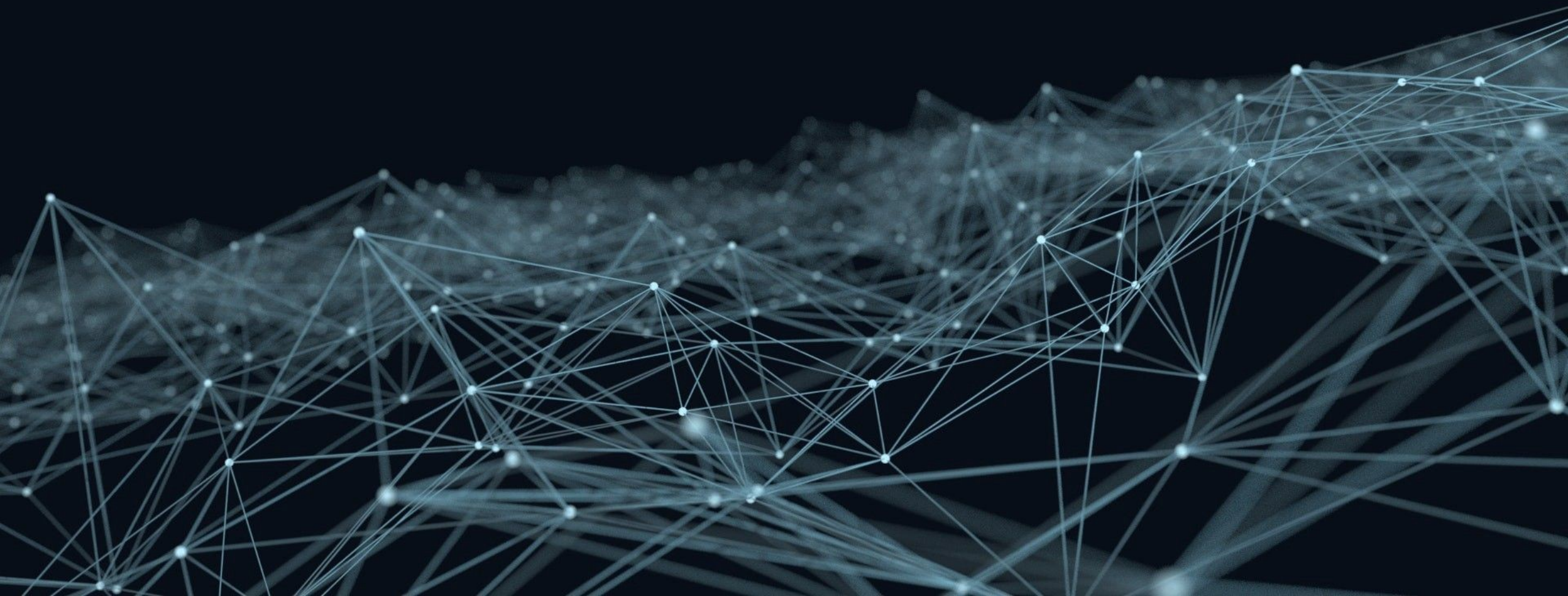
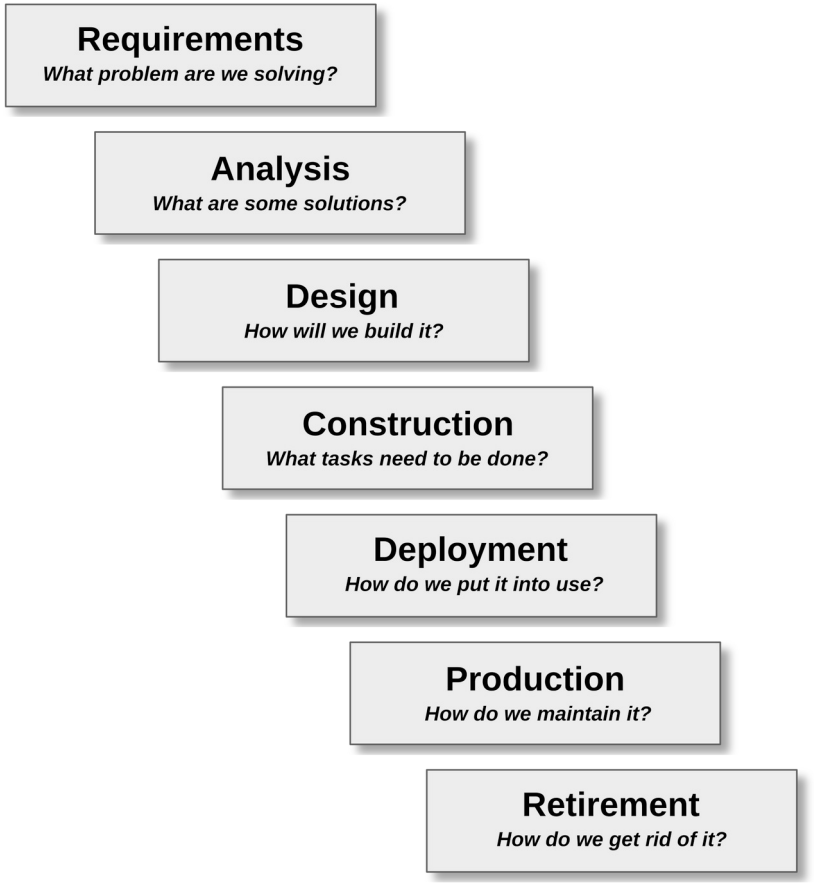
Agile and Scrum

1. Development Processes



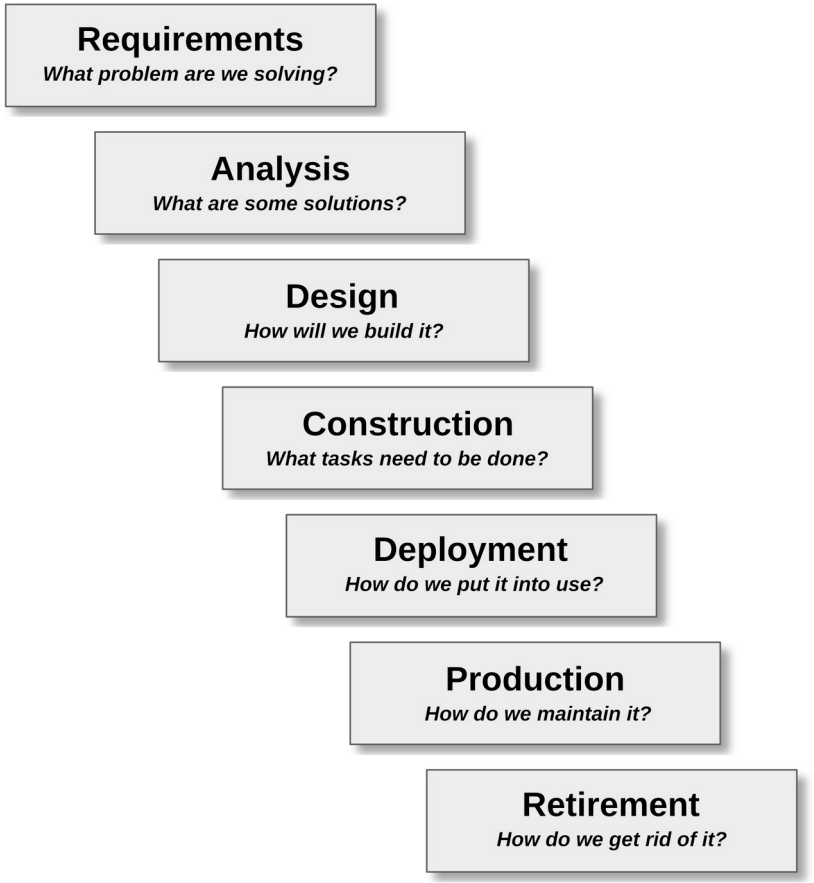
The Engineering Cycle

- Used whenever a product or service is engineered
- Series of ordered logical steps
  - Each step builds on the previous one
- This is *not* a development process
- Different types of processes apply the cycle differently
- The cycle describes the phases in the lifecycle of the product or service
  - Similar to the Application Lifecycle Model



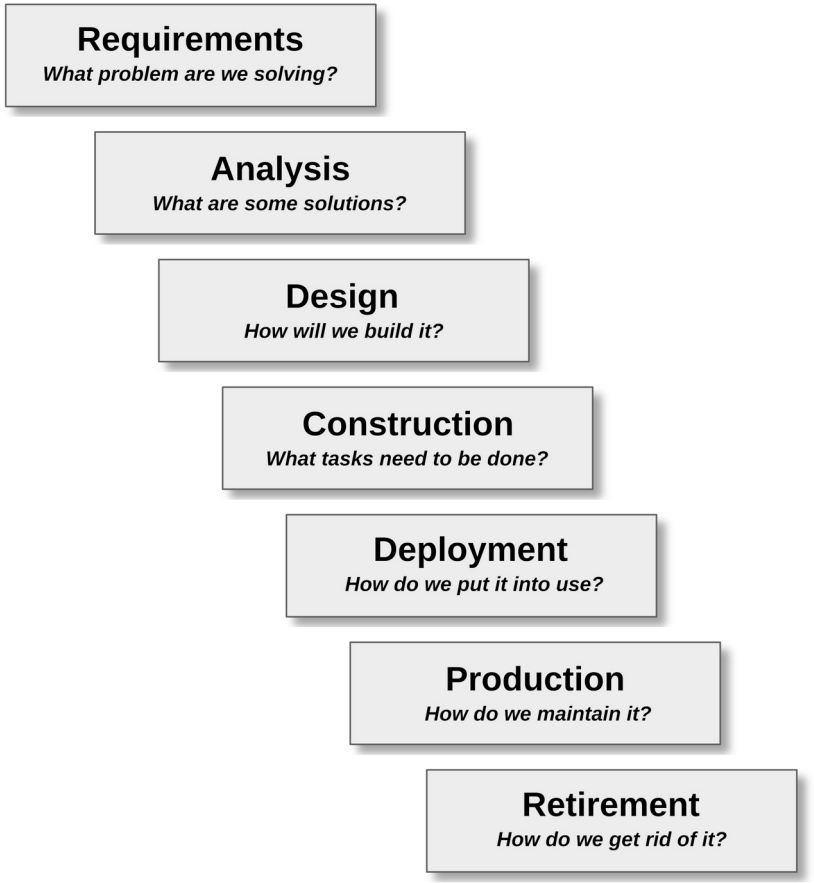
Requirements

- Identify the problem to be solved
  - What is needed by stakeholders
  - What factors constrain solutions
- Acceptance criteria
  - How will we know the problem is solved?
- What are the specific properties a solution should have to satisfy the requirements
  - KPI – key performance indicators
- The type of process we use often depends on how much of the requirements we have at the start



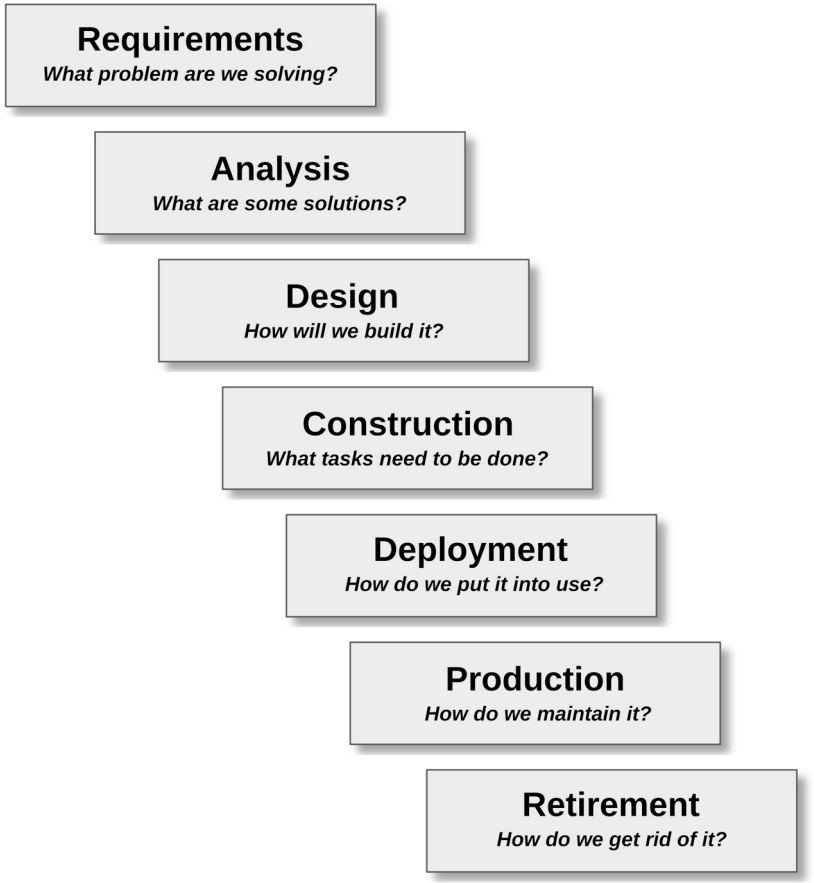
Analysis

- Evaluate possible solutions
  - What would a solution look like?
  - How would it perform?
  - How would it be organized?
  - How would it be tested?
- Evaluate possible choices
  - Risk factors
  - Compare to similar sorts of solutions
- At some point we have to chose one solution to develop
  - In some cases we may need to abandon a solution that is not working
  - In this case we need to plan for re-engineering the solution



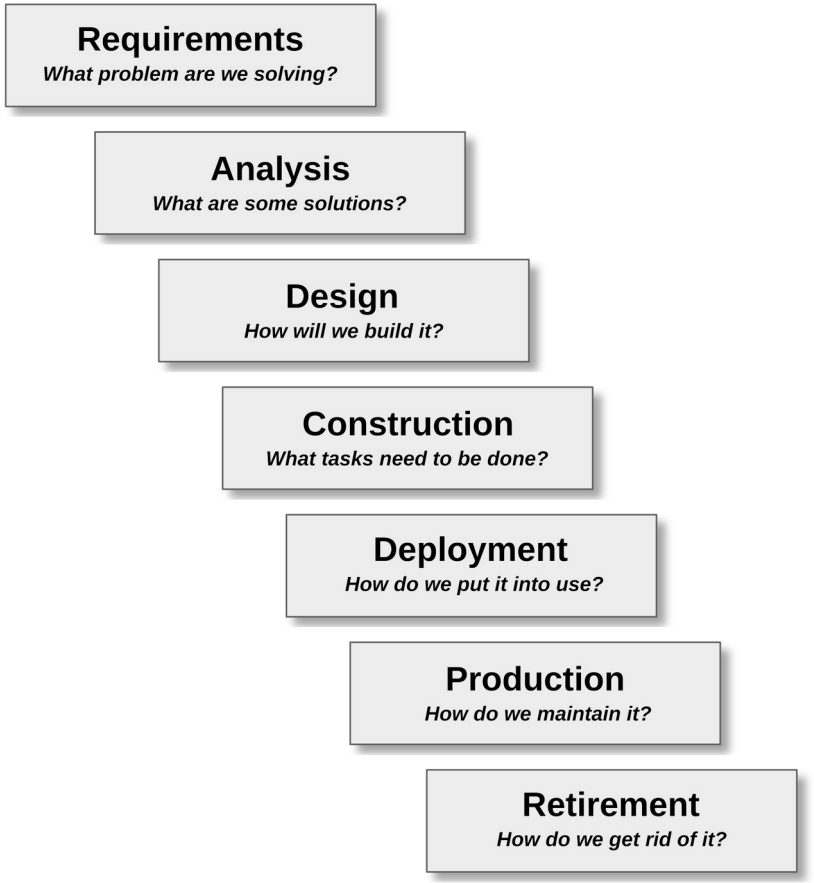
Design

- Only one solution can be built
- What is possible with the available resources?
  - Budget and time constraints
  - Staffing and equipment
  - Other resources
- Design the architecture
  - How is the solution structured
  - Can the architecture and solution evolve
  - What are the components of the solution
  - Choose technologies
- Define the plan for construction phase
  - Project management



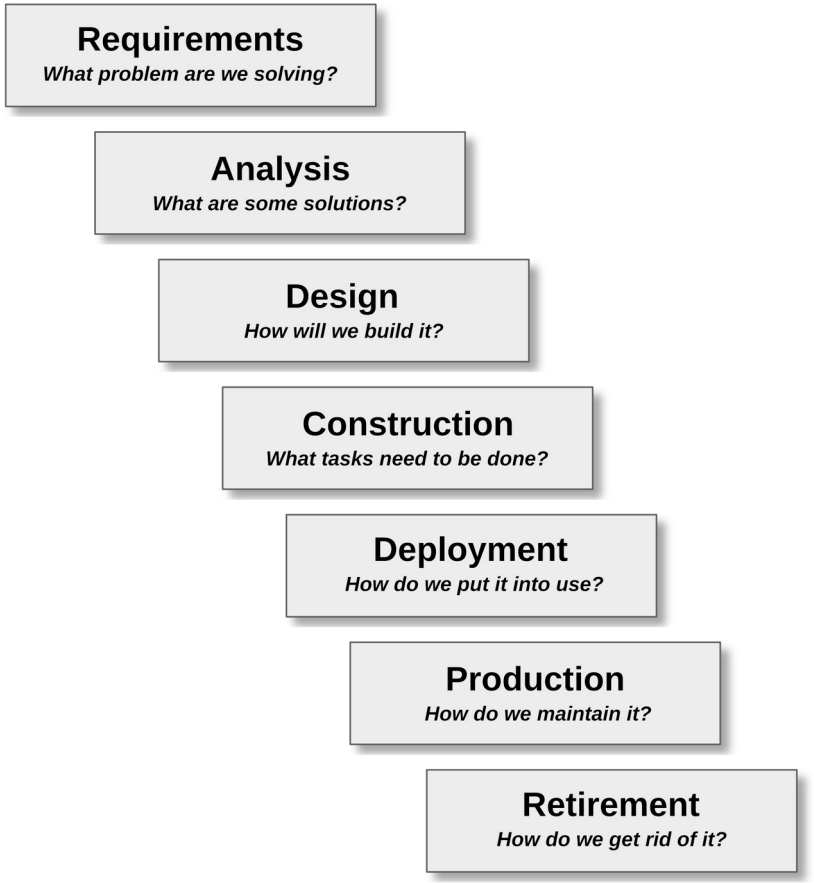
Construction

- Apply a development process
  - Roles and responsibilities
  - Testing, coding, etc.
- Define the methodology
  - Agile/Scrum, etc.
- Phase that consumes most of the resources
- The goal of the previous steps
  - To reduce the amount of errors made in construction
  - Reduce the amount of rework



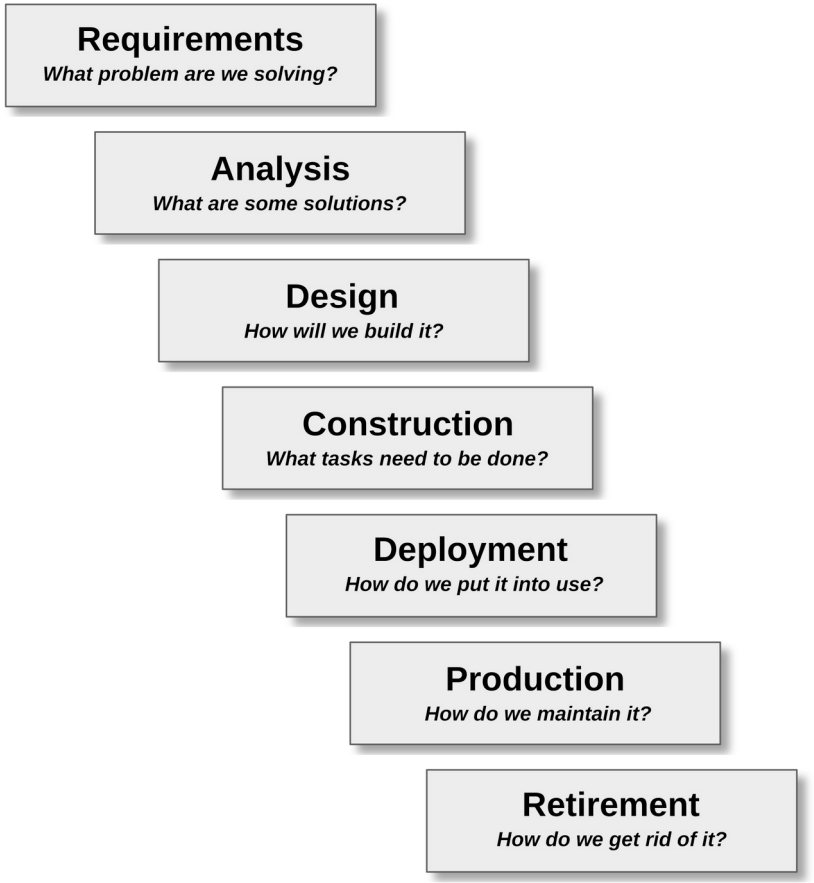
Deployment

- Get the product into production
- How to deliver to the users
- Acceptance testing
  - Test marketing
  - Beta testing and feedback
  - Are we meeting the KPI
- May require the deployment of a support infrastructure
  - Training of users
  - Support systems
  - Training of support staff
- Failed deployment may require re-engineering the solution



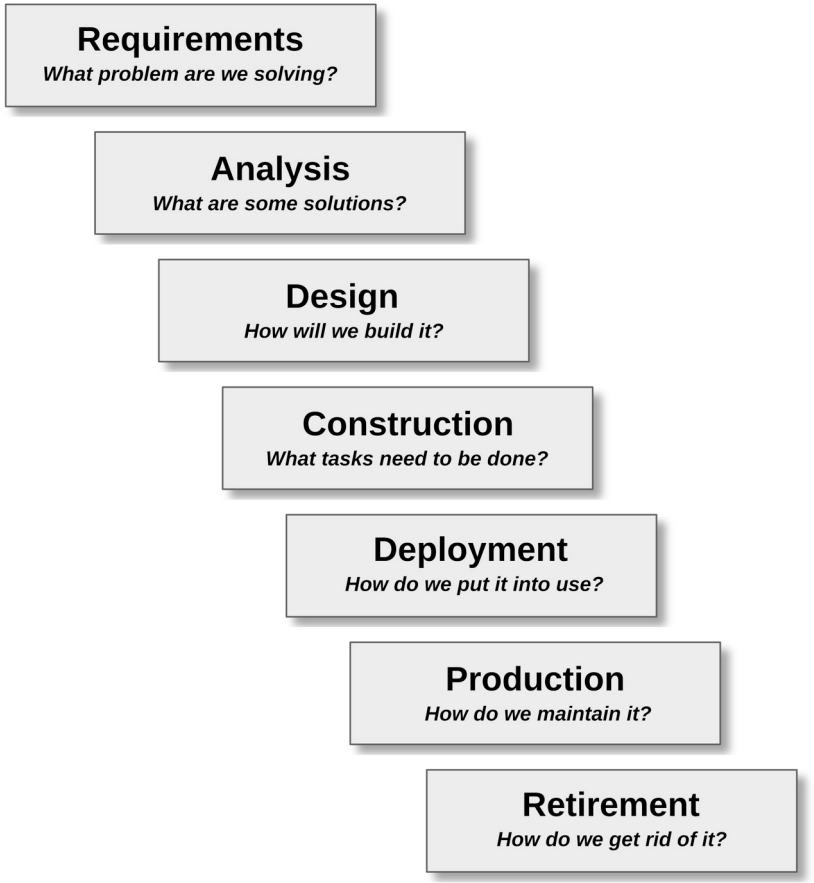
Production

- Monitor performance
- Manage changing requirements
- Collect data for future development
- Evaluate service level agreements
- Look for “gotchas” in the real world
- Adapt deployment to changing loads and traffic



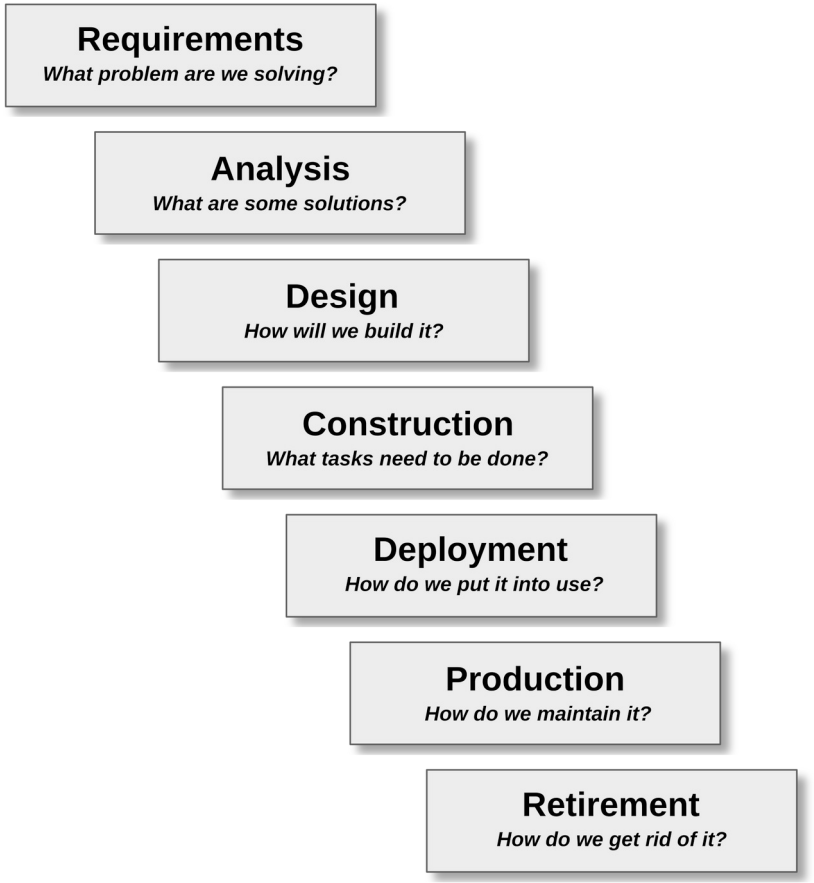
Retirement

- Plan for taking the application out of service
  - Without disrupting the business
  - Without services to clients becoming unavailable
- Plan for transitioning users to the application’s replacement



Causes of Failures

- Product failures are often the result of premature construction
  - We start building something before we are clear on what it should do
- Or we do not evaluate how well the product is performing in production



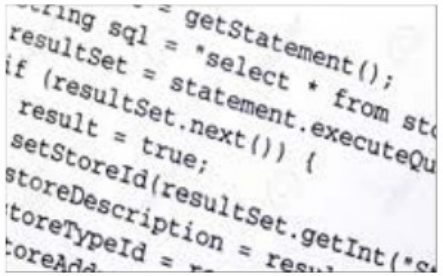
The most important single aspect of software development is to be clear about what you are trying to build.
Edsger Dijkstra

First, solve the problem.
Then, write the code
Donald Knuth

Terminology

Defects

causes



causes

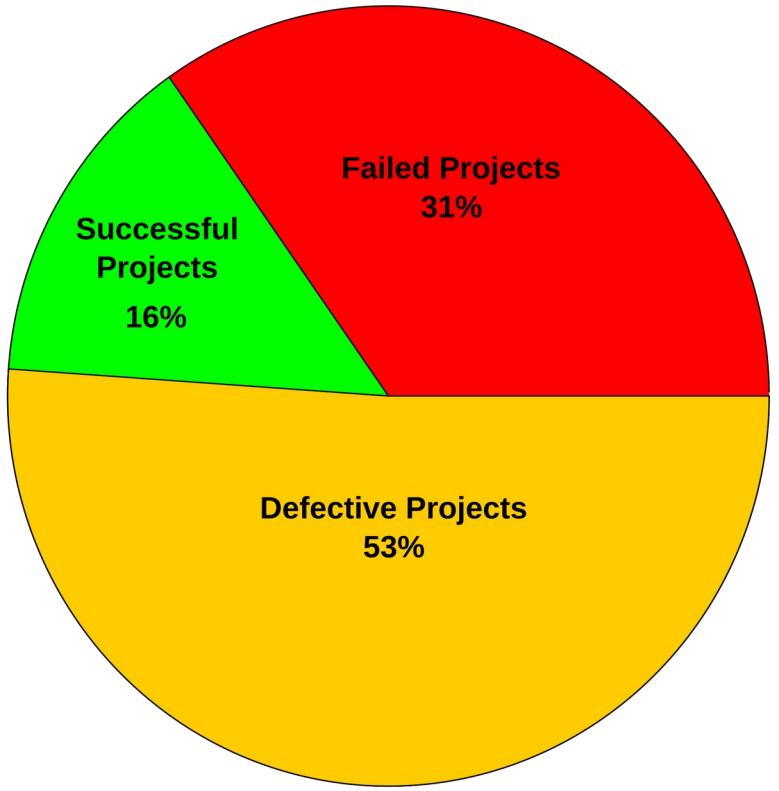
Error **Fault** **Failure**

Terminology

- *Defect*: A generic term for any of the following
- *Failure*: When a product does not perform its required functions according to the stated specifications or fails testing
- *Fault*: An incorrect step, construct, process or mistake in the product that causes failures
  - We can eliminate faults by fixing the code that results in a failure
  - Faults always result in the same failures until the code is fixed
- *Error*: A error is a human action that resulted in a fault
  - Misunderstanding requirements
  - Lack of skill for construction tasks
  - Lack of communication between stakeholders and team members or between team members
  - And so on...

Chaos Report Analysis

- Resolution Type 1
  - Success: Project was on time, on budget and to spec.
- Resolution Type 2
  - Defective: Project was late, and/or over budget and/or not to spec
- Resolution Type 3
  - Failed: Project cancelled



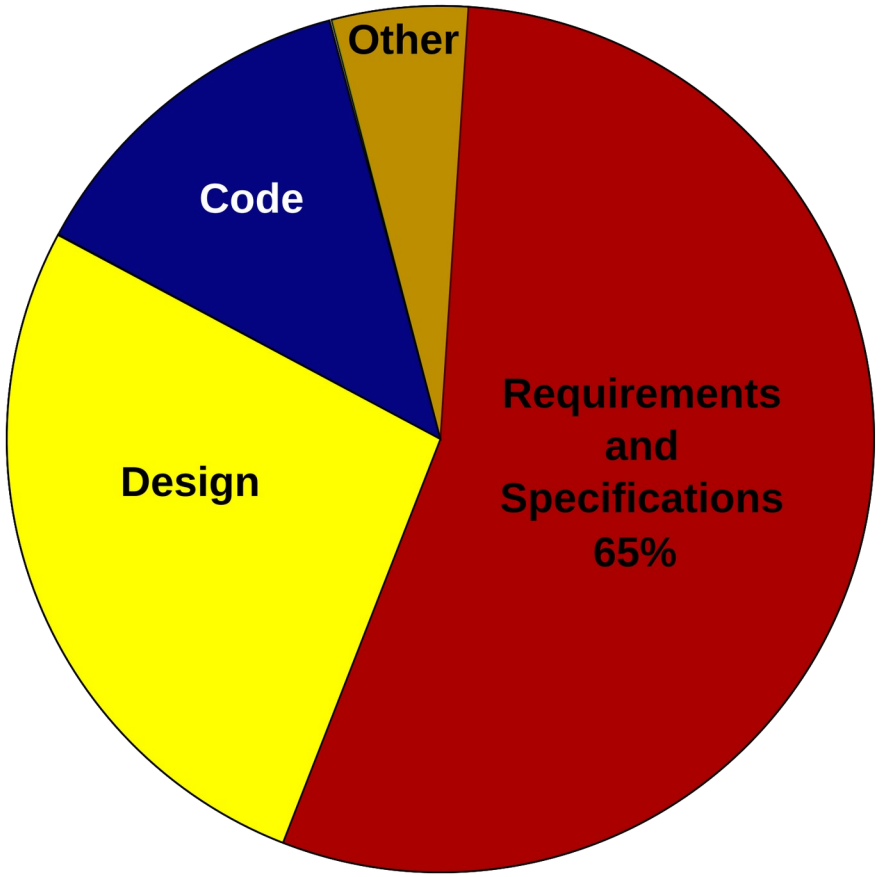
Factors in Resolution Types

Success Factors	%
User Involvement	*16*
Executive Management Support	14
Clear Statement of Requirements	*13*
Proper Planning	*10*
Realistic Expectations	*8*
Smaller Project Milestones	8
Competent Staff	7
Ownership	5
Clear Vision and Objectives	*3*
Hard Working and Focused Staff	2
Other	13

Defective Factors	%
Lack of User Involvement	*13*
Incomplete Requirements & Specs	*12*
Changing Requirements & Specs	*12*
Lack of Executive Support	8
Technology Incompetence	7
Lack of Resources	6
Unrealistic Expectations	*6*
Unclear Objectives	*5*
Unrealistic Time Frames	*4*
New Technology	4
Other	23

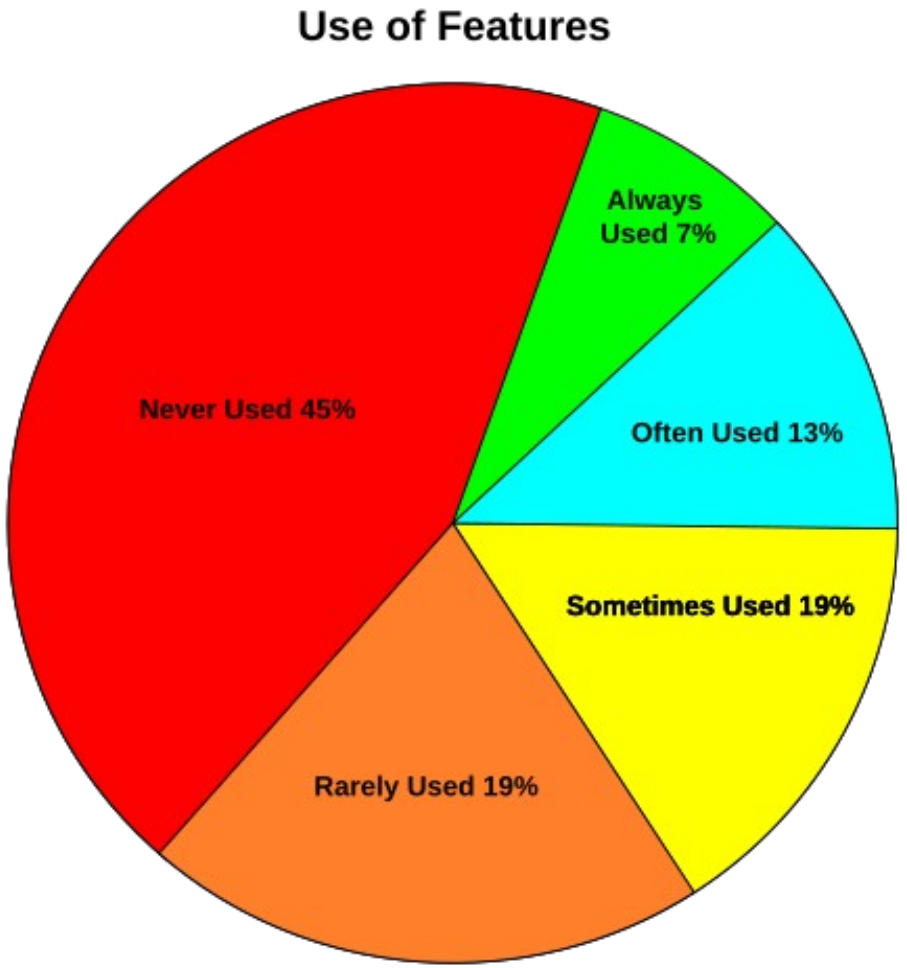
Failure Factors	%
Incomplete Requirements	*13*
Lack of User Involvement	*12*
Lack of Resources	11
Unrealistic Expectations	*10*
Changing Requirements & Specs	*9*
Lack of Management Support	9
Lack of Planning	*8*
Didn't Need It Any Longer	*8*
Lack of IT Management	6
Technological Illiteracy	4
Other	10

Sources of Errors

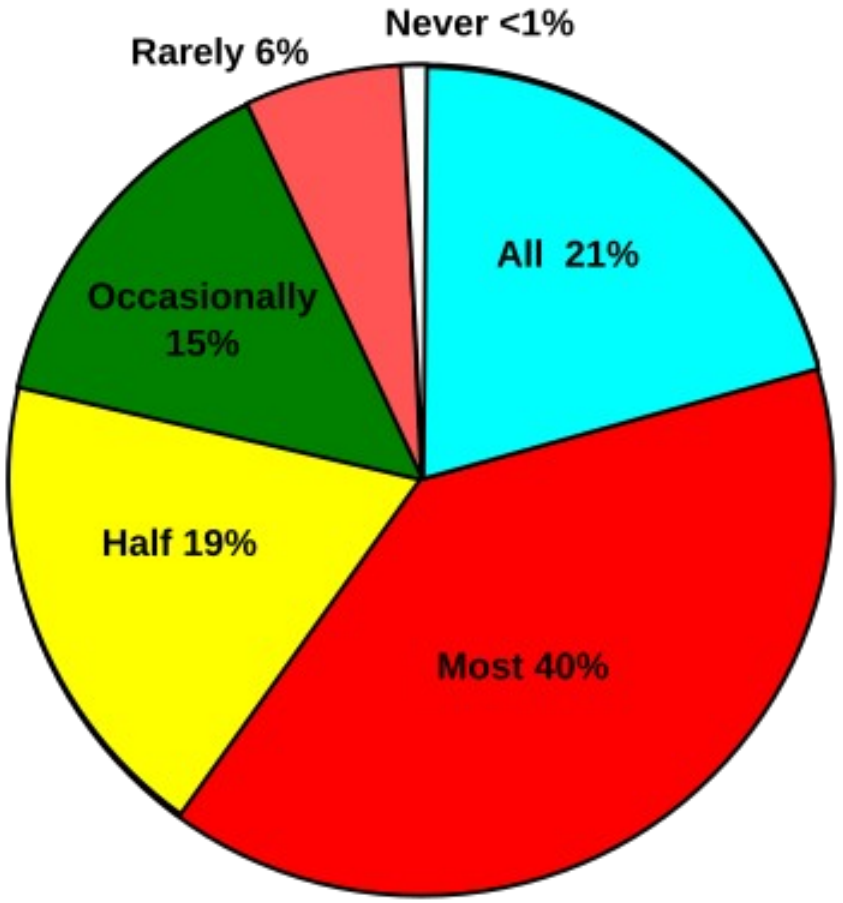


Over Engineering

- Over engineering is building features that are not needed or used.
- The typical usage of features in software shows only about 20% are used always or often.
- Even more remarkable is that 45% are never used.
- Development effort on never-used features is not efficient



The Cost of Rework



:

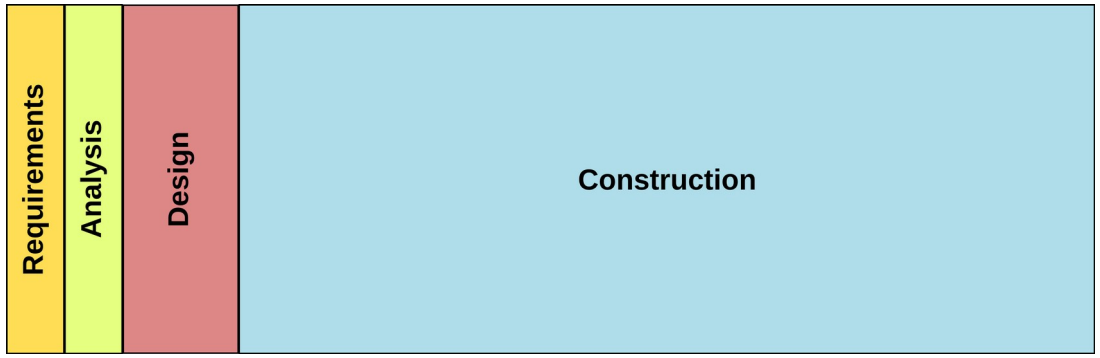
Geneca Survey of IT Professionals:

"How much of your time do you spend on avoidable or preventable rework (eg. changing requirements, missing features)?"

Results are consistent with other studies.

The Cost of Rework

- Defective requirements, design, and construction
  - Result in re-work
  - Typically consumes 40 to 50 percent of the total cost of development
  - Can be as high as 80%.
- Construction
  - Typically consumes 80% or more of the resources of a project
  - Only purpose of the requirements, analysis and resign phases is to prevent errors that force rework in the construction phase



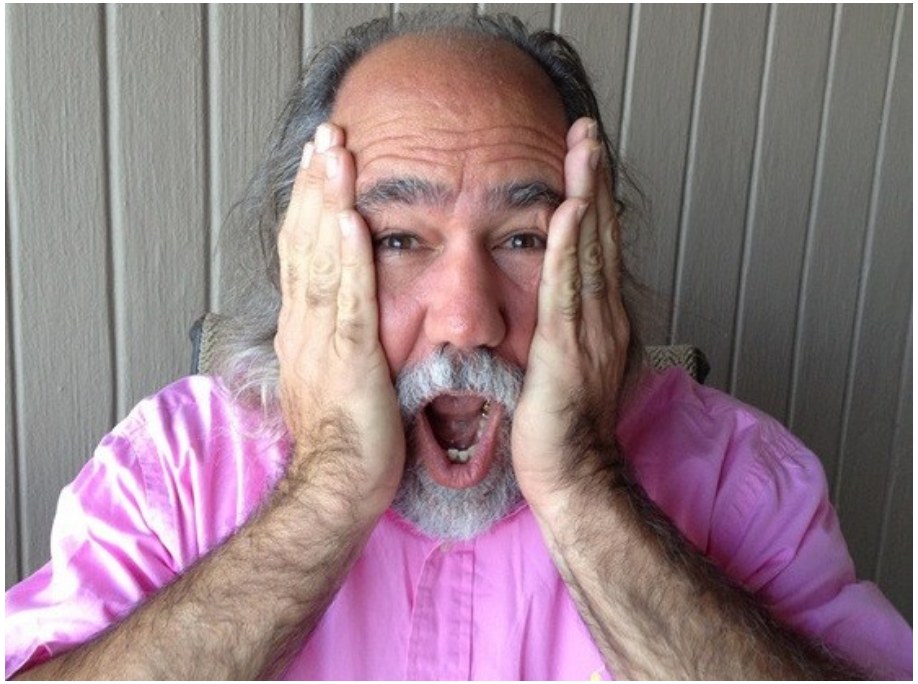
Mission Critical Industrial Strength Software

Mission critical software tends to have a long lifespan, and over time, many users come to depend on their proper functioning. In fact, the organization becomes so dependent on the software that it can no longer function in its absence. At this point, we can say the software has become industrial-strength.

The distinguishing characteristic of industrial strength software is that it is intensely difficult, if not impossible, for the individual developer to comprehend all of the subtleties of its design.

Stated in blunt terms, the complexity of such systems exceeds the human intellectual capacity. Alas, this complexity we speak of seems to be an essential property of all large software systems. By 'essential' we mean that we may master this complex ity, but we can never make it go away.

Grady Booch



Legacy Monolithic Issues

- Codebase is enormous
  - Little or no documentation
  - Coupled to legacy technology
- Eg. Internal Revenue Service
  - Running 1960s vintage code
  - Application highly complex
  - Attempts to replace it have failed
  - Over $15 billion so far
- The IRS is not unique
  - Both public and private sectors

Legacy Monolithic Issues

CIO News Feature

No Easy IT Fix for the IRS

The internal revenue service's Master File is an accident waiting to happen. A legacy of the Kennedy administration, this database stores the taxpaying histories of 227 million individuals and corporations, including every transaction between taxpayers and the IRS for the past 40 years. The Master File is used to determine if you've paid what you owe, and without it the government would have no way to flag returns for audits, pursue tax evaders or even know how much money is or should be flowing into its coffers.

Yet the system still runs code from 1962, written in an archaic programming language almost no one alive understands. Every year, programmers, some who have worked at the IRS for decades, add new code to the Master File to reflect new rules passed by Congress. As a result, the system has become a high-tech Rube Goldberg machine. Those familiar with the Master File say it is poised for a fatal crash that would shut the government down.

From the GAO report:

As a result of the lack of policies and procedures, [the modernization] projects did not consistently follow disciplined requirements development and management practices.... none of the projects met all of the practices needed for effective requirements management. For example, two did not implement all needed practices in eliciting (gathering) requirements; two did not have fully documented requirements; and two could not produce fully traceable requirements (i.e., the requirements could not be tracked through development and testing). Unless BSM takes the steps needed to develop and institutionalize disciplined requirements development and management processes and to improve interim guidance, it will continue to face risks, including cost overruns, schedule delays, and performance shortfalls.

Legacy Monolithic Issues

CIO News Feature

No Easy IT Fix for the IRS

The internal revenue service's Master File is an accident waiting to happen. A legacy of the Kennedy administration, this database stores the taxpaying histories of 227 million individuals and corporations, including every transaction between taxpayers and the IRS for the past 40 years. The Master File is used to determine if you've paid what you owe, and without it the government would have no way to flag returns for audits, pursue tax evaders or even know how much money is or should be flowing into its coffers.

Yet the system still runs code from 1962, written in an archaic programming language almost no one alive understands. Every year, programmers, some who have worked at the IRS for decades, add new code to the Master File to reflect new rules passed by Congress. As a result, the system has become a high-tech Rube Goldberg machine. Those familiar with the Master File say it is poised for a fatal crash that would shut the government down.

IT Failures and Complexity

The United States is losing almost as much money per year to IT failure as it did to the [2008] financial meltdown. However the financial meltdown was presumably a onetime affair. The cost of IT failure is paid year after year, with no end in sight. These numbers are bad enough, but the news gets worse. According to the 2009 US Budget [02], the failure rate is increasing at the rate of around 15% per year.

Is there a primary cause of these IT failures? If so, what is it?.... The almost certain culprit is complexity.... Complexity seems to track nicely to system failure.

Once we understand how complex some of our systems are, we understand why they have such high failure rates.

We are not good at designing highly complex systems. That is the bad news. But we are very good at architecting simple systems. So all we need is a process for making the systems simple in the first place.

Roger Sessions

Engineering Challenge

- Systems or products can be very large and complex
  - Causes a breakdown in applying the engineering cycle
- We often have to break a complex product down into simpler components
  - And even subdivide those in order manage complexity
- If a project is too large
  - We lose control of managing the artifacts at each stage
  - Too many requirements, too much architecture, etc
- With new tech, requirements often don’t exist until we deploy prototypes
  - If the product is too large, this become almost impossible to do

FBI Virtual Case File System

washingtonpost.com

FBI Rejects Its New Case File Software

Database Project Has Cost Nearly $170 Million

By Jonathan Krim
Washington Post Staff Writer
Friday, January 14, 2005; Page A05

FBI Director Robert S. Mueller III, right, conferred with Sept. 11 panelist Jamie S. Gorelick last year. "There were problems we did not anticipate," he said about the new computer software yesterday. (Lucian Perkins -- The Washington Post)

The new system was called “Virtual Case File” which was supposed to, according to the Post article, provide a modern database for storing and indexing all case information and entries by agents, enabling them to share files electronically and search easily for links between cases that might not otherwise seem connected.

The contract was awarded to Science Applications International Corp. (SAIC) of San Diego who, after a $170 million expenditure, delivered in their latest version in December of 2004. However, the FBI concluded that the delivered system was outdated and was seriously deficient resulting in it being abandoned before it was even launched. In one FBI document from January 2005, the FBI stated that the system did not meet their needs.

From the Post article: *"I am frustrated,"* FBI Director Robert S. Mueller III said when asked about the software at a news conference in Birmingham, according to Reuters news service. ***"There were problems we did not anticipate."*** Sen. Charles E. Grassley (R-Iowa) said in a statement that *"the likely possibility of the FBI scrapping the system is disappointing," and "I hope we haven't just been pouring money down a rat hole at taxpayers' expense."*

FBI Virtual Case File System

washingtonpost.com

FBI Rejects Its New Case File Software

Database Project Has Cost Nearly $170 Million

By Jonathan Krim
Washington Post Staff Writer
Friday, January 14, 2005; Page A05

FBI Director Robert S. Mueller III, right, conferred with Sept. 11 panelist Jamie S. Gorelick last year. "There were problems we did not anticipate," he said about the new computer software yesterday. (Lucian Perkins -- The Washington Post)

The FBI faced obstacles in a number of key areas relating to the VCF program.

(Mueller testimony)

1. *We did not have a complete set of defined VCF requirements when the original contract was signed in June 2001.*

2. *The contract was based on hours worked -- cost plus an award fee. We now know these types of contracts are difficult to manage. Although the requirements were solidified in November 2002, the contract remained a cost-plus-award-fee contract.*

3. *We lacked skill sets in our personnel such as qualified software engineering, program management, and contract management. We also experienced a high turnover in Trilogy program managers and Chief Information Officers.*

4. *We underestimated the complexity of interfacing with our legacy system, of addressing our security needs, and of establishing an enterprise architecture.*

US Terrorist Watchlist

BART GORDON, TENNESSEE
CHAIRMAN

RALPH M. HALL, TEXAS
RANKING MEMBER

U.S. HOUSE OF REPRESENTATIVES

COMMITTEE ON SCIENCE AND TECHNOLOGY

SUITE 2320 RAYBURN HOUSE OFFICE BUILDING
WASHINGTON, DC 20515–6301
(202) 225–6375
TTY: (202) 226–4410
http://science.house.gov

August 21, 2008

Mr. Edward Maquire
Inspector General
Office of the Director of National Intelligence
Washington, D.C. 20511

Dear Inspector General Maquire:

The National Counterterrorism Center (NCTC) was established in 2004 to coordinate government counterterrorism efforts and integrate the government's terrorism intelligence data. The NCTC "serves as the central and shared knowledge bank on terrorism information" and "establishes the information technology (IT) systems and architectures within the NCTC and between the NCTC and other agencies that enable access to, as well as integration, dissemination, and use of, terrorism information."[3] As part of its mission, the NCTC, which reports to the Office of the Director of National Intelligence (ODNI) maintains, updates and integrates all intelligence information from across the federal government regarding known or suspected terrorists.[4] This data is stored in a database called the Terrorist Identities Datamart Environment (TIDE) that is used to help compile the government's consolidated terrorist watchlist.[5]

But, the Subcommittee has learned that the TIDE database is suffering from serious, long-standing technical problems. The Subcommittee has also learned that a critical NCTC initiative, named "Railhead," which is intended to replace TIDE with enhanced capabilities has suffered from severe technical troubles, poor contractor management and weak government oversight. As a result, potentially hundreds of millions of dollars have been wasted, delivery schedules have slipped, contractor employees have been laid off in order to restrain escalating costs, and the NCTC is now scrambling either to fix the technical troubles or possibly to abandon the program altogether. The end result is a current IT system used to identify terrorist threats that has been crippled by technical flaws and a new system that if actually deployed will leave our country more vulnerable than the existing yet flawed system in operation today.

US Terrorist Watchlist

BART GORDON, TENNESSEE
CHAIRMAN

RALPH M. HALL, TEXAS
RANKING MEMBER

U.S. HOUSE OF REPRESENTATIVES

COMMITTEE ON SCIENCE AND TECHNOLOGY

SUITE 2320 RAYBURN HOUSE OFFICE BUILDING
WASHINGTON, DC 20515–6301
(202) 225–6375
TTY: (202) 226–4410
http://science.house.gov

August 21, 2008

Mr. Edward Maquire
Inspector General
Office of the Director of National Intelligence
Washington, D.C. 20511

Dear Inspector General Maquire:

The TIDE database has evolved overtime as both contractors and government employees have attempted to expand and enhance the database to improve their own use of the system. But none of them appear to have taken into account the overall design or engineering architecture of the entire system. As a result, there are now dozens of tables or categories for identical fields of information making the ability to search or locate key data inefficient, ineffective and more time consuming and difficult than necessary.

In addition, the TIDE database relies on Structured Query Language (SQL), a cumbersome computer code that must utilize complicated sentence structures to query the tables, rows and columns that encompass the TIDE database. Without proper documentation on whether a table contains information on names, addresses, vehicles, license plates or an individual's nationality, for instance, analysts have no valid mechanism to conduct a search of these "undocumented" tables.

The original TIDE database, built by Lockheed Martin, replaced the Department of State's TIPOFF database, designed and built by The Analysis Corporation[9], in the wake of the 9.11 terrorist attacks to automate the terrorist watch list. The TIDE database was built in Oracle as a relational database management system (RDBMS). This original database, however, suffers from basic design, management and maintenance inefficiencies and problems. For instance, only about 60% of the data, including names and addresses, mentioned in CIA cables provided to NCTC are actually extracted from these messages and placed into the TIDE database.[10]

US Terrorist Watchlist

BART GORDON, TENNESSEE
CHAIRMAN

RALPH M. HALL, TEXAS
RANKING MEMBER

U.S. HOUSE OF REPRESENTATIVES

COMMITTEE ON SCIENCE AND TECHNOLOGY

SUITE 2320 RAYBURN HOUSE OFFICE BUILDING
WASHINGTON, DC 20515–6301
(202) 225–6375
TTY: (202) 226–4410
http://science.house.gov

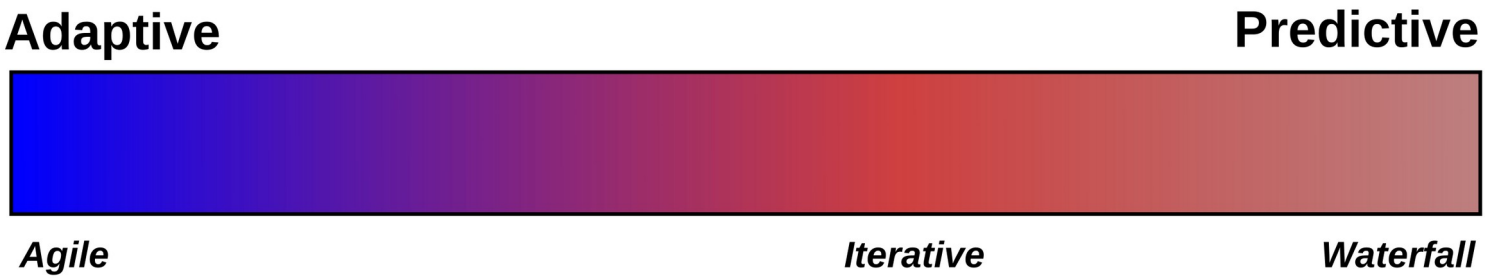
August 21, 2008

Mr. Edward Maquire
Inspector General
Office of the Director of National Intelligence
Washington, D.C. 20511

Dear Inspector General Maquire:

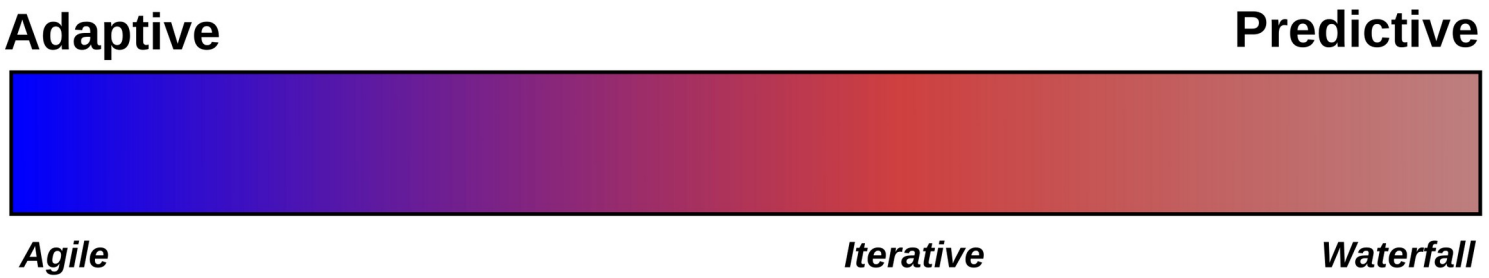
Without a detailed index of the data stored in each table in TIDE, the SQL search engine is blindfolded, unable to locate or identify undocumented data. The current TIDE database is composed of data fields that are presented in 463 separate tables, 295 of which are undocumented, according to one internal Railhead document.[11] As a result, critical terrorist intelligence in the TIDE system may not be searched at all. "Existing TIDE data model is complex, undocumented, and brittle," the document notes, "which poses significant risk to RLSI [Railhead Lead System Integrator[12]] data migration and modeling."

Process Types



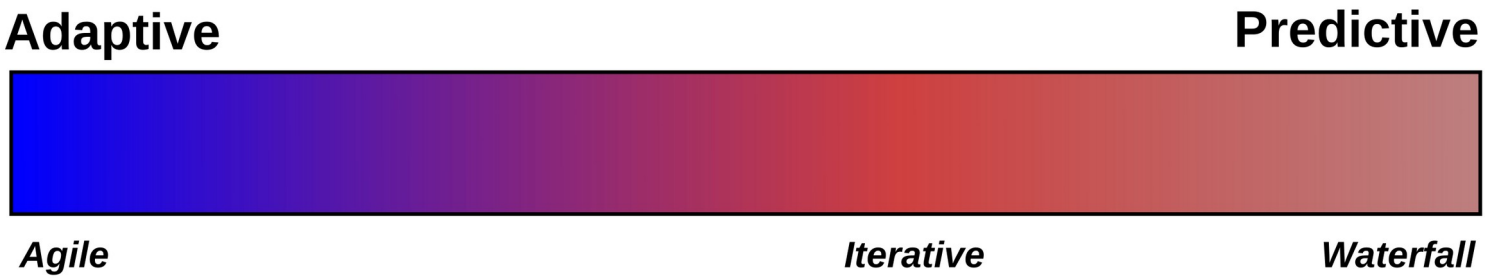
- Highly predictive methodologies work by planning the future in detail
  - The project plan is developed for a specific known development target.
  - Changes to that plan almost always cause rework.
  - Predictive teams usually use a change management process so that only significant changes are considered for incorporation into the project
- Predictive processes have all of the requirements at the start of the project
  - The project can predict exactly what the end results should look like

Process Types



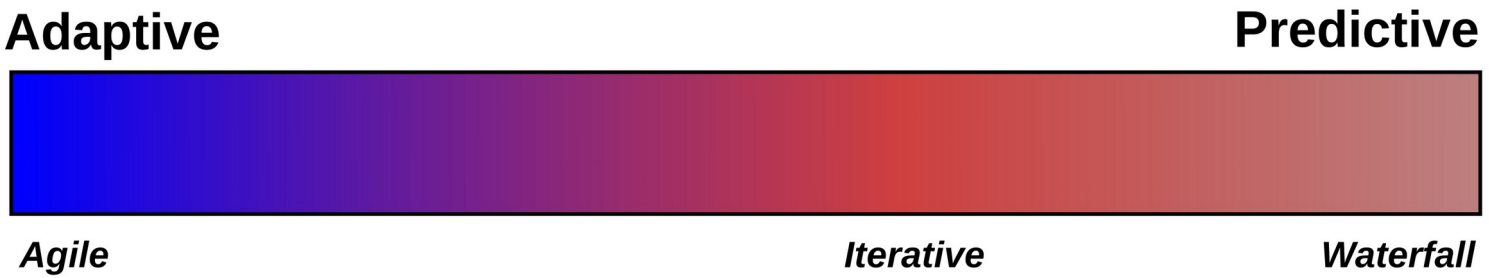
- Highly adaptable methodologies are designed to react quickly to new and changing requirements
- The final result can only be predicted to the degree the requirements are known
  - Characterized by the use of prototypes to uncover unknown requirements
  - The known requirements are used to create a part of a specification and design for the prototype
  - The discovered requirements are used to modify and extend the specification and design
  - This process cycles iteratively

Process Types



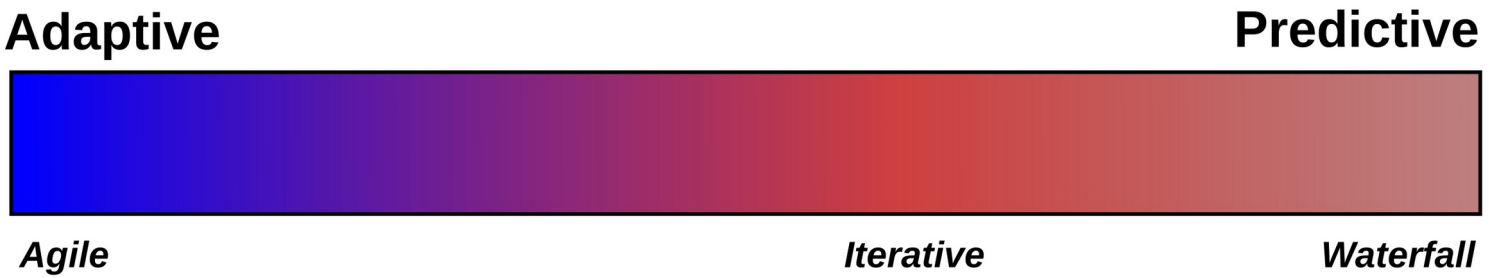
- Projects can be anywhere on the spectrum of adaptability
  - For example, parts of it may be completely specified
  - But there may be some aspects, like the user interface, that are not
- Novel technology is often developed adaptively
  - User cannot give requirements until they actually get a prototype to try out
- The final form of the product if often decided by stopping development
  - At the point where the solution is adequate to solve the problem

Process Types



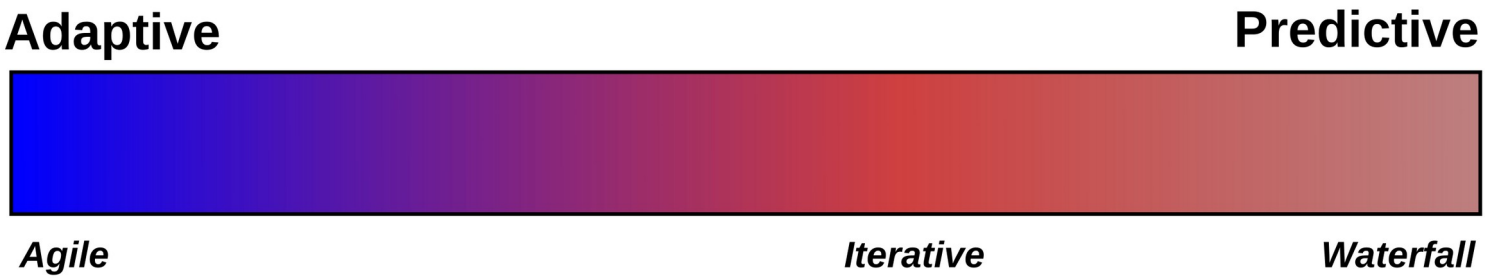
- Iterative projects are mostly predictive
  - However, if the project is long lived (multi-year for example)
  - Occasional reality checks are done on an incremental version of the product
  - This ensures any requirements changes or technology drift has been accounted for

Process Types



- Predictive and adaptive processes occur in every industry and field
  - Planned surgeries are predictive – emergency room medicine is adaptive
  - Military strategy involves both predictive and adaptive processes
  - New product development is very often adaptive to respond to market reaction
  - Mature product development is predictive
  - Six Sigma, for examples, ensures each product produced matches the predicted outcome

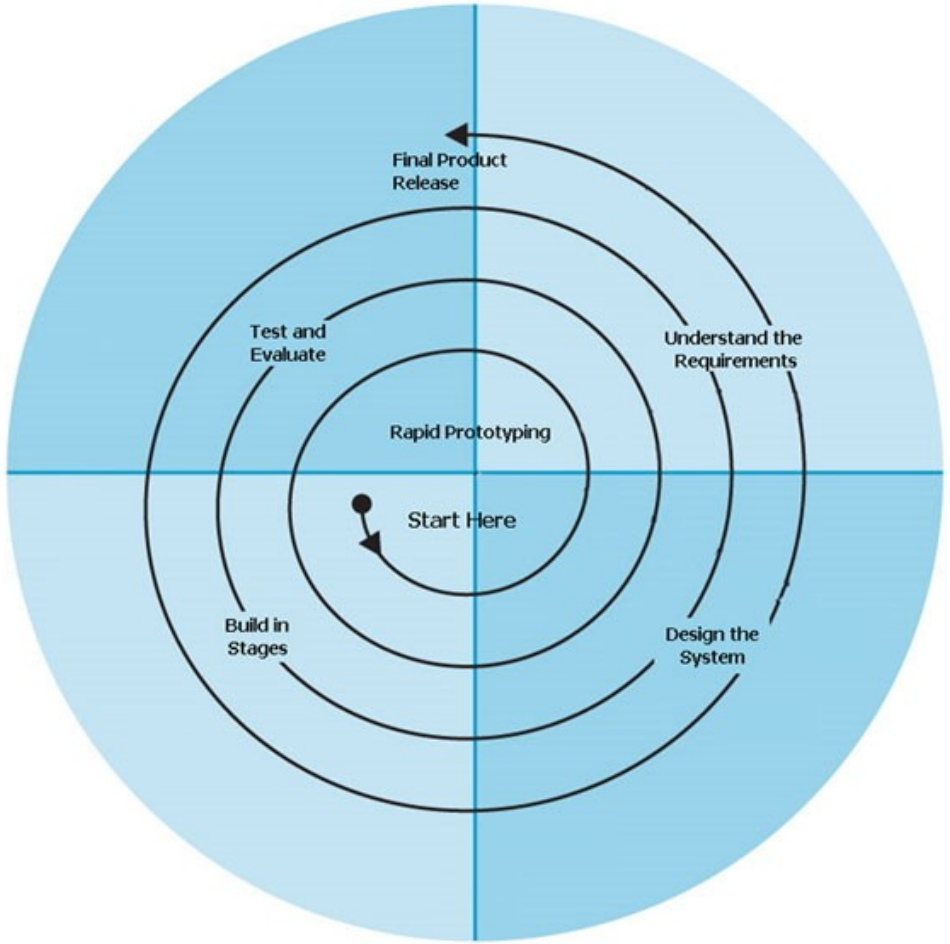
Process Types



- Complex projects can be predictive in terms of the outcome
  - But evolving an optimal can be adaptive
  - Finding the right way to construct the predicted outcome
- Adaptive processes are used when effects of a problem can be observed
  - But the nature of the underlying problem is unknown
  - Which mean that the nature of the solution is unknown
  - Different potential solutions are tried until one is found that works

Adaptive Software Processes

- Developed during the 1980s
  - New product development with brand new technologies
  - Like human machine interfaces
- Two that were quite successful
  - Barry Boehm's spiral methodology
  - James Martin's Rapid Application Development (RAD) at IBM
  - Development of user interfaces, have requirements that are too fluid for a predictive approach
  - These models centred around getting prototypes into the hands of the users to start generating feedback that would be used to continuously improve the product



http://softwaretesting-qaqc.blogspot.com

Agile Development Processes

- In the late 1990s there was a rebellion against the high level of ceremony and formalism of RUP, which had become quite popular
- Starting with Extreme Programming, several methodologies started to emerge that shared a common core of ideas:
  - Close collaboration between the development team and business experts
  - Face-to-face communication as opposed to written documentation
  - Frequent delivery of working prototypes to facilitate developer and customer communication
  - Small, tight, self-organizing teams
  - Sets of techniques to "craft" code and organize the team to anticipate the inevitable "churn" of constantly changing requirements.
- These methodologies referred to themselves as "light-weight", "agile" and "low ceremony" and were intended for smaller projects with fast turnaround times that did not need the formality or complexity of the RUP style methodologies

Martin Fowler

Agile methodologies developed as a reaction to these [heavy-weight] methodologies. For many people the appeal of these agile methodologies is their reaction to the bureaucracy of the engineering methodologies. These new methods attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff. The most immediate difference is that they are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways they are rather code-oriented: following a route that says that the key part of documentation is source code.

Agile methods are adaptive rather than predictive. Engineering [heavy-weight] methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.

Agile methods are people-oriented rather than process-oriented. The goal of engineering methods is to define a process that will work well whoever happens to be using it. Agile methods assert that no process will ever make up the skill of the development team, so the role of a process is to support the development team in their work.

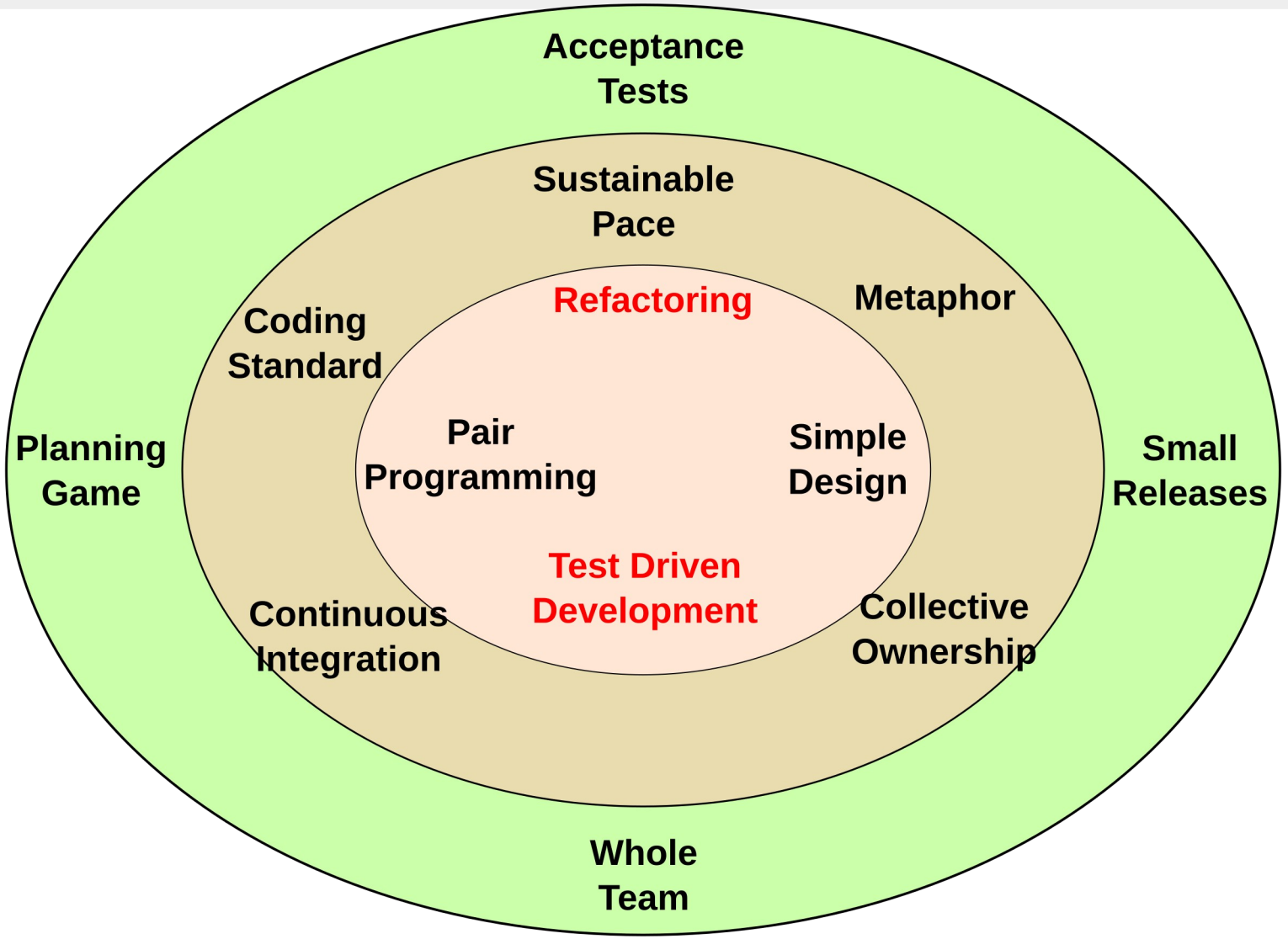
Agile Manifesto

- On February 11-13, 2001, the famous Snowbird conference took place
- Representatives from a number of adaptive methods like Extreme Programming, SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development and Pragmatic Programming met to identify common principles they shared
- Out of that came the agile Manifesto and the 12 Agile principles

Extreme Programming (XP)

- Developed by Kent Beck
- XP is
  - A “set of values, principles and practices for rapidly developing high-quality software that provides the highest value for the customer in the fastest way possible.”
  - Takes 12 well-known software development "best practices" to their logical extremes
  - Inteded for projects of up to a dozen programmers and twice that with some difficulty
  - XP does not scale well
  - For large scale development a project is organized overall along more traditional models, but is then split into multiple smaller XP projects.

The XP Onion


Acceptance Tests
Sustainable Pace
Refactoring
Metaphor
Coding Standard
Planning Game
Pair Programming
Simple Design
Small Releases
Test Driven Development
Continuous Integration
Collective Ownership
Whole Team

XP Practices

- XP is seen as a sets of practices within practices
  - Each layer of the onion feeds into and defines the activities in the layer beneath it.
- The Planning Game:
  - Customers and developers cooperate to produce the maximum business value as rapidly as possible.
  - Customers come up with a list of desired features for the system
  - Each feature is written out as a User Story, which gives the feature a name.
  - Developers estimate how much effort each story will take, and how much can be done an iteration.
  - Customers prioritize the stories to be implemented, as well as when and how often to produce a production release of the system
- Small Releases:
  - Start with the smallest useful feature set. Release early and often, adding a few features each time.
- Metaphor:
  - Each project has an organizing metaphor, to provide an easy to remember naming convention.

XP Practices

- Simple Design:
  - Always use the simplest possible design that gets the job done.
  - The requirements will change tomorrow, so only do what's needed to meet today's requirements.
- Continuous Testing (TDD):
  - Tests are written for a feature to be added, then the code is written. When the suite runs, the job is done.
- Refactoring:
  - Refactor out any duplicate code generated in a coding session.
  - You can do this with confidence that you didn't break anything because you have the tests.
- Pair Programming:
  - All production code is written by two programmers sitting at one machine
  - Essentially, all code is reviewed as it is written.

XP Practices

- Collective Code Ownership:
  - No one "owns" a module.
  - Anyone is expected to be able to work on any part of the codebase at any time.
- Continuous Integration:
  - All changes are integrated into the codebase at least daily.
  - The tests have to pass 100% both before and after integration.
- 40-Hour Work Week (Sustainable Pace):
  - Programmers go home on time, up to one week of overtime is allowed.
- On-site Customer Tests:
  - Development team has continuous access to a real live customer, that is, someone who will actually be using the system who does continuous test generation and acceptance.
- Coding Standards:
  - Everyone codes to the same standards so that there is no way to tell by inspection who on the team worked on a specific piece of code.

Evolving Architecture

- XP embraces change which means that an architecture and design "evolve" rather than being initially specified as something the application has to "fit into”
- Architecture and design are expressed in XP through:
  - Spike
  - Metaphor
  - First Iteration
  - Small Releases
  - Refactoring
  - Team Practices
- Design evolves because it is driven by the requirements of the customer, which usually evolve and change as the system is developed
  - Refactoring is not just writing code, evolving design is not the same as a haphazard or unstructured design.
  - If developers do not apply good design principles to the emerging design, it will become brittle and non-functional in a very short period of time.
  - Refactoring simplifies because a simplified design is usually more robust s

Spike

- Quick throw-away potential solution
- Based on the user stories and the spikes, an architectural approach will emerge that will suggest the system's structure
  - Stories suggest constraints that guide architectural choices.
  - One story might identity security and access constraints while another might suggest performance levels for multiple users
  - This may suggest a spike simulating multiple simultaneous users that might provide insight into replication and rollover designs.
- The spikes allow the developer and customer to examine the costs and trade-offs of the various proposed solutions
  - Spikes are used to experiment with implementations of functionality and how the solution would impact the existing system structure

Metaphor

- An effective metaphor is developed that helps guides the solution.
  - Describes the solution in the language of the problem domain
  - Describes the main conceptual objects and their relationships and provides a "check-point" or a high level view of a system that keeps the design oriented or on track
  - Also provides a common language for naming and talking about the artifacts of the system.
- The metaphor may change as more is discovered about the customer's domain
  - Not tied to the architecture but suggests and influences the architecture
  - The system design supports the metaphor.

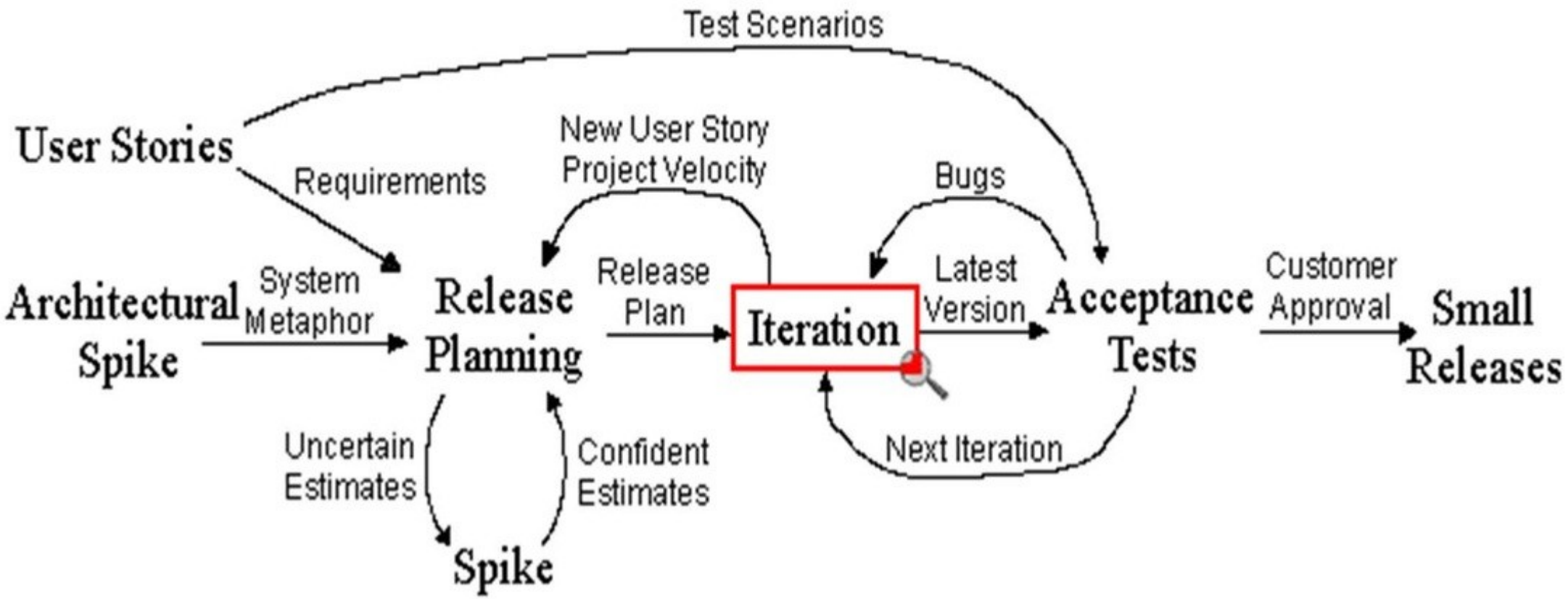
First Iteration

- First iteration a ZFR (“ziffer,” Zero Feature Release)
  - Does nothing but puts an initial architecture (often called the architecture spike) into place
  - The first iteration is guided by a selection of user stories that have architectural significance – i.e. they force the whole system to be considered
  - This is the working skeleton since the first release has no functionality.
  - The focus is making sure that the solution is installable and configurable.
- Also ensures the environment is set up correctly, the compilation and link tools work, and all of the other configuration issues have been resolved

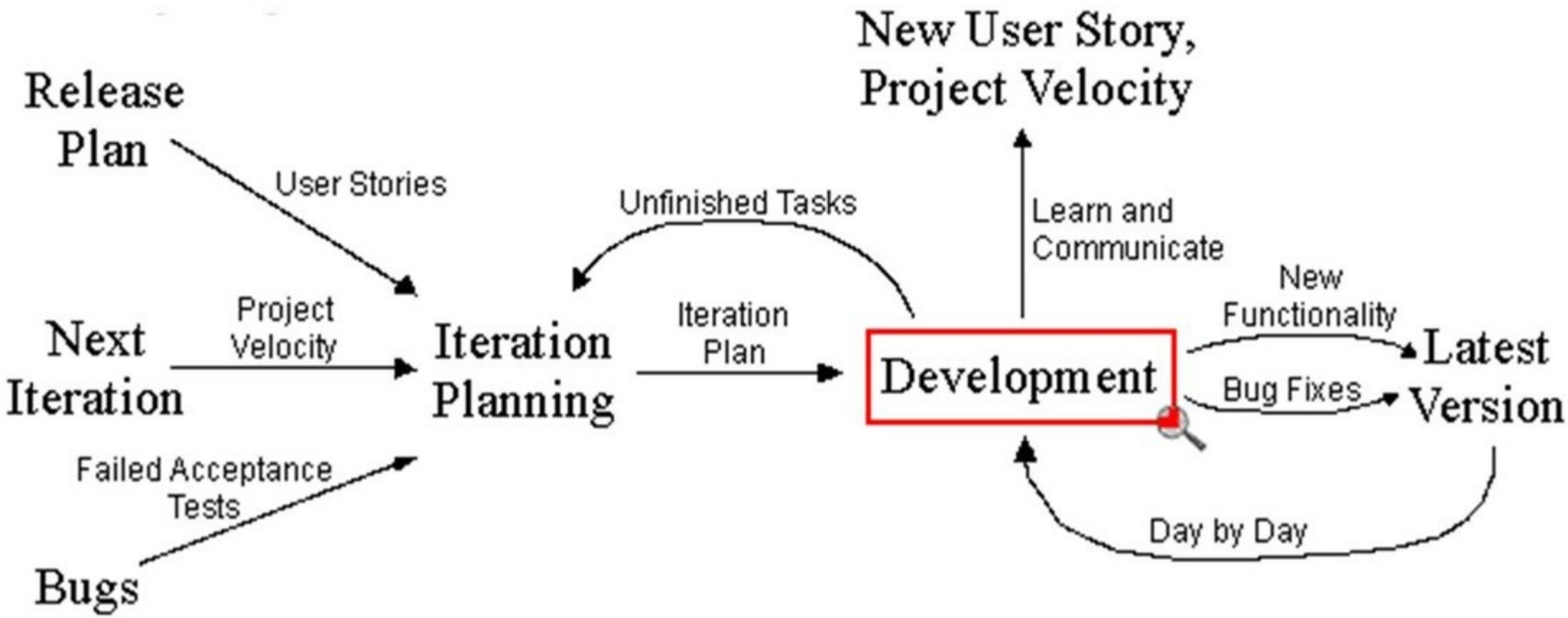
Small Releases and Refactoring

- A constant stream of small releases allows monitoring the impact of design changes and additions
  - Each release is a checkpoint: new functionality may break the existing design, so we have to go back and make modifications to the design to accommodate the new requirements
- Remember that a spike is a throw-away solution to see how to solve a particular problem
  - Spikes help us choose low-risk ways to extend or modify the architecture and design.
  - Refactoring allows us to change the design as we move forward.
  - As we add new functionality, we may have to change the design of parts of the system without changing their functionality.
  - Constant refactoring allows us to keep the design fluid and flexible while not giving up robustness.

Extreme Programming Project

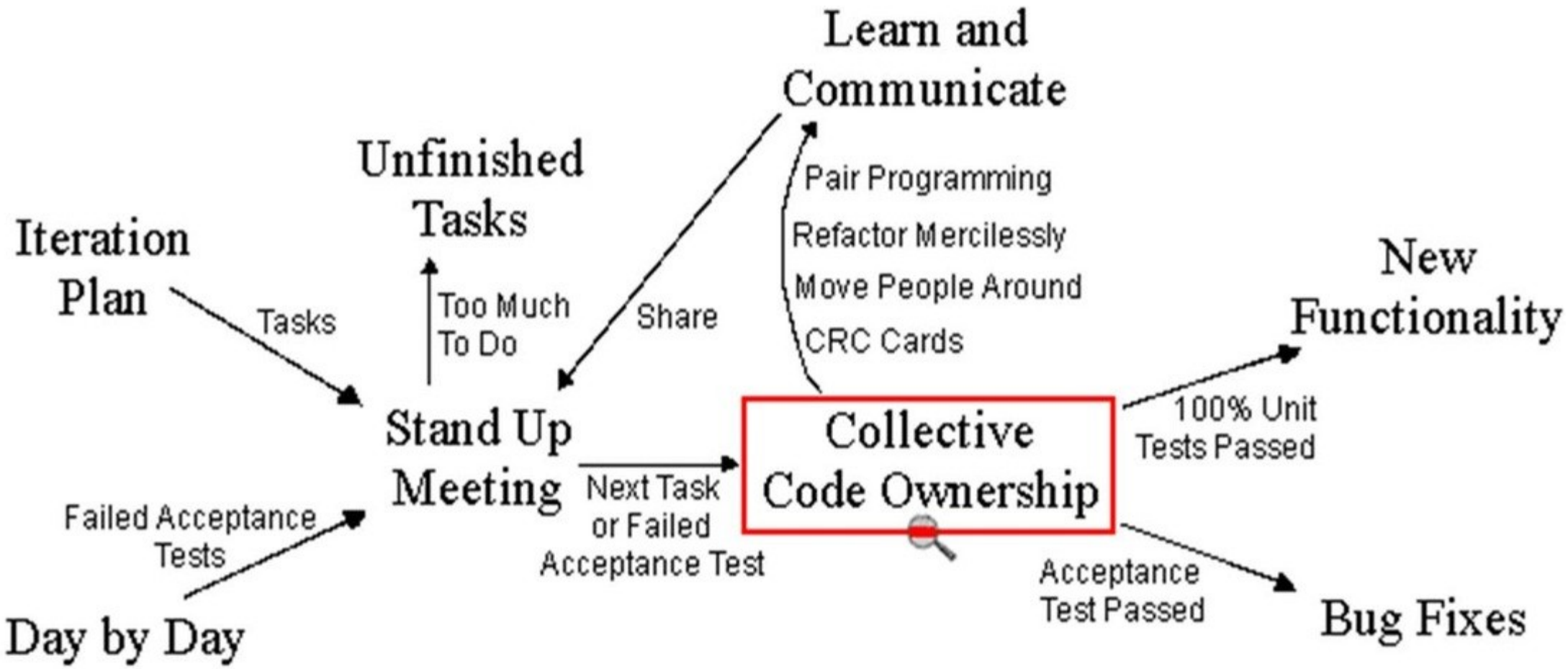


Iteration


Zoom Out
New User Story,
Project Velocity
Release
Plan
User Stories
Unfinished Tasks
Learn and
Communicate
New
Functionality
Project
Velocity
Iteration
Plan
Next
Iteration
Iteration
Planning
Development
Latest
Version
Bug Fixes
Failed Acceptance
Tests
Day by Day
Bugs

Development

Zoom Out



End of Module