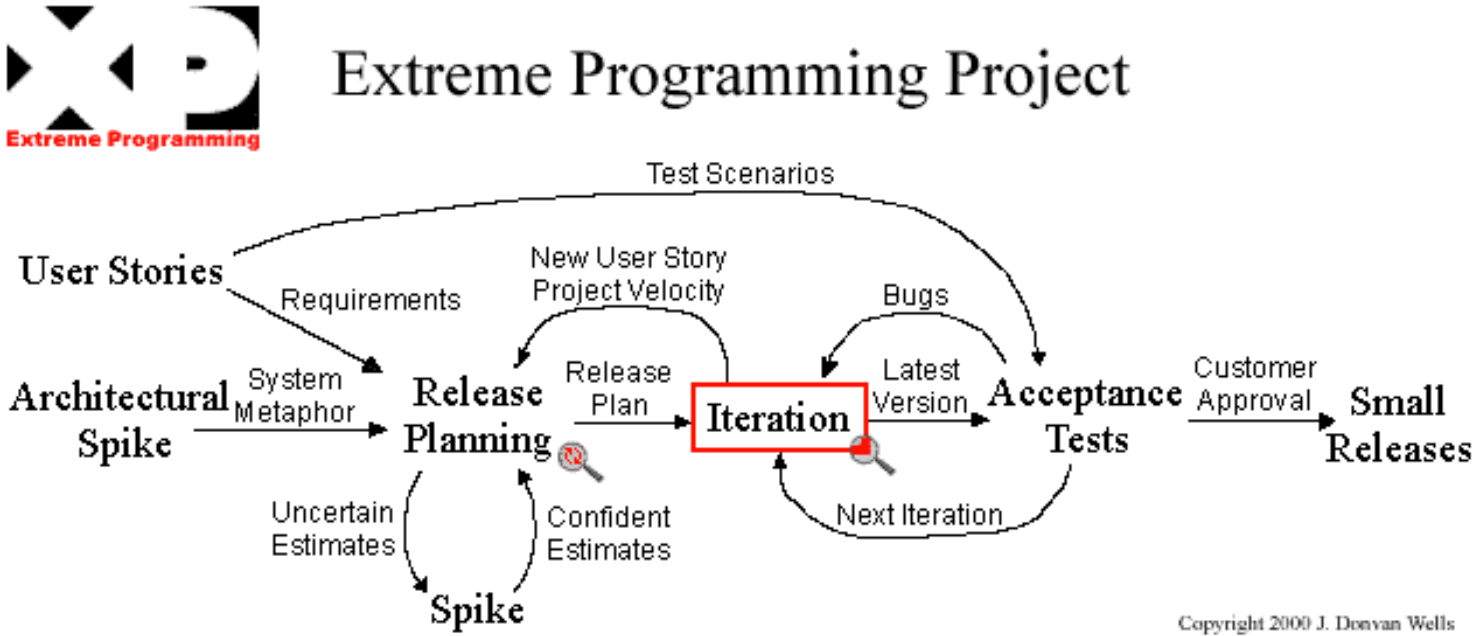


- XP Design
  - Follows the KISS (Keep It Simple, Stupid!) principle
  - Encourage the use of CRC (Class-Responsibility-Collaboration) cards
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype
  - Encourages "refactoring"—an iterative refinement of the internal program design
- XP Coding
  - Recommends the construction of a unit test for a store *before* coding commences
  - Encourages "pair programming"
  - "Continuous integration" strategy helps uncover compatibility and interfacing errors early.
- XP Testing
  - All unit tests are executed daily
  - acceptance tests, also called customer tests, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer.

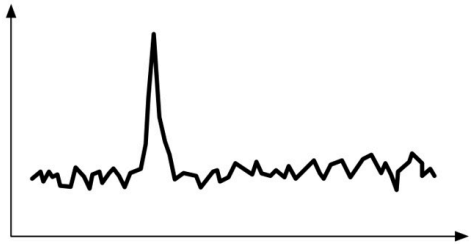
XP Practices in each activity

http://www.extremeprogramming.org/rules.html

- XP Managing (by Don Wells)
  - Give the team a dedicated open work space.
  - Set a sustainable pace.
  - A stand up meeting starts each day.
  - The Project Velocity is measured.
  - Move people around.
  - Fix XP when it breaks.



Spike



- a **spike** is the proof-of-concept type of activities, such as research design, investigation, exploration, and prototyping.
- Its purpose is to gain the knowledge necessary to reduce the risk and uncertainty of a technical approach, by learning about a feature, technology, or process.
- By doing this, it helps better understand a requirement, or increase the reliability of a *Story* estimate.
- A spike often occurs in a small time-boxed period designated to perform "the smallest implementation, which demonstrates plausible technical success."
- The spike typically takes between a few hours and a few days to complete, and codes written within this period is usually *thrown away* afterward, since it was built with nonproduction coding habits.
- The output of a **spike** story is an estimate for the original story.

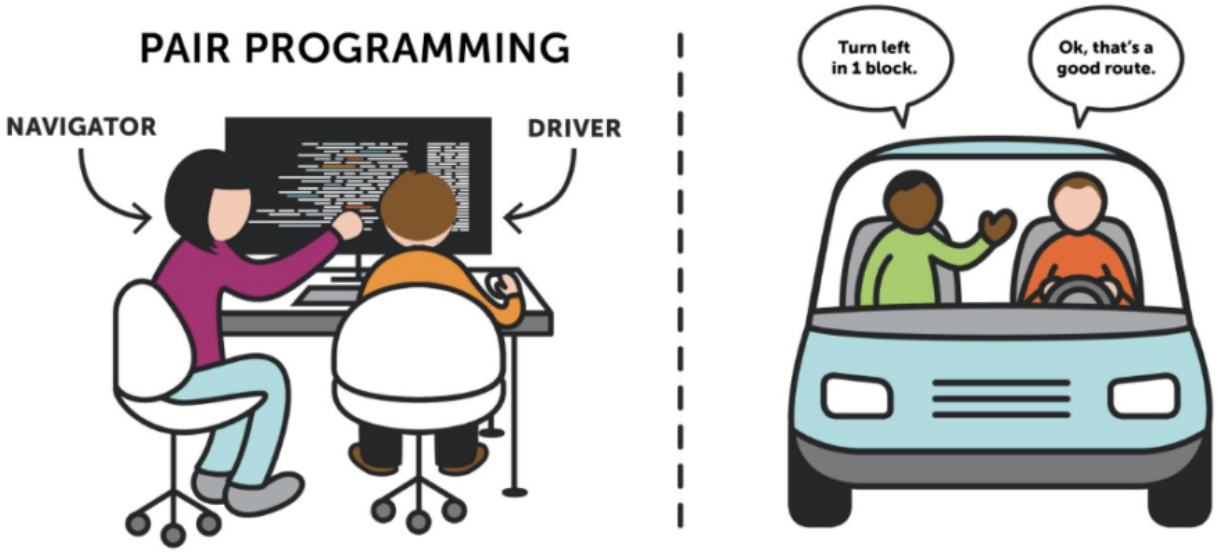
Pair Programming Style

- Pair programming is a dialog between two people simultaneously programming (and analyzing and designing and testing) and trying to program better.
- Pair programmers:
  - Keep each other on task.
  - Brainstorm refinements to the system.
  - Clarify ideas.
  - Take initiative when their partner is stuck, thus lowering frustration.
  - Hold each other accountable to the team's practices.

Metaphor

Williams and R. Kessler, *Pair Programming Illuminated*, Addison-Wesley, 2003.

- One programmer is the "**driver**" and the other the "**navigator.**" The driver controls the keyboard and focuses on the immediate task of coding, and the navigator acts as a reviewer, observing and thinking about more strategic architectural issues.



Pair Programming Style

- Driver/navigator style
- Unstructured style
- Ping-pong style
- Backseat Navigator Style

- These pairings include:
  - Expert - Expert
  - Expert - Novice
  - Novice - Novice

Test-Driven Development (TDD)

- TDD is a technique in which you write the tests *before* you write the code you want to test
- This seems backward, but it really does work better:
  - When tests are written first, you have a clearer idea what to do when you write the methods
  - Because the tests are written first, the code is necessarily written to be testable
  - Writing tests first encourages you to write simpler, single-purpose methods

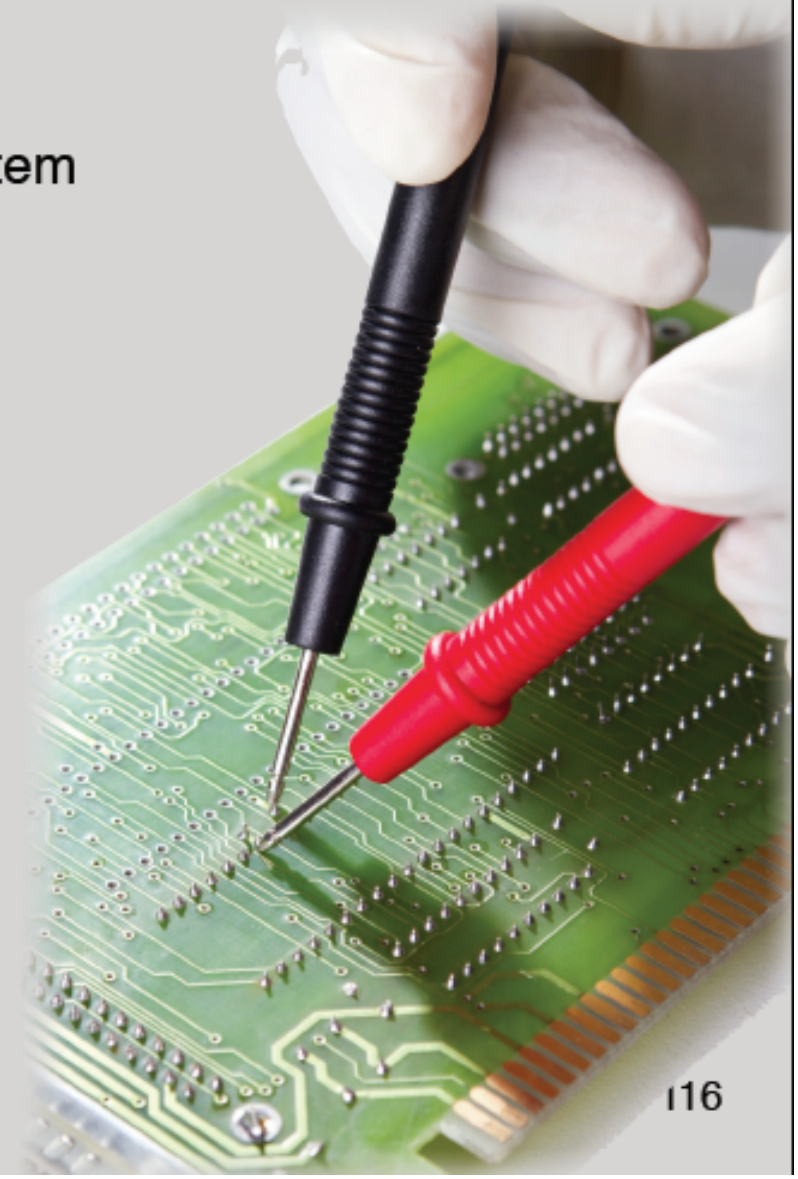
Continuous integration

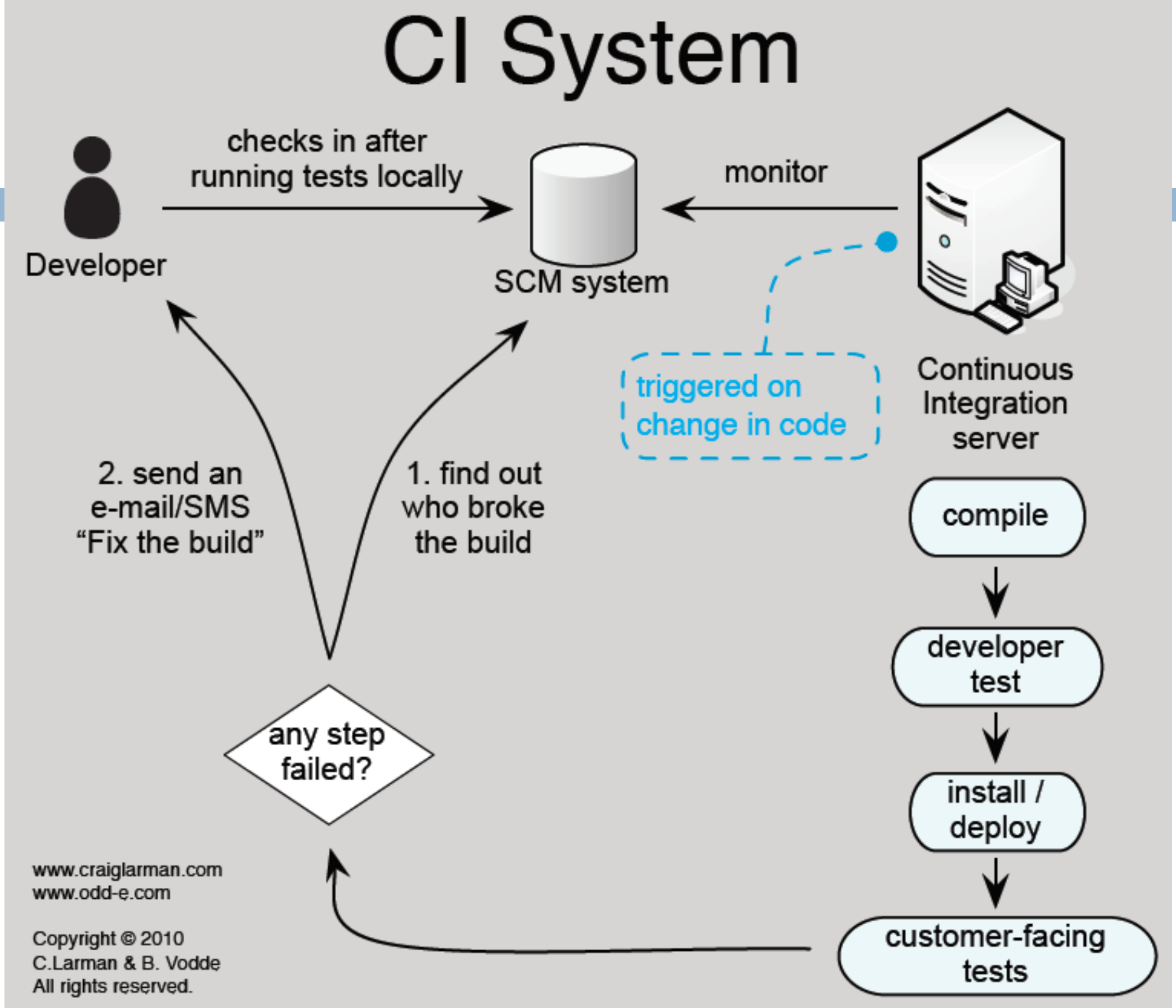
- Unit testing is testing a single, more-or-less independent part of a program in isolation
- Integration testing is testing the complete program with all its parts, to make certain they all work together
- Continuous integration is performing integration tests frequently (usually daily)
  - Continuous integration is more important on larger projects
  - For class projects, last-minute integration is still a bad idea

Continuous Integration

Continuous Integration is a **developer practice**
with the goal to always keep a **working system**
by making **small changes**, slowly growing the system
and **integrating** them at least **daily**
on the **mainline**
typically supported by a **CI system**
with lots of **automated tests**

- Increases transparency
- Increases cooperation and communication
- Enables people to work on same code




CI System
checks in after running tests locally
monitor
Developer
SCM system
triggered on change in code
Continuous Integration server
2. send an e-mail/SMS "Fix the build"
1. find out who broke the build
compile
developer test
install / deploy
customer-facing tests
any step failed?
www.craiglarman.com
www.odd-e.com
Copyright © 2010 C.Larman & B. Vodde All rights reserved.

Coding standards

- Coding standards make it simpler for your teammates (or yourself, weeks later) to read your code
- Java has some *very well-defined* conventions you should learn
  - Some conventions are strictly mechanical, such as formatting (spacing and indentation)
    - Eclipse can, upon request, correct your formatting for you
  - Some conventions require human intelligence
    - Use meaningful names, with the correct part of speech
    - Use adequate comments, written for the correct audience
    - Prefer clearly written code to "clever" code

https://www.cs.cornell.edu/courses/JavaAndDS/JavaStyle.html

https://peps.python.org/pep-0008/

Simple design

- "Do the simplest thing that can possibly work"
  - Complexity to achieve program efficiency is almost always a very bad investment
  - Complexity to support future features is seldom a good idea
- Program for the features you need *today*, not the ones you think you will want tomorrow
  - YAGNI -- "You ain't gonna need it"
- Follow the DRY principle: "Don't repeat yourself"
  - Don't have multiple copies of identical (or very similar) code
  - Don't have redundant copies of information

Lean

- Lean Manufacturing vs. Lean Software Development vs. Lean Startup

- **Taiichi Ohno** at Toyota brought a lot of ideas together in the **Toyota Production System** (TPS) which became the foundation of Lean Manufacturing.
- A key concept is to eliminate waste — the non-value-added components, and inefficiency in any processes of manufacturing operations
- The **lean** philosophy became very popular in manufacturing in the 1970s – 1980s

Five Key Principles of Lean Manufacturing

Lean Production, Lean Manufacturing, or the Toyota Production System

1. **Value.** Value is always defined by the customer's needs for a specific product.
2. **Value stream.** Once the value (end goal) has been determined, the next step is mapping the "value stream," or all the steps and processes involved in taking a specific product from raw materials and delivering the final product to the customer.

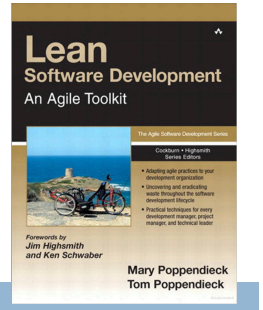
Five Key Principles

3. **Flow**. After the waste has been removed from the value stream, the next step is to be sure the remaining steps flow smoothly with no interruptions, delays, or bottlenecks.
4. **Pull**. With improved flow, time to market (or time to customer) can be dramatically improved by creating "**just in time**" manufacturing or delivery. Thus, products don't need to be built in advance or materials stockpiled, creating expensive inventory that needs to be managed, saving money for both the manufacturer and the customer.

Five Key Principles

5. **Perfection.** As gains continue to pile up, it is important to remember lean is not a static system and requires constant effort and vigilance to perfect.

Lean Software Development



- Lean Software Development is an Agile practice that is based on the principles of Lean Manufacturing.
- Lean Software Development is based on 7 Principles and 22 Tools detailed in the book.
- The fundamental principle of Lean Software Development is "Eliminate Waste", where waste is extra processes, defects, extra features, etc.
- According to the father of the TPS, Taiichi Ohno, **Waste** is defined as anything that does not produce value for the customer.
- Lean Software Development emphasizes delivering value to the customer, eliminating waste, and continuous improvement.

Seven Lean Software Development Principles

1. Eliminate Waste
2. Amplify Learning
3. Decide as Late as Possible
4. Deliver as Fast as Possible
5. Empower the Team
6. Build Integrity In
7. See the Whole

Seven Lean Software Development Principles

1. Eliminate Waste
   - If a component is sitting on a shelf gathering dust, that is waste.
   - If a development cycle has collected requirements in a book gathering dust, that is waste.
   - If a manufacturing plant makes more stuff than is immediately needed, that is waste.
   - If developers code **more features than** are immediately **needed**, that is waste.
   - In manufacturing, moving product around is waste.
2. Amplify Learning
   - Development is an exercise in discovery, while production is an exercise in reducing variation.
   - The best approach to improving a software development environment is to amplify learning.

Seven Lean Software Development Principles

3. Decide as Late as Possible
   - Development practices that provide for late decision making are effective in domains that involve uncertainty, because they provide an options-based approach.
4. Deliver as Fast as Possible
   - Without speed, you cannot delay decisions.
   - Without speed, you do not have reliable **feedback**.
   - In development, the discovery cycle is critical for learning:
     - Design, implement, feedback, improve.
   - The shorter these cycles are, the more can be learned.

Seven Lean Software Development Principles

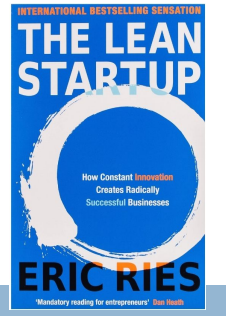
5. Empower the Team
   - Involving developers in the details of technical decisions is fundamental to achieving excellence.
   - Because decisions are made late and execution is fast, it is not possible for a central authority to orchestrate activities of workers.
   - Thus, lean practices use **pull techniques** to schedule work and contain local signaling mechanisms so workers can let each other know what needs to be done.
   - In lean software development, the **pull mechanism** is an agreement to deliver increasingly refined versions of working software at regular intervals.

Seven Lean Software Development Principles

6. Build Integrity In
   - Conceptual integrity means that the system's central concepts work together as a smooth, cohesive whole, and it is a critical factor in creating perceived integrity.
   - Software with integrity has a coherent architecture, scores high on usability and fitness for purpose, and is maintainable, adaptable, and extensible.
7. See the Whole
   - When individuals or organizations are measured on their specialized contribution rather than overall performance, suboptimization is likely to result.

Lean Startup

https://theleanstartup.com/principles



- Lean Startup applies Lean principles to entrepreneurship and startup development.
- Concepts:
  - Eliminate Uncertainty
  - Work Smarter, Not Harder
  - Develop an MVP
  - Validated Learning

Lean Startup: Principles

https://theleanstartup.com/principles

1. Entrepreneurs are Everywhere
2. Entrepreneurship is Management
3. Validated Learning
4. Innovation Accounting
5. Build-Measure-Learn

Build Me A Pyramid

http://www.informationweek.com/two-ways-to-build-a-pyramid/d/d-id/1012280?

Year 0 – Builder 1

Builder 1

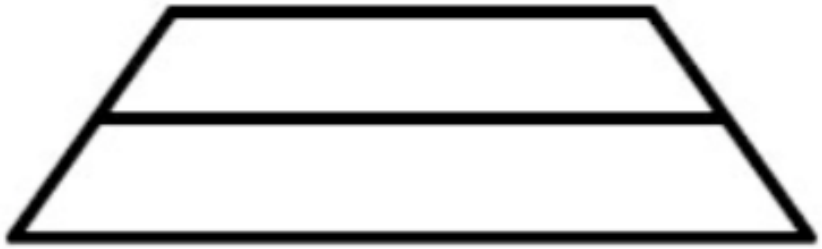
Build Me A Pyramid

Year 1 - Builder 1

Builder 1

Build Me A Pyramid

Year 2 - Builder 1



Builder 1

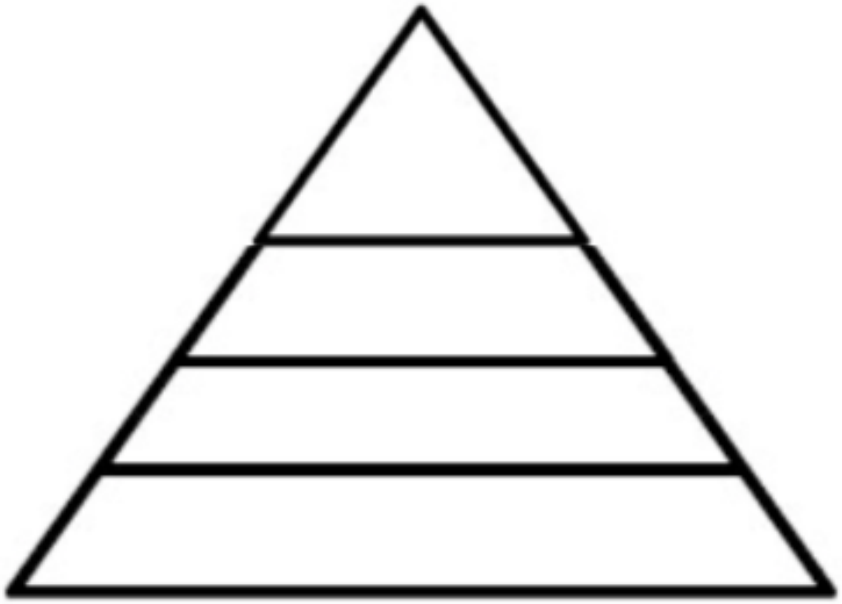
Build Me A Pyramid

Year 3 Builder 1

Builder 1

Build Me A Pyramid

Year 4 Builder 1



Builder 1

Build Me A Pyramid

Year 1 - Builder 2

Builder 2

Build Me A Pyramid

Year 2 - Builder 2

Builder 2

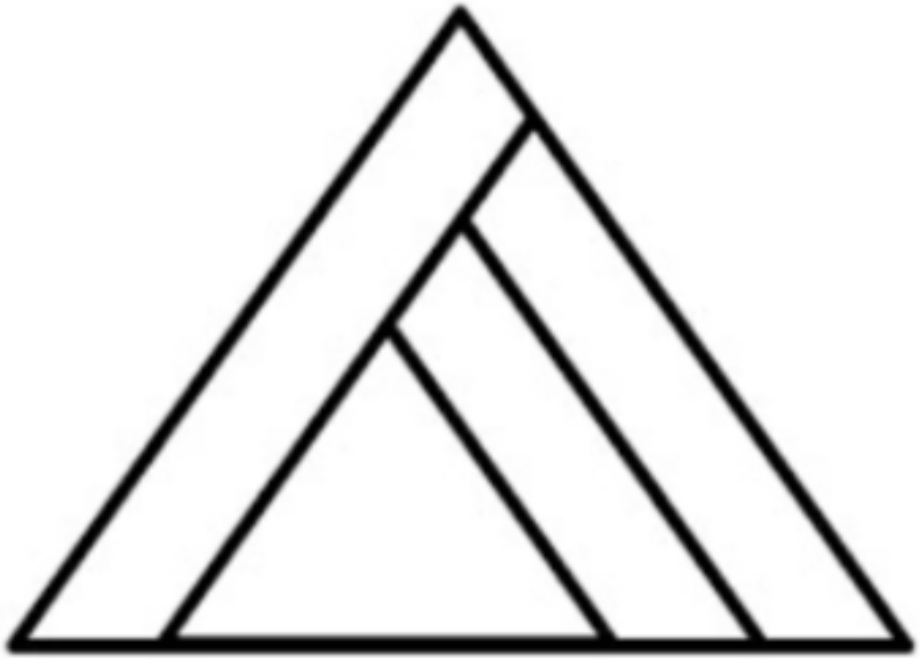
Build Me A Pyramid

Year 3 - Builder 2

Builder 2

Build Me A Pyramid

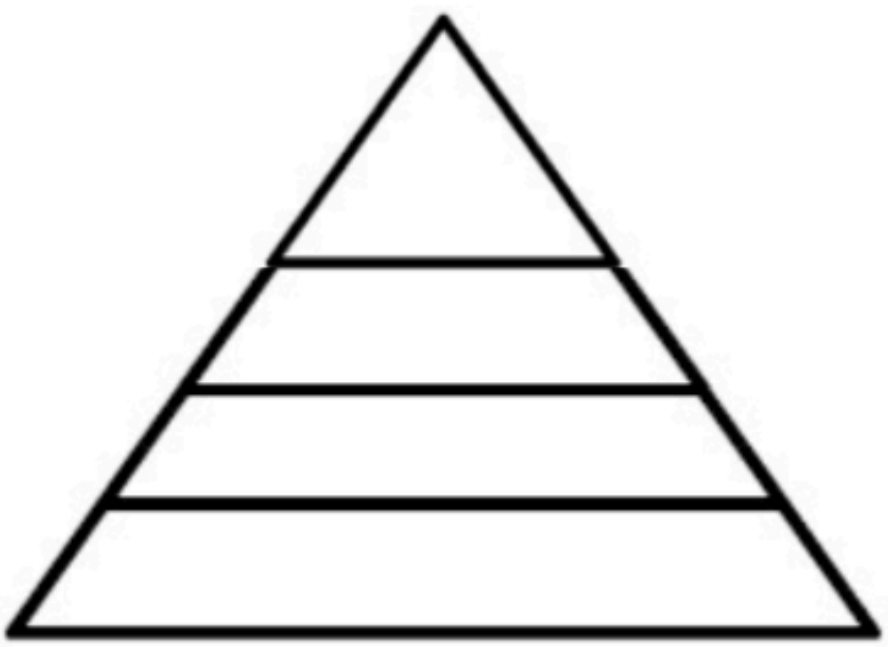
Year 4 - Builder 2



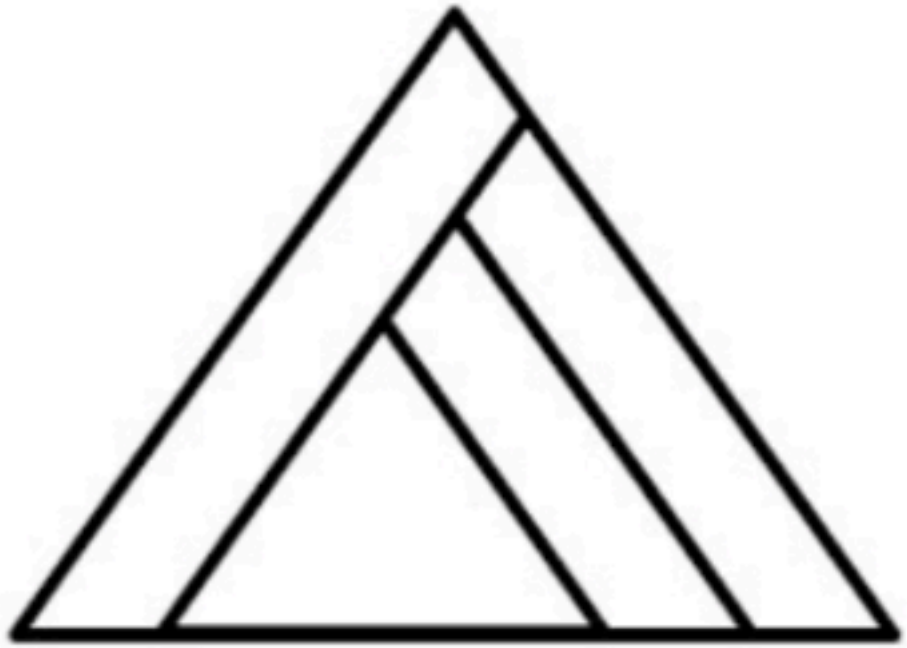
Builder 2

Build Me A Pyramid

Year 4 – comparison the pyramids of two diff. builders



Builder 1



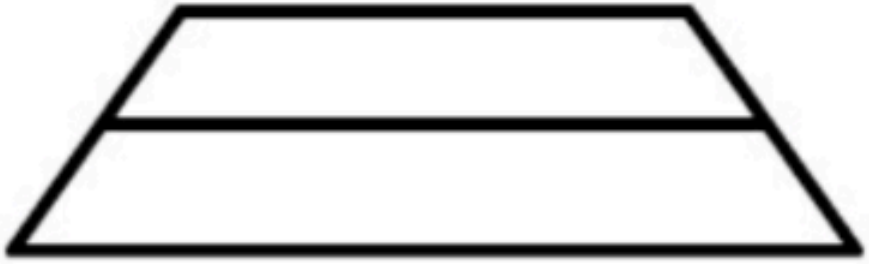
Builder 2

Build Me A Pyramid

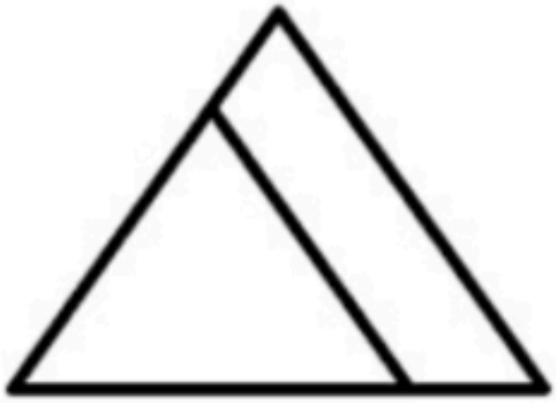
What if the pharaoh had died after year 2?

Build Me A Pyramid

What if the pharaoh had died after year 2?



Builder 1



Builder 2

Minimum Viable Product

http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html

- The term was coined and defined in 2001 by **Frank Robinson,** but used a lot and popularized by **Steve Blank** and **Eric Ries**

"A **minimum viable product** (MVP) has just those core features that allow the product to be deployed, and no more."

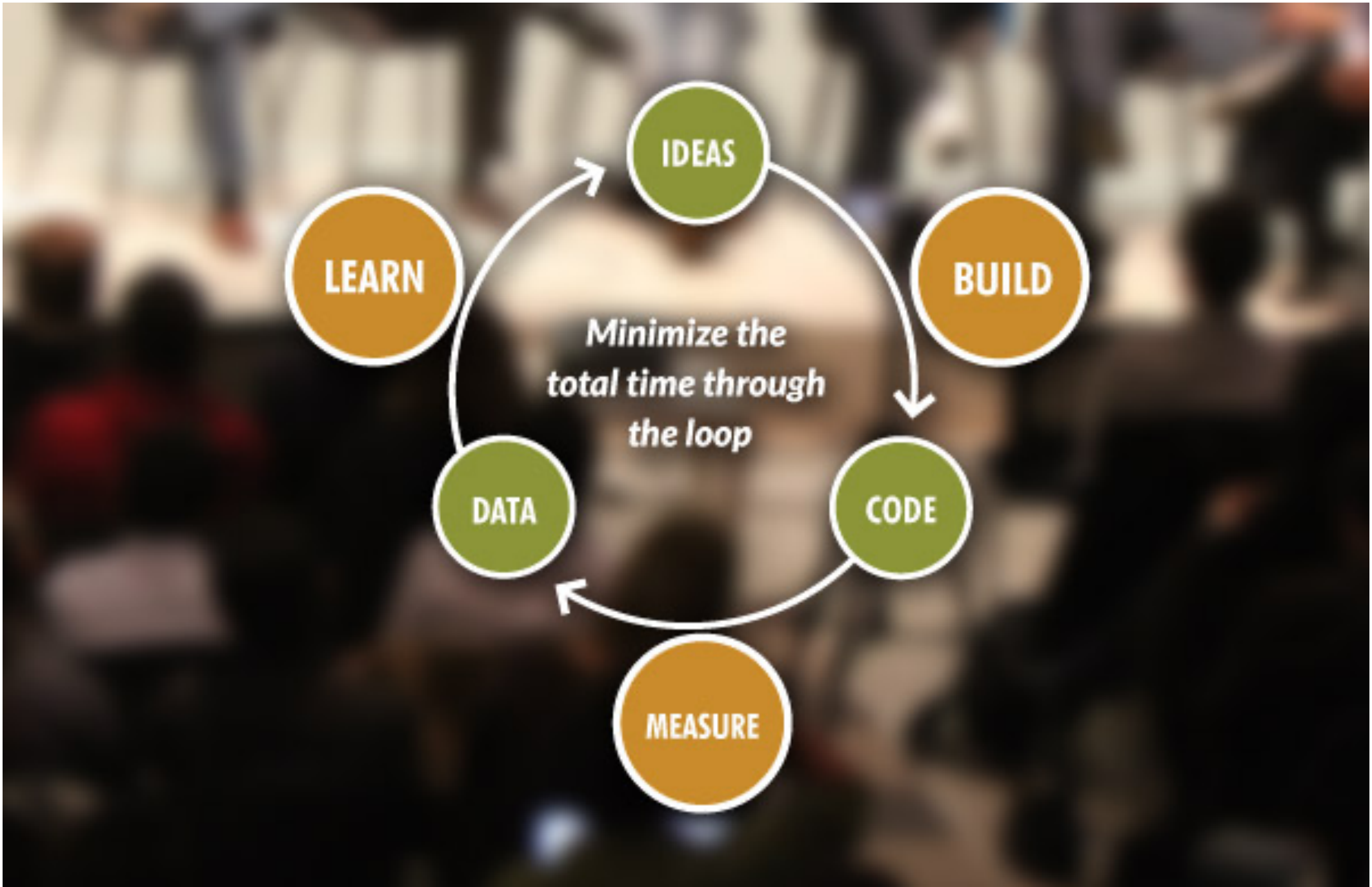
- Eric Reis

Minimum Viable Product

http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html

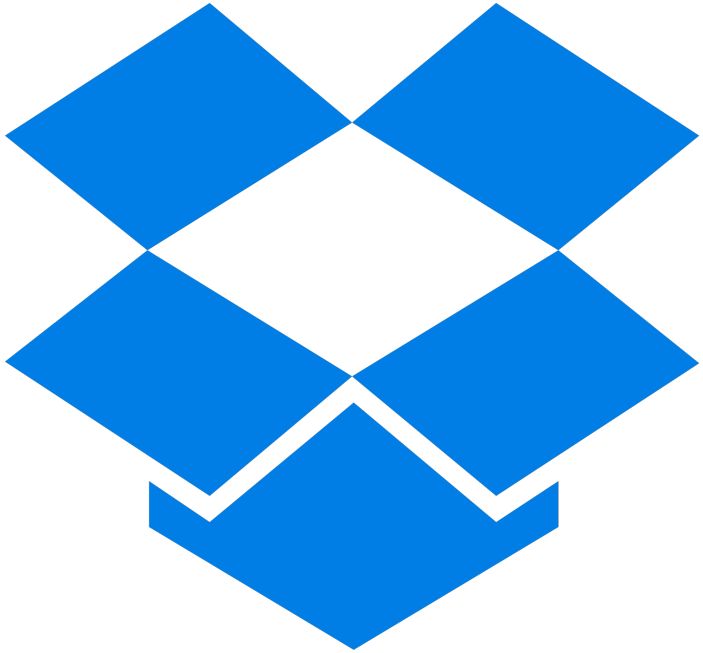
- The minimum viable product sequence is not just a sequence of bad versions of the product
- Each version is defined to answer a specific question
- The **Build - Measure - Learn** cycle is the key to using the MVP effectively

Build-Measure-Learn



Famous MVPs: Dropbox

https://techcrunch.com/2011/10/19/dropbox-minimal-viable-product/



One of the techniques Drew Houston used to validate the concept for Dropbox is so powerful – and so simple – that most entrepreneurs overlook it. It's an example of building a minimum viable product (MVP). This is later called the Dropbox MVP, in Drew's honor.

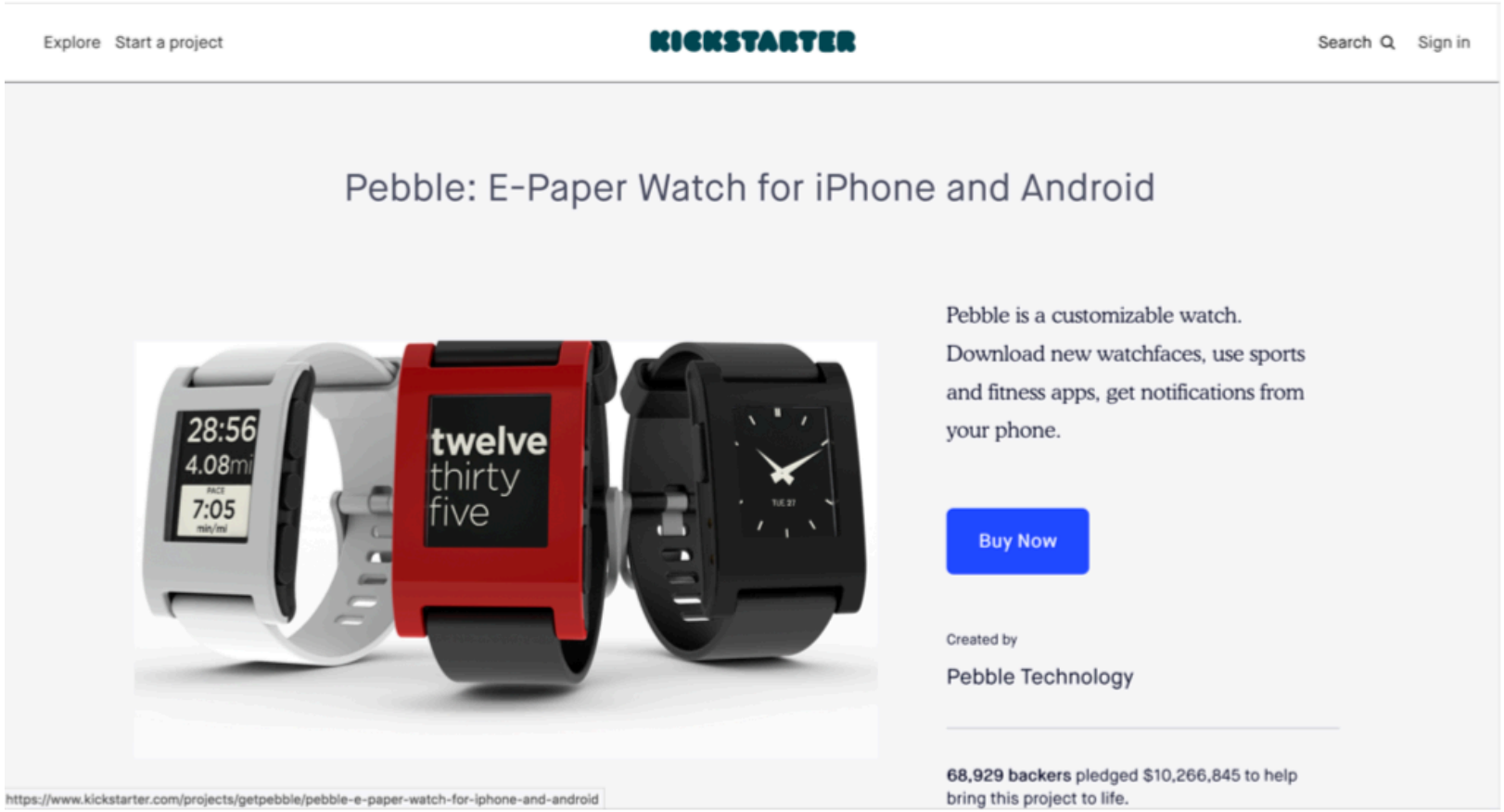
https://youtu.be/7QmCUDHpNzE?si=cDLt5xlhZQt_LDXH

Famous MVPs: KickStarter

https://www.kickstarter.com/projects/getpebble/pebble-e-paper-watch-for-iphone-and-android

Use KickStarter as a way to validate a startup idea

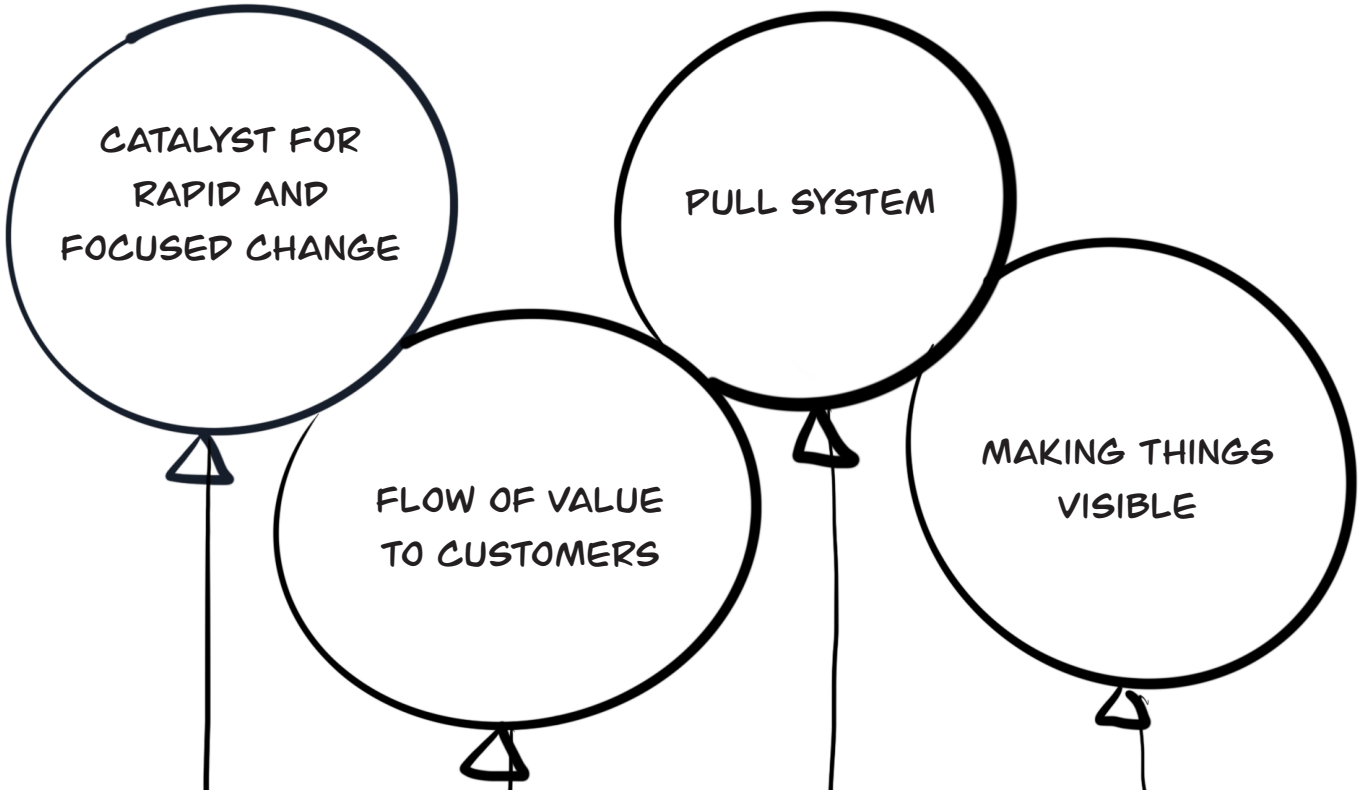
KickStarter: Collecting revenue *before* building



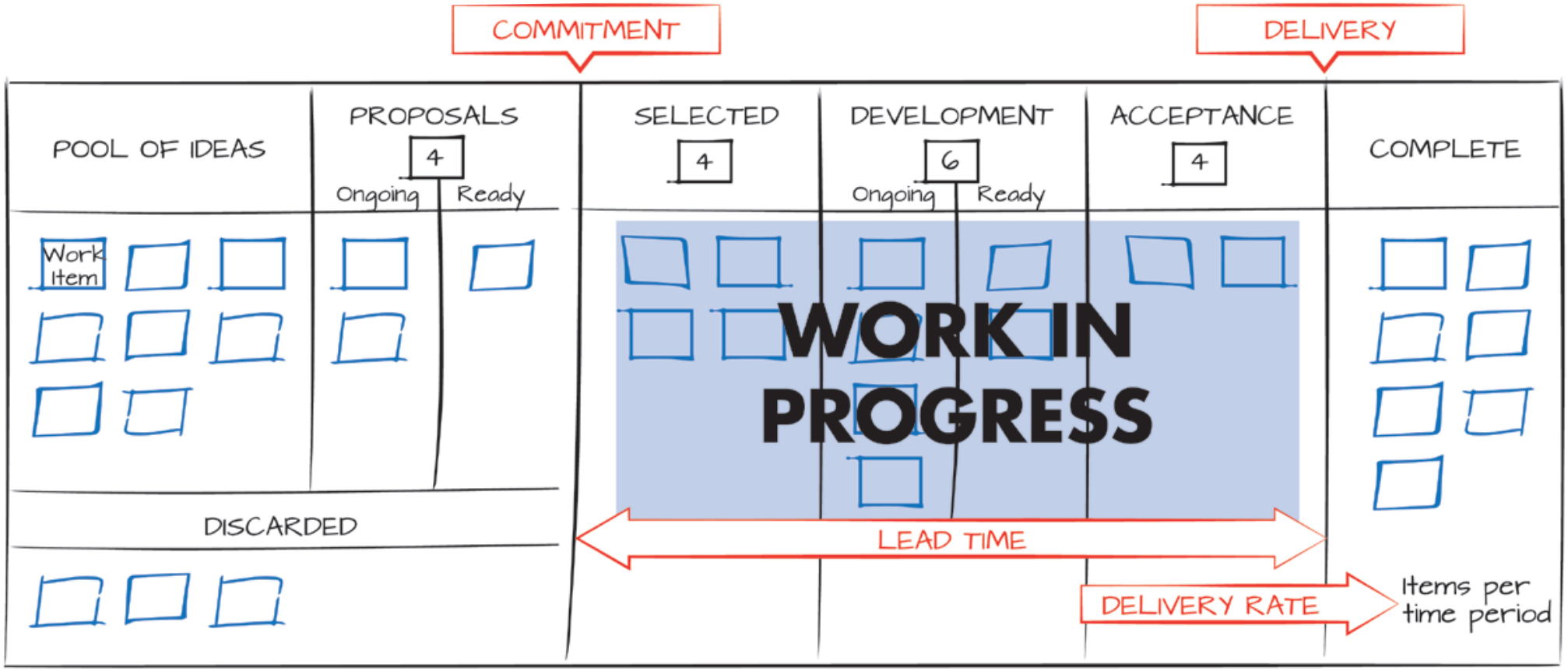
"A Crowdfunding campaign is not a way to test an MVP, it is an MVP in its own right."

Kanban

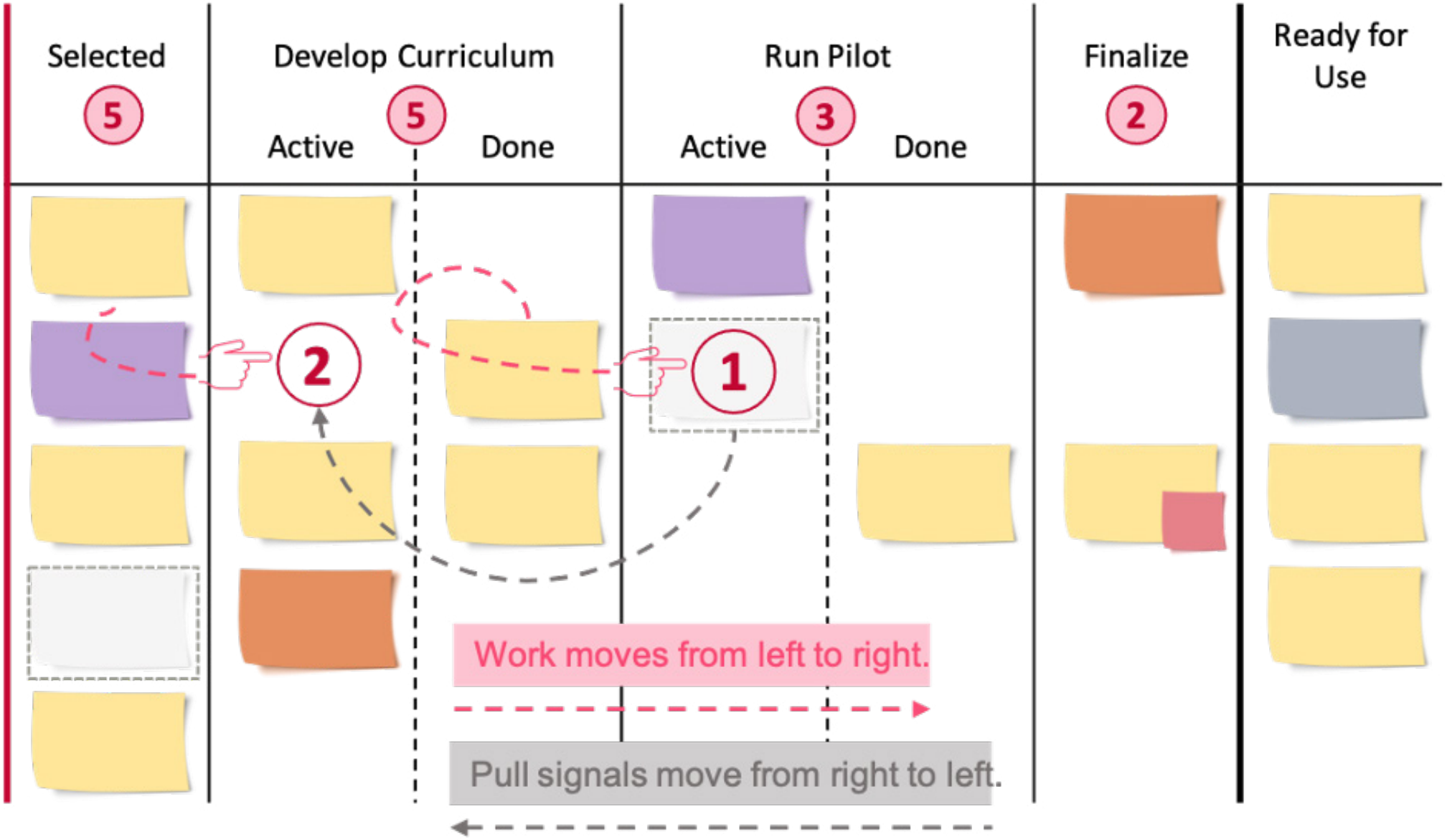
- The Kanban method is a lean methodology that describes methods for improving any process or workflow
- Kanban is a visual management method for optimizing workflow.



Kanban Board



Kanban Board



Practices

VISUALIZE

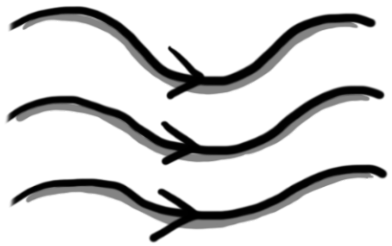
- SHOW WORK & ITS FLOW
- VISUALIZE RISKS
- KANBAN BOARD
- WORK IN PROGRESS LIMITS

LIMIT WORK IN PROGRESS

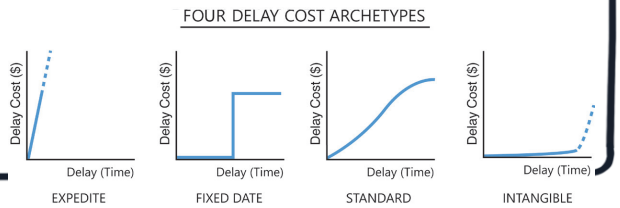
- PULL SYSTEM
- RESPECTED
- INSPECT + ADAPT
- FOCUS

Practices

MANAGE FLOW



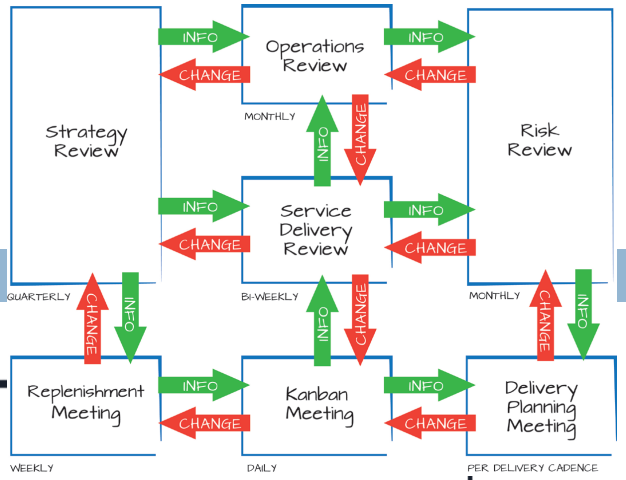
- MAXIMIZE VALUE DELIVERED
- SYSTEM PREDICTABILITY
- SERVICE LEVEL AGREEMENT
- CLASSES OF SERVICES
- COST OF DELAY



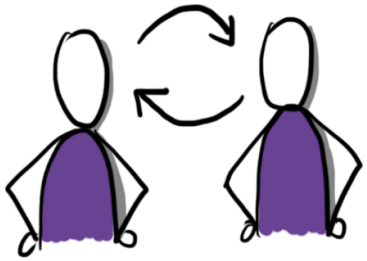
EXPLICIT POLICIES

- DEFINE THE FLOW
- DEFINE THE WHOLE PROCESS
- EVOLUTION
- RESPECT THE RULES
- TRANSPARENCY
- CONSTRAINS & CONTROL DECISIONS

Practices



FEEDBACK LOOPS



- ENABLE EVOLUTIONARY CHANGE
- INFORMATION EXCHANGE
- OBSERVE + CHANGE
- INSPECT + ADAPT
- 7 CADENCES

IMPROVE COLLABORATIVELY EVOLVE EXPERIMENTALLY

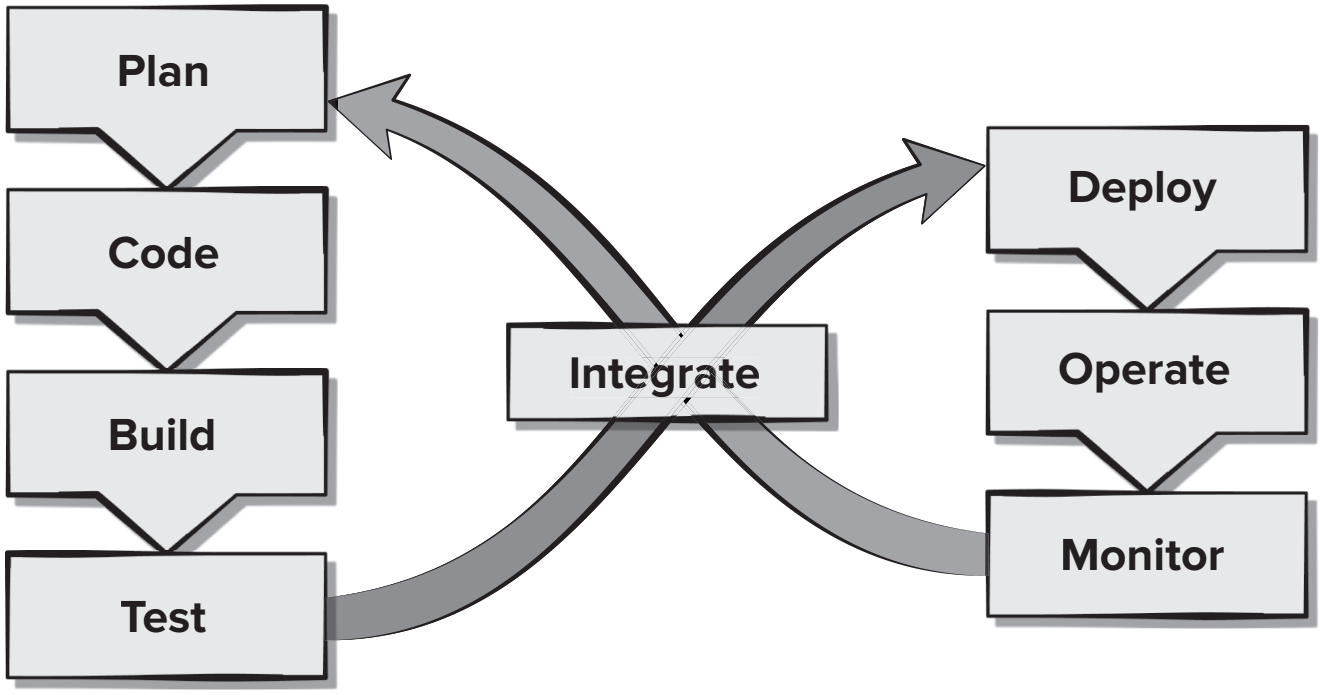
- EMPIRICAL OBSERVATIONS
- DATA-DRIVEN EXPERIMENTS
- USE MODELS
- DEVELOP HYPOTHESES
- INTRODUCE CHANGE
- OBSERVE & MEASURE THE RESULTS

Kanban Metrics

- **Lead time** is the time it takes for a single work item to pass through the system from the start (commitment point) to completion
- **Delivery rate** is the number of completed work items per unit of time, such as features per week, training classes per months, or new hires per month
- **WIP** (work in progress) is the amount of work items in the system (or a defined part of it) at a certain point in time

DevOps

- DevOps was created by Patrick DeBois to combine Development and Operations.
- DevOps attempts to apply agile and lean development principles across the entire software supply chain.



Lean in DevOps

- Principles and practices started here
  - Deliver more value with less waste
  - Strong focus on respect for people

Brief History of DevOps

2007

Patric Debios

2008

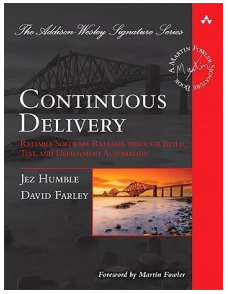
Agile Conference

2009

- 10+ Deploy per Day
- First DevOpsDays

2010

Continuous Delivery

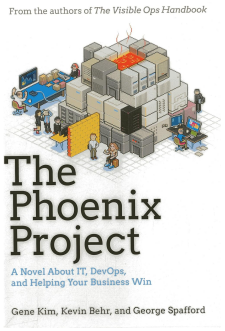


2014 STATE OF DEVOPS REPORT

DORA
DEVOPS RESEARCH & ASSESSMENT

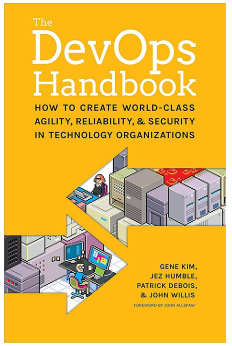
2013

The Phoenix Project



2016

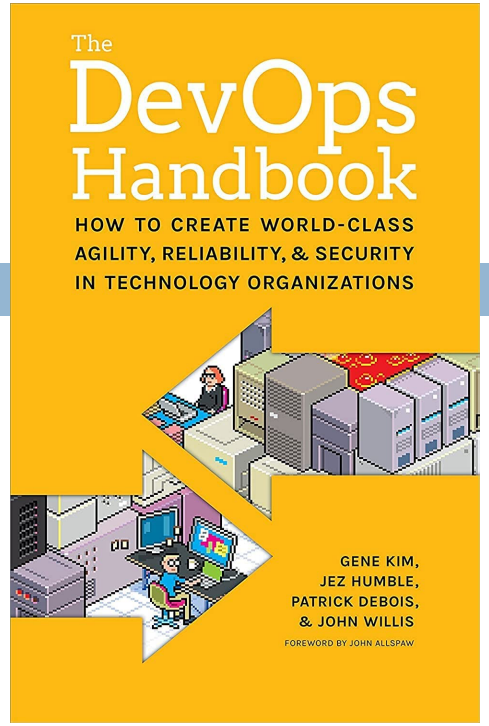
The DevOps Handbook



2019

40 DevOpsDays

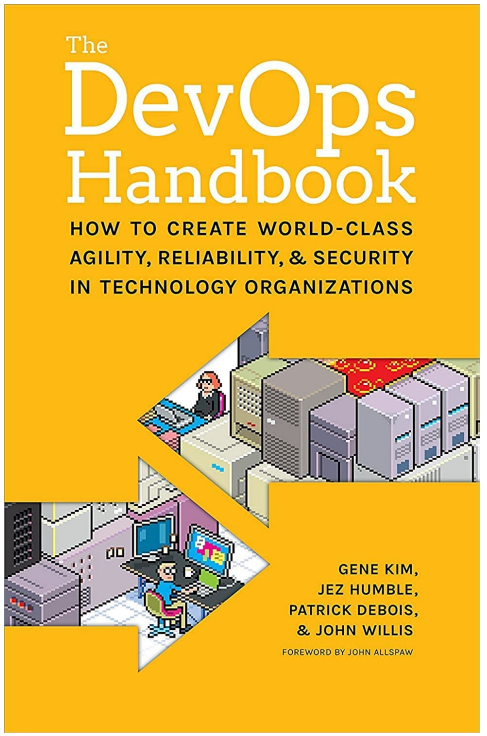
Definition



- John Willis, an author of The DevOps Handbook;
  - “Unfortunately, DevOps means whatever the definer wants you to believe and no definition is wrong.”
  - “DevOps is about service as a supply chain in all the things that enable fast, resident, and consumable delivery of the service.”
  - "DevOps is about humans. DevOps is a set of practices and patterns that turn human capital into high-performance organizational capital.”

~~Definition~~

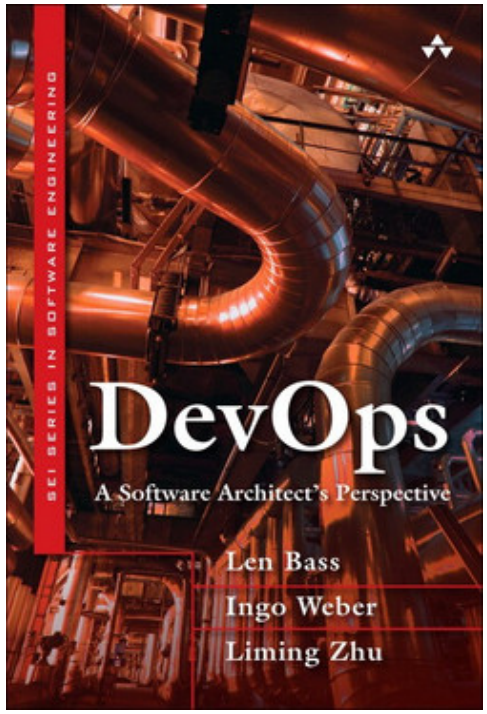
- Gene Kim, another author of The DevOps Handbook;
  - “DevOps should be defined by the outcomes.
    It is those sets of cultural norms and technology practices that enable the fast flow of planned work from, among other things, development through tests into operations, while preserving world class reliability, operation, and security.
    DevOps is not about what you do, but what your outcomes are.
    So many things that we associate with DevOps, such as communication and culture, fit underneath this very broad umbrella of beliefs and practices.”



  - "DevOps is the emerging professional movement that advocates a collaborative working relationship between Development and IT Operations, resulting in the fast flow of planned work (i.e., high deploy rates), while simultaneously increasing the reliability, stability, resilience and security of the production environment."

~~Definition~~ 1

- “A set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality"



Keys DevOps Acronyms

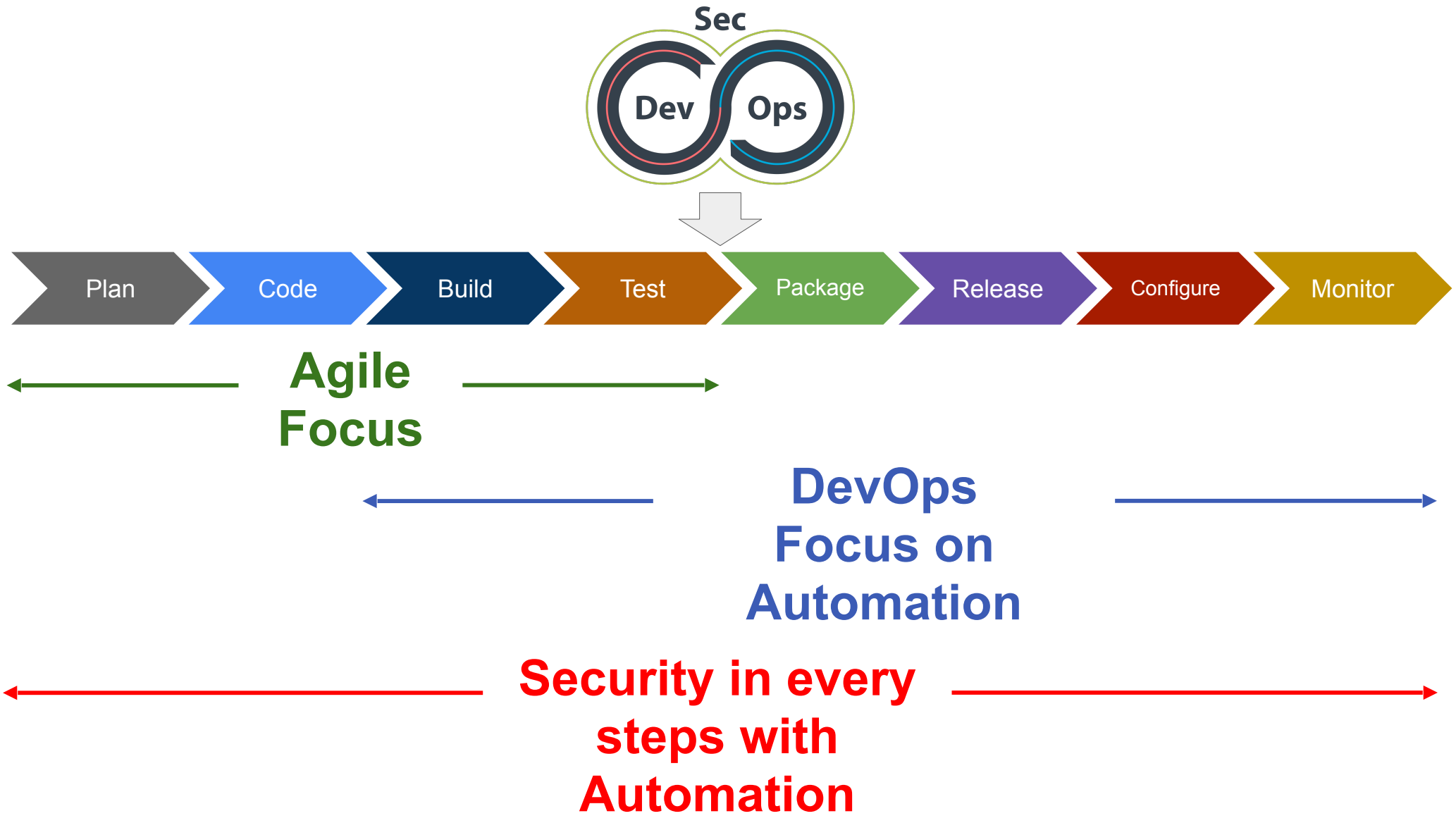
- **CAMS**
  - **C**ulture
  - **A**utomation
  - **M**easurement
  - **S**haring

— Damon Edwards and John Willis —

- **CALMS**
  - **C**ulture
  - **A**utomation
  - **L**ean
  - **M**easurement
  - **S**haring

— Jez Humble —

DevSecOps and Agile



DevOps Principles

- The Three Ways
  - brings CALMS to life
- The Seven Principles and Seven Wastes of Lean
- Improvement Kata
  - Comes from Toyota production system
  - A process for continuous improvement
- A3 Problem Solving Framework

The Three Ways

- Developed by Gene Kim and Mike Orzen
  1. Systems thinking
  2. Amplifying feedback loops
  3. A culture of continuous experimentation and learning

The Three Ways

1. Systems thinking
   - emphasizes the performance of the entire system as opposed to the performance of a specific silo work group or team.

The Three Ways

2. Amplifying feedback loops

- A feedback loop is a process that allows for reflection on its own output before determining the next steps that need to be completed.
- Build automated tests into the pipeline
- Embed operations engineers into the development teams.
- e.g., Dashboard visible for all to see

The Three Ways

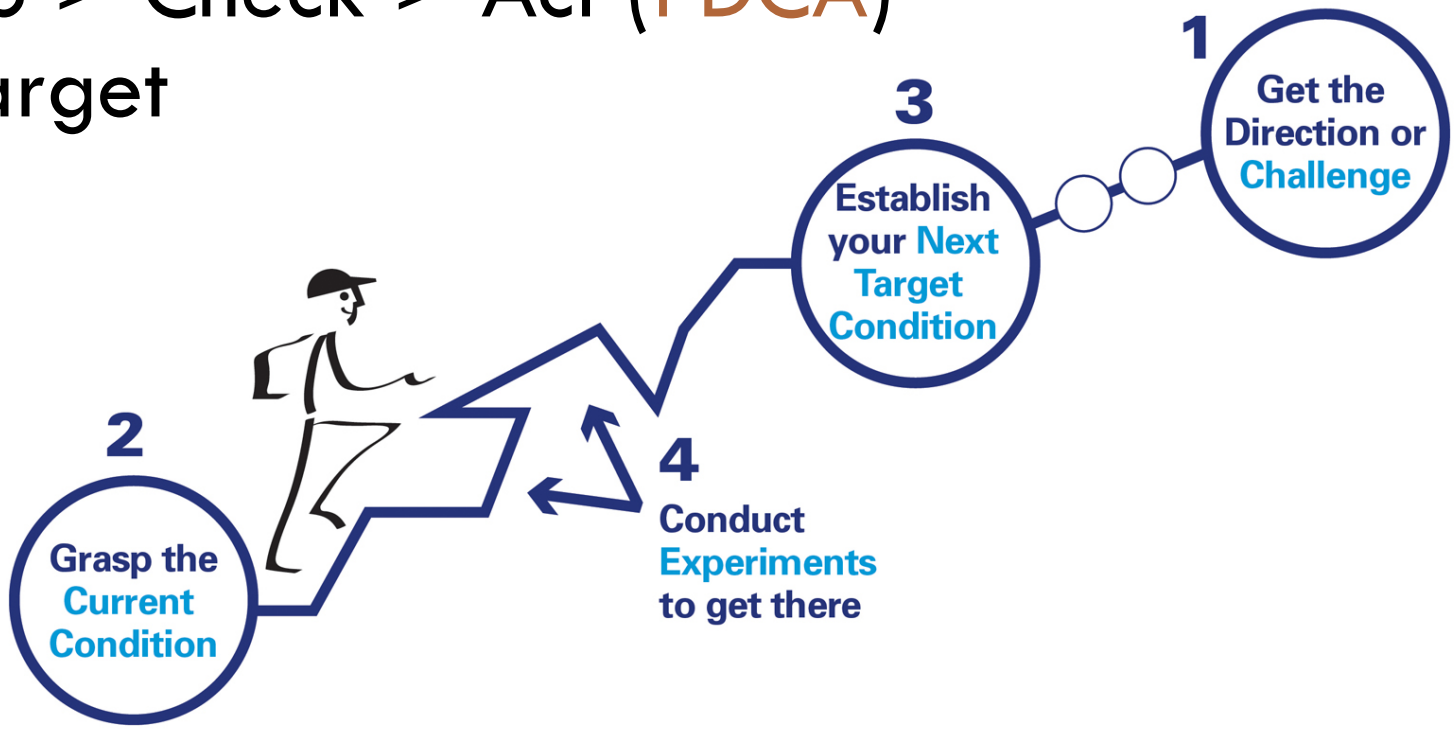
3. A culture of continuous experimentation and learning
   - Create that culture!
   - Encourage risk-taking & failing forward
   - Affirm that repetition in practice is a prerequisite to mastery

Seven Wastes of Lean

1. Partially Done Work
2. Extra Features
3. Revisiting Decisions
4. Handoffs
5. Delays
6. Task Switching
7. Defects

Improvement Kata Steps

1. Understand vision & direction for project
2. Analyze to understand current condition
3. Establish target condition
4. Plan > Do > Check > Act (PDCA) toward target



A3 Problem Solving Framework

A3 Steps	PDCA Cycle
Background	Plan
Problem Statement/Current Condition	
Goal/Target Condition	
Analysis	
Proposed Countermeasures	Do
Implementation Plan	Check
Follow Up Actions	Act

- ✓ Problem description
- ✓ Background
- ✓ Current situation
- ✓ Root cause analysis
- ✓ Target condition
- ✓ Action plan and countermeasures
- ✓ Follow-up plan with a clear implementation plan

- Step 0: Identify a problem or need
- Step 1: Conduct research to understand the current situaion
- Step 2: Conduct root cause analysis
- Step 3: Devise countermeasures to address root causes
- Step 4: Develop a target state
- Step 5: Create an implementation plan
- Step 6: Develop a follow-up plan with predicted outcomes
- Step 7: Discuss plans with all affected parties
- Step 8: Obtain approval for implementation
- Step 9: Implement plans
- Step 10: Evaluate the results

Organizational Cultures

Pathological	Bureaucratic	Generative
Power-oriented	Rule-oriented	Performance-oriented
Low cooperation	Modest cooperation	High cooperation
Messengers shot	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure leads to scapegoating	Failure leads to justice	Failure leads to inquiry
Novelty crushed	Novelty leads to problems	Novelty implemented

DevOps

- The DevOps approach involves several stages that loop continuously until the desired product exists:
  - Continuous development
  - Continuous testing
  - Continuous integration.
  - Continuous deployment
  - Continuous monitoring