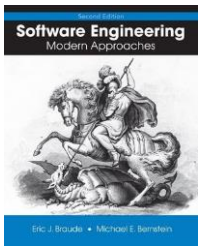
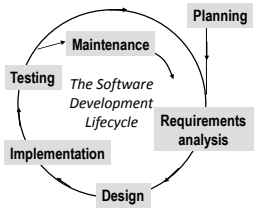
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 1. Introduction



- Why is software engineering important?
- Who and what does it consist of?
- What are its main activities?
- What are the principles of software engineering?
- What ethics are involved?
- What sorts of case studies will be used to illustrate the subject?

Goal of Software Engineering

- Creation of software systems that are
  - Reliable
  - Efficient
  - Maintainable
  - Meet the needs of customers
- Production of system meets
  - Schedule
  - Budget

What is Software Engineering?

- Engineering discipline
  - *the design, analysis and construction of an artifact for some **practical** purpose*
- IEEE definition:
  - *"the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is the application of engineering to software."*

NATO Study Group

- NATO Study Group on Computer Science (1968)
  - one of the first uses of the phrase *software engineering*
- "Programming management will continue to deserve its current poor reputation for cost and schedule effectiveness until such time as a more complete understanding of the program design process is achieved."

NATO Study Group (cont.)

- "Today we tend to go on for years, with tremendous investments to find that the system, which was not well understood to start with, does not work as anticipated. We build systems like the Wright brothers built airplanes — build the whole thing, push it off the cliff, let it crash, and start over again."

Software Disasters

- Numerous examples of software disasters
  - Ariane Project
  - 1990 AT&T Disaster
  - Radiation Overdose
  - The link below has a list with several disasters due to software faults

https://en.wikipedia.org/wiki/List_of_software_bugs

Software Failure

- What is it?
  - Failure to meet expectations
- What expectations are not achieved?
  - Over budget
  - Exceeds schedule and/or misses market window
  - Doesn't meet stated customer requirements
  - Lower quality than expected
  - Performance doesn't meet expectations
  - Too difficult to use

Software Failure (cont.)

- Reasons for failure:
  - Unrealistic or unarticulated project goals
  - Poor project management
  - Inaccurate estimates of needed resources
  - Badly defined system requirements
  - Poor reporting of the project's status
  - Unmanaged risks
  - Poor communication among customers, developers, and users
  - Inability to handle the project's complexity
  - Poor software design methodology
  - Wrong or inefficient set of development tools
  - Inadequate test coverage
  - Inappropriate (or lack of) software process

Software Engineering Activities

4 P's of Software Engineering

- **People**
  - *Project stakeholders*
- **Product**
  - *The software product plus associated documents*
- **Project**
  - *The activities carried out to produce the product*
- **Process**
  - *Framework within which the team carries out the activities necessary to build the product*

People

Stakeholders

- ***Business Management***
- ***Project Management***
- ***Development Team***
- ***Customers***
- ***End-Users***

The Software Product Artifacts

- Project documentation

  Documents produced during software definition and development
- Code

  Source and object
- Test documents

  Plans, cases, and results
- Customer documents

  Documents explaining how to use and operate product
- Productivity measurements

  Analyze project productivity

Project

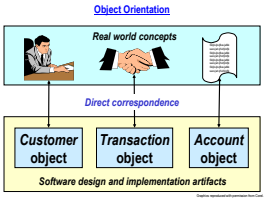
Software Project Activities

-- which produce a software product: Mainly…

- **Planning**
  - plan, monitor and control the software project
- **Requirements analysis**
  - define what to build
- **Design**
  - how to build the software
- **Implementation**
  - program the software
- **Testing**
  - validate software meets the requirements
- **Maintenance**
  - resolve problems; adapt software to meet new requirements;

Project (cont.)

- Development paradigm
  - e.g. object-oriented

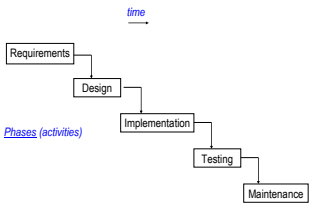


Process

- Framework for carrying out the activities of a project in an *organized* and *disciplined* manner.
- Imposes structure
- Waterfall or Iterative

Waterfall Process

The Waterfall Software Process



- Simplest process
- Sequential
- Basis for others

Iterative Process

- Software projects rarely follow ***strict*** waterfall
- Some ***iteration*** between specifications, design, implementation and test
- Requires discipline
  - e.g. update specifications when design changes

Software Engineering Principles

Software Engineering Principles

1. ***Make Quality Number 1***
2. ***High Quality Software is Possible***
3. ***Give Products to Customers Early***
4. ***Use an Appropriate Sofware Process***
5. ***Minimize Intellectual Distance***
6. ***Inspect Code***
7. ***People are the Key to Success***

Source: 201 Principles of Software Engineering, Alan Davis

Software Engineering Ethics

- Most disciplines operate under a strict set of *ethical* standards
- The *Merriam-Webster* online dictionary defines *ethics* as:
  - **1**: the discipline dealing with what is good and bad and with moral duty and obligation
  - **2:** a set of moral principles

Software Engineering Ethics (cont.)

- ACM/IEEE-CS Joint Task Force - *Software Engineering Code of Ethics and Professional Practices* (Version 5.1):
  - "Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

A Portuguese version can be found here:
https://www.computer.org/cms/Computer.org/professional-education/pdf/doc.pdf

Software Engineering Ethics (cont.)

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC - Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

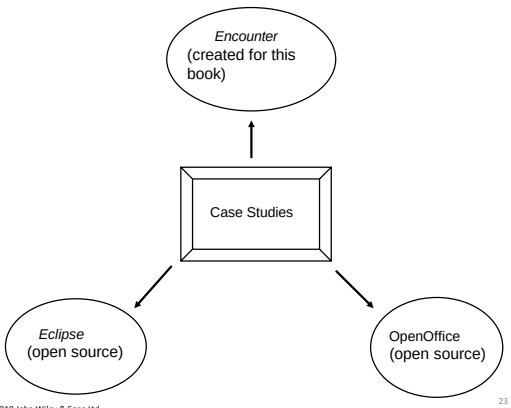
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

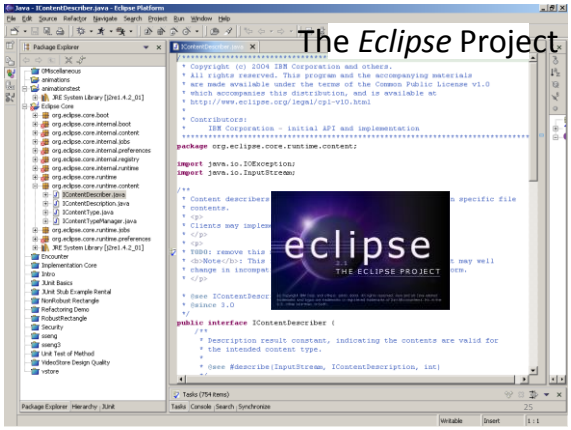
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

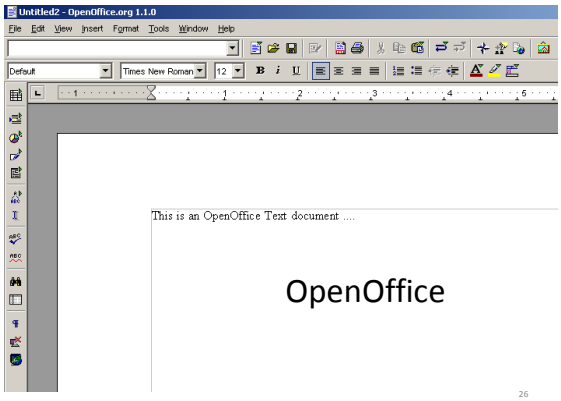
6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

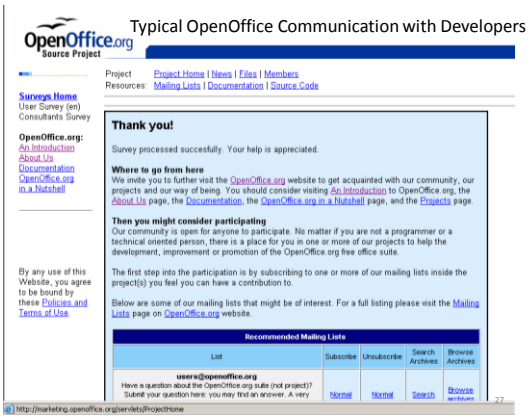
7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.









Project	Lead/Co-Lead	Short Name	Description
API	Michael Hoennig, Juergen Schmidt	api	The application programming interface.
Application Framework	Mathias Bauer, Carsten Driesner	framework	The framework for applications.
Build Tools and Environment	Martin Hollmichel	tools	The tools used in build process and the build environment.
Database Access	Frank Schoenheit, Dirk Grobler	dba	The database access for the applications.
Documentation	Scott Carr	documentation	End-user documentation for the various components making up OpenOffice.org.
External	Martin Hollmichel (acting)	external	This project will host all the external code allowed.
Graphic Applications	Kai Ahrens, Christian Lippka	graphics	The graphic applications such as Draw and Impress.
Graphic System Layer	Christof Pintaske, Hennes Rohling	gsl	The Visual Class Library and other modules.
Installation	Dirk Voelzke	installation	Creating the installation set.
Linguicomponent	Kevin Hendricks	lingucomponent	Creating dictionaries, thesauri, and other related tools.

OpenOffice Accepted Projects

http://projects.openoffice.org/accepted.html

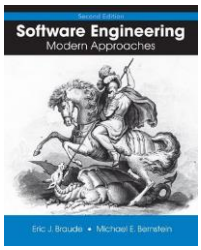
Localization	Dieter Loeschky, Nils Fuhrmann Pavel Janik	l10n	Localization. This project includes Internationalization (i18n).
Marketing	Jacqueline McNally	marketing	The project furthering the growth and use of OpenOffice.org technology. Efforts include: developing collateral, logos, public outreach.
Porting	Martin Hollmichel, Kevin Hendricks	porting	Porting to new platforms.
Quality Assurance	Michael Bemmer, Scott Carr, Gordon Shum	qa	Quality Assurance: testing and qualifying all builds of OpenOffice.org.
Spreadsheet	Niklas Nebel, Eike Rathke	sc	The spreadsheet application.
Universal Content Broker	Matthias Huetsch, Andreas Bille	ucb	Allows the applications to transparently access content with different structures.
UNO Development Kit / Component Technology	Kay Ramme, Kai Sommerfeld	udk	Object model development and component technology. Includes the old OI and Scripting projects.
User Interface	Oliver Specht	ui	Common user interface for OpenOffice.org applications.
Utilities	Hennes Rohling	util	Utilities used in development.
Website	Louis Suárez-Potts, Kay Schenk,	www	The OpenOffice.org website; the project for establishing the appearance of the Project.
Word Processing	Andreas Martens, Caolan McNamara	sw	The Word Processing Application

OpenOffice Accepted Projects 2

http://projects.openoffice.org/accepted.html

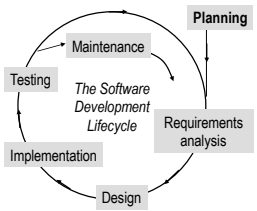
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 3: Software Process



Phase most relevant to this chapter is shown in bold

Learning goals of this chapter

- What are the main activities of software processes?
- What are the main software process types?
- How would a team go about selecting a process?

Software Process

- Software project composed of activities
  - E.g. planning, design, testing, etc.
- Activities organized into phases
- A *software process:*
  - prescribes the **order** and **frequency** of phases
  - specifies **criteria** for moving from one phase to the next
  - defines the **deliverables** of the project

Umbrella Activities

- Generic activities implemented ***throughout*** the life of a project (umbrella activities)
  - Project management
  - Configuration management
  - Quality management
  - Risk management

Software Process Benefits

- Process **DOES NOT** mean
  - "overhead"
  - "unnecessary paperwork"
  - "longer schedules"
  - etc.
- Software process has **positive** effect if applied correctly
  - Meet schedules
  - Higher quality
  - More maintainable

Software Process Phases

Phases of Software Processes

1. **Inception**
   Software product is conceived and defined
2. **Planning**
   Initial schedule, resources and cost are determined
3. **Requirements Analysis**
   Specify what the application must do; answers **"*what*?"**
4. **Design**
   Specify the parts and how they fit; answers **"*how*?"**
5. **Implementation**
   Write the **code**
6. **Testing**
   Execute the application with input test data
7. **Maintenance**
   Repair defects and add capability

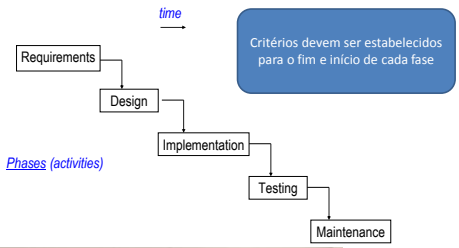
Example – Video Store Application

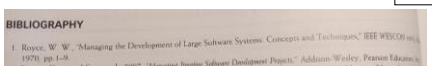
Software Process Phases: Video Store Example

- **Inception**
  " …An application is needed to keep track of video rentals …"
- **Planning** (Software Project Management Plan)
  "…The project will take 12 months, require 10 people and cost $2M…"
- **Requirements Analysis** (Product: Software Requirements Spec.)
  "…The clerk shall enter video title, renter name and date rented. The system shall …"
- **Design** (Software Design Document: Diagrams and text)
  " … classes *DVD*, *VideoStore*, …, related by …"
- **Implementation** (Source and object code)
  … class DVD{ String title; … } …
- **Testing** (Software Test Documentation: test cases and test results)
  "… Ran test case: *Rent "The Matrix" on Oct 3; rent "SeaBiscuit" on Oct 4; return "The Matrix" on Oct 10*…
  Result: *"SeaBiscuit" due Oct 4, 2004 balance of $8. (correct)* …"
- **Maintenance** (Modified requirements, design, code, and text)
  Defect repair: "Application crashes when balance is $10 and attempt is made to rent *"Gone With the Wind"*. …"
  Enhancement: "Allow searching by director."

Waterfall Process

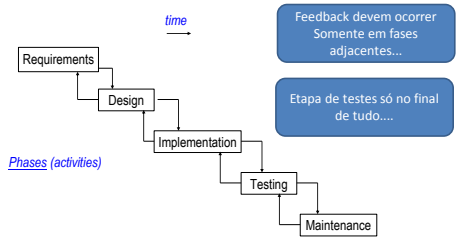
The Waterfall Software Process




BIBLIOGRAPHY
1. Royce, W. W., "Managing the Development of Large Software Systems: Concepts and Techniques," IEEE WESCON, 1970, pp. 1–9.

Waterfall Process – with feedback

The Waterfall Software Process with Feedback



Waterfall Process - Advantages

- Simple and easy to use
- Practiced for many years
- Easy to manage
- Facilitates allocation of resources
- Works well for smaller projects where requirements are very well understood

Waterfall Process - Disadvantages

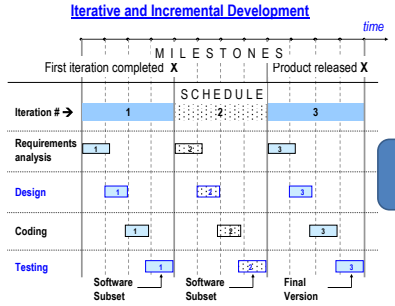
- Requirements must be known up front
- Hard to estimate reliably
- No feedback of system by stakeholders until after testing phase
- Major problems are discovered too late in process
- Lack of parallelism
  - Otherwise, disjoint parts could be completed in parallel…
- Inefficient use of resources (people are also resources)

Iterative and Incremental

- *Iterative*
  - repeated execution of the waterfall phases, in whole or in part, resulting in a refinement of the requirements, design and implementation
- *Incremental*
  - operational code produced at the end of an iteration
  - supports a subset of the final product functionality and features
- **Artifacts** evolve during each phase
- Artifacts considered complete only when software is released

Iterative and Incremental (cont.)

Iterative and Incremental Development



The "relased" is the final product !

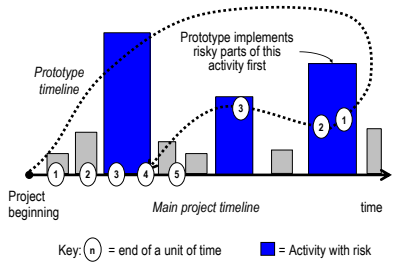
Release Types

- Proof of concept
  - Used to investigate the feasibility of a particular aspect of the software
- Prototype
  - A working version demonstrating a particular capability that is deemed to high risk.
- "Internal" release
- "External" release

Prototyping

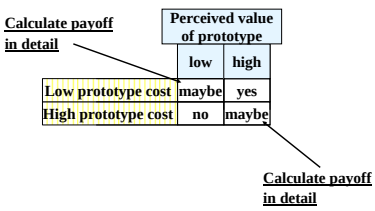
Protótipos podem ser construídos com objetivos diversos: interfaces ou avaliar grandes riscos.

Prototype Rationale



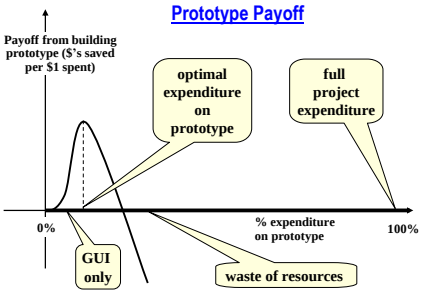
Prototyping (cont.)

Prototype Payoff: First Cut



Prototyping (cont.)

Prototype Payoff



1) Evitar tempo gasto no desenvolvimento de requisitos que o protótipo mostra que são desnecessários;
2) Retrabalho de mudar requisitos só depois que o cliente visualizasse o produto final.

Prototype Payoff Calculations for E-commerce Clothing Application

Valor que será economizado se o protótipo for construido. (ver fatores)

	Esti-mated cost	Gross Benefit excluding code re-use		Percentage of prototype code reused in application	Net Payoff		
		min	max		min	max	average
Prototype feature	*B*	*D*	*E*	*C*	*D-(1-C)B*	*E-(1-C)B*	
1. GUI screenshots	$10,000	$10,000	$80,000	50%	$5,000	$75,000	$40,000
2. Transaction security	$50,000	$10,000	$300,000	80%	$0	??	$145,000
3. Complete transaction	$80,000	$10,000	$400,000	50%	??	$200,000	165,000
4. Customer tries on clothing	$120,000	$20,000	$140,000	30%	-$64,000	$56,000	-$4,000
	260,000	50,000	920,000		- 89,000	781,000	346,000

Errado ! 360.000

Prototyping (cont.)

Prototype Payoff Calculations for E-commerce Clothing Application

	Esti-mated cost	Gross Benefit excluding code re-use		Percentage of prototype code reused in application	Net Payoff		
		min	max		min	max	average
Prototype feature	*B*	*D*	*E*	*C*	*D-(1-C)B*	*E-(1-C)B*	
1. GUI screenshots	$10,000	$10,000	$80,000	50%	$5,000	$75,000	$40,000
2. Transaction security	$50,000	$10,000	$300,000	80%	$0	$290,000	$145,000
3. Complete transaction	$80,000	$10,000	$400,000	50%	-$30,000	$200,000	$85,000
4. Customer tries on clothing	$120,000	$20,000	$140,000	30%	-$64,000	$56,000	-$4,000

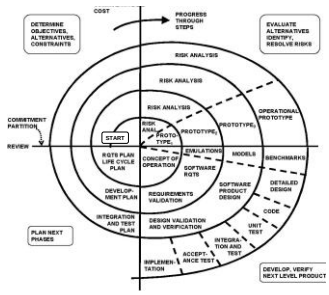
- Qual das features compensa mais e qual compensa menos ? Como julgar ?
- Qual o custo estimado de se implementar os quatro protótipos (com reuso) ?
- Considerando o valor max, quantos porcento o protótipo toma da receita ?

Spiral Model

- Barry Boehm, TRW Defense Systems,1988
- One of the earliest and best known iterative and incremental processes
- Risk-driven process
- Project starts at the center, and each cycle of the spiral represents one iteration
- Goal of each cycle is to increase the degree of system definition and implementation, while decreasing the degree of risk

Spiral Model

The Spiral Model



Spiral Model - Iteration Steps

1. Identify critical objectives and constraints
2. Evaluate project and process alternatives
3. Identify risks
4. Resolve (cost-effectively) a subset of risks using analysis, emulation, benchmarks, models and prototypes
5. Develop project deliverables including requirements, design, implementation and test
6. Plan for next and future cycles – update project plan including schedule, cost and number of remaining iterations
7. Stakeholder review of iteration deliverables and their commitment to proceed based on their objective being met

Spiral Model - Advantages

- ***Risks are managed early and throughout the process*** – risks are reduced before they become problematic
- ***Software evolves as the project progresses*** - errors and unattractive alternatives eliminated early.
- ***Planning is built into the process*** – each cycle includes a planning step to help monitor and keep a project on track.

Spiral Model - Disadvantages

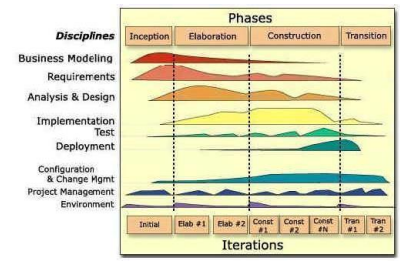
- ***Complicated to use*** - risk analysis requires highly specific expertise. There is inevitably some overlap between iterations.
- ***May be overkill for small projects*** – complication may not be necessary for smaller projects. Does not make sense if the cost of risk analysis is a major part of the overall project cost.

Unified Process

- Developed by Jacobson, Rumbaugh and Booch
- Major elaboration and refinement of Spiral
- Use-case driven
- Commercial product: Rational Unified Process (RUP)

Unified Process

Unified Process



http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

Inception

- Establish feasibility
- Make business case
- Establish product vision and scope
- Estimate cost and schedule, including major milestones
- Assess critical risks
- Build one or more prototypes

Elaboration

- Specify requirements in greater detail
- Architectural baseline
- Iterative implementation of core architecture
- Refine risk assessment and resolve highest risk items
- Define metrics
- Refine project plan, including detailed plan for beginning Construction iterations

Construction

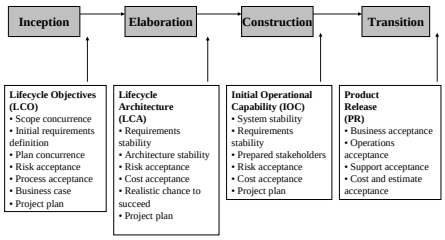
- Complete remaining requirements
- Iterative implementation of remaining design
- Thorough testing and preparation of system for deployment

Transition

- Beta tests
- Correct defects
- Create user manuals
- Deliver the system for production
- Training of end users, customers and support
- Conduct lessons learned

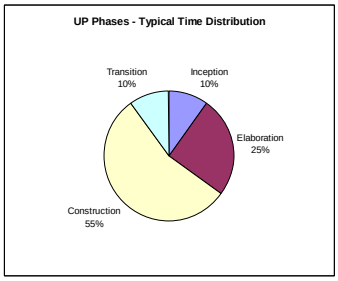
Unified Process Milestones

Unified Process Milestones



Adapted from http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

Phases Times



Adapted from: Ambler, S.W., *A Manager's Introduction to the Rational Unified Process (RUP)*

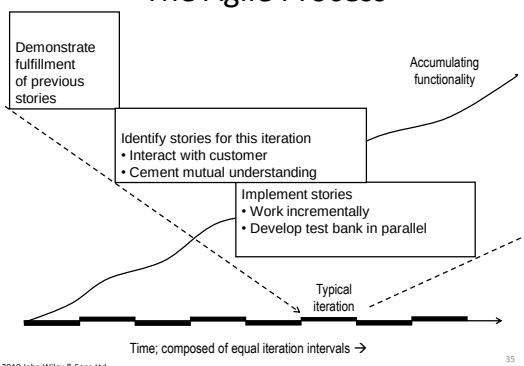
Agile Processes

The Agile Manifesto:

Agile processes value …

- … **individuals and interactions** over processes and tools
- … **working software** over comprehensive documentation
- … **customer collaboration** over contract negotiation
- … **responding to change** over following a plan

The Agile Process



Open Source

Reasons for Making a Project Open Source *1*

☺Leveraging large number of resources

☺Professional satisfaction

☺ To enable tailoring and integration

☺ Academic and research

☺ To gain extensive testing

☺ To maintain more stably

Richard Stallman
Originator of open source development
http://stallman.org/

Open Source (cont.)

Reasons for Making a Project Open Source *2*

☺To damage a competitor's product

☺To gain market knowledge

☺ To support a core business

☺ To support services

Open Source (cont.)

Reasons Against Open Source

☹ Documentation inconsistent or poor

☹No guarantee that developers will appear

☹No management control

☹No control over requirements

☹Visibility to competitors

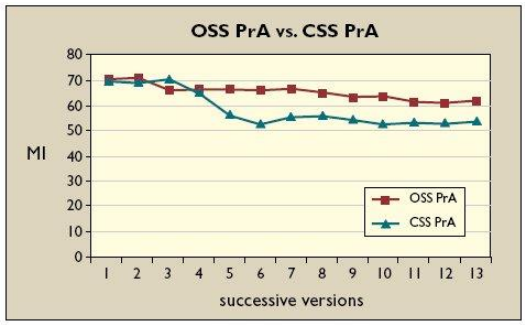
Bill Gates
http://www.microsoft.com/billgates/default.asp

Maintainability Index: OSS vs. CSS Same Application *2*

Project Mnemonic Code	Application Type	Total Code Size (KLOCs)	No. of releases measured	Project Evolution Path
OSSPrA	Operating system application	343	13	OSS project that gave birth to a CSS project while still evolving as OSS
CSSPrA	Operating system application	994	13	CSS project initiated from an OSS project and evolved as a commercial counterpart of OSSPrA

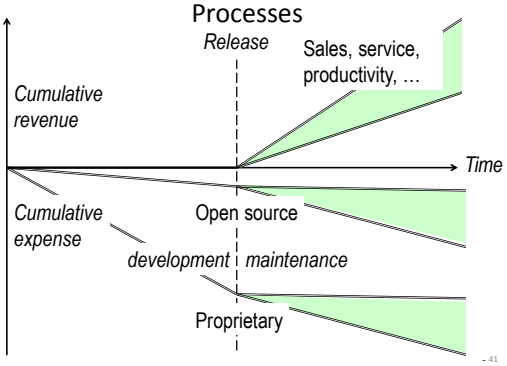
Communications of the ACM v 47, Number 10 (2004), Pp 83-87; Open source software development should strive for even greater code maintainability ; Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou

Maintainability Index: OSS vs. CSS Same Application *2*



Communications of the ACM v 47, Number 10 (2004), Pp 83-87; Open source software development should strive for even greater code maintainability ; Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou

Hybrid Open Source / Proprietary Processes



Collabnet Enterprise Edition

http://www.collab.net/products/index.html

Initial Student Team Meeting*: General Issues*

1. Set agenda and time limits.
2. Choose the team leader.
3. Get everyone's commitment to required time
   - Define an expected average number of hours per week
   - Gather dates of planned absences
4. Take a realistic census of team skills
   - Common problem: inflated programming skill claims
5. Begin forming a vision of the application
6. Decide how team will communicate.
7. Take meeting minutes with concrete action items

Communication Precepts

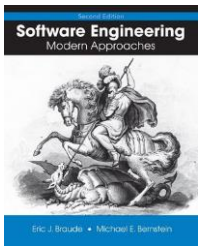
1. Listen to all with concentration
   - Don't pre-judge
2. Give all team members a turn
   - See the value in every idea
3. Don't make assumptions
   - Ask questions to clarify
4. When in doubt, communicate

Communication Plan

1. Meetings: Team will meet each Monday from ... to ... in …
   Caveat: do not replace face-to-face meeting with remote meetings unless remote meetings are clearly effective.
2. Meeting alternative: Team members should keep Fridays open from ... to ... in case an additional meeting is required.
3. Standards: Word processor, spreadsheet, compiler, ….
4. E-mail: Post e-mails?; require acknowledgement?
   Caveat: e-mail is poor for intensive collaboration
6. Collaboration: Tools for group collaboration and discussion –
   e.g. Yahoo Groups, Wiki tool, Google tools, …
7. Other tools: Microsoft Project (scheduling), Group calendar, …

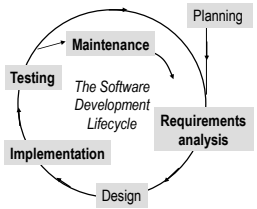
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 4: Agile Software Processes



Phases most relevant to this chapter are shown in bold

Learning goals of this chapter

- How did agile methods come about?
- What are the principles of agility?
- How are agile processes carried out?
- Can agile processes be combined with non-agile ones?

Agile Processes

- Prior to 2001, group of methodologies shared following characteristics:
  - Close collaboration between programmers and business experts
  - Face-to-face communication (as opposed to documentation)
  - Frequent delivery of working software
  - Self-organizing teams
  - Methods to craft the code and the team so that the inevitable requirements churn was not a crisis

Agile Alliance – Agile Manifesto

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile 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.

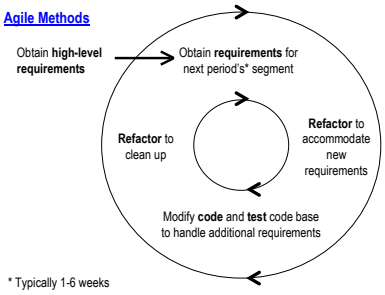
Agile Principles

- 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.

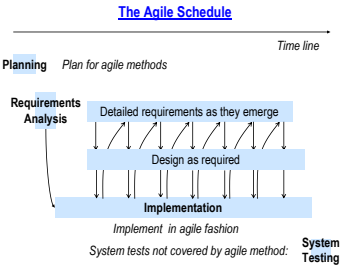
Agile Methods

Agile Processes MANIFESTO → RESPONSES:	1. Individuals and interactions over processes and tools	2. Working software over comprehensive documentation	3. Customer collaboration over contract negotiation	4. Responding to change over following a plan
a. Small, close-knit **team** of **peers**	y			y
b. Periodic **customer** requirements **meetings**	y		y	y
c. **Code-centric**		y		y
d. **High-level** requirements statements only			y	y
e. **Document as needed**			y	y
f. **Customer reps** work within team	y			y
g. **Refactor**				y
h. **Pair** programming and no-owner code	y			
i. Unit-test-intensive; Acceptance-**test-driven**		y	y	
j. **Automate** testing		y	y	

Agile Cycle



Agile Schedule



Extreme Programming (XP)

- Kent Beck, 1996
- Project at Daimler Chrysler
- Simple and efficient process

XP Values

The "Values" of Extreme Programming *1 of 2*

1. **Communication**
   - Customer on site
   - Pair programming
   - Coding standards
2. **Simplicity**
   - Metaphor: entity names drawn from common metaphor
   - Simplest design for current requirements
   - Refactoring

Beck: Extreme Programming Explained

XP Values (cont.)

The "Values" of Extreme Programming *2 of 2*

3. **Feedback** always sought
   - Continual testing
   - Continuous integration (daily at least)
   - Small releases (smallest useful feature set)
4. **Courage**
   - Planning and estimation with customer user stories
   - Collective code ownership
   - Sustainable pace

Beck: Extreme Programming Explained

XP Principles

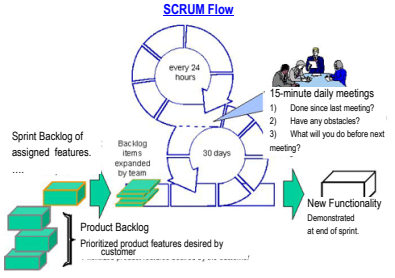
- Planning Process
- Small Releases
- Test-driven Development
- Refactoring
- Design Simplicity
- Pair Programming
- Collective Code Ownership
- Coding Standard
- Continuous Integration
- On-Site Customer
- Sustainable Pace
- Metaphor

SCRUM

- Developed in early 1990s
- Based on assumption:
  - development process is unpredictable and complicated
  - can only be defined by a loose set of activities.
- Development team empowered to define and execute the necessary tasks to successfully develop software

SCRUM

SCRUM Flow



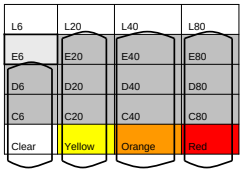
Quoted and edited from http://www.controlchaos.com/

Crystal

- Alistair Cockburn
- Family of agile methods
- Frequent delivery, close communication and reflective improvement

Crystal

Coverage of Different Crystal Methodologies



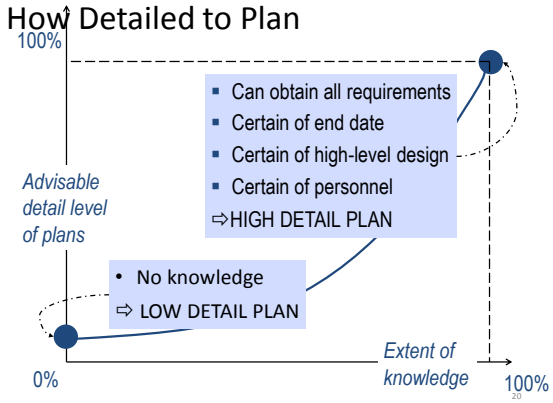
L = loss of life
E = loss of essential moneys
D = loss of discretionary moneys
C = loss of comfort

Adapted from Crystal Clear, Alistair Cockburn

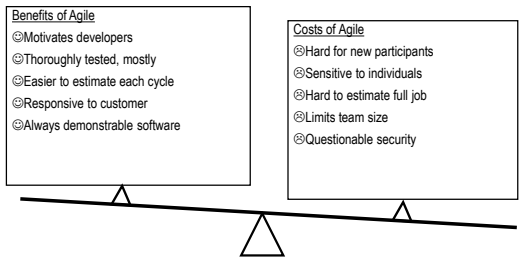
Crystal Properties

- Frequent delivery
- Reflective improvement
- Close communication
- Personal safety
- Focus
- Easy access to expert users
- Technical environment with automated testing, configuration management and frequent integration

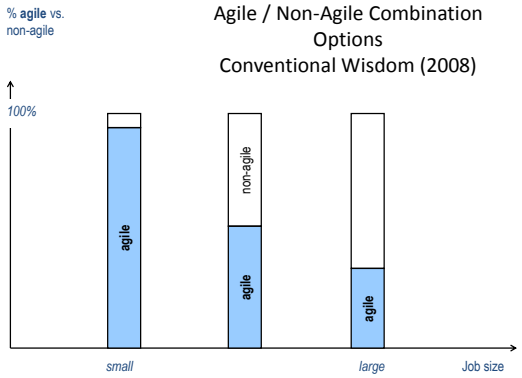
How Detailed to Plan



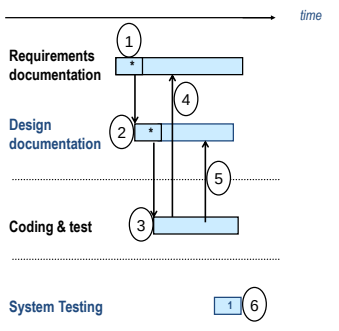
Agile / Non-Agile Tradeoff



Agile / Non-Agile Combination Options Conventional Wisdom (2008)

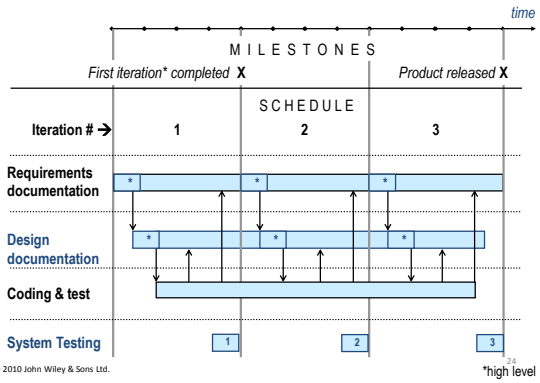


Integrating Agile with non-Agile Methods *1: Time Line*



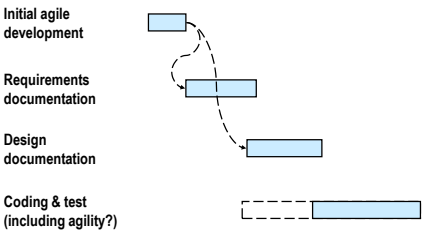
* High level

Integrating Agile with non-Agile Methods *2*: Iterations



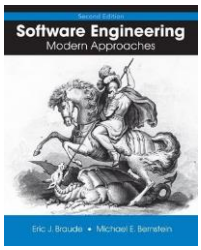
*high level

Agile-Driven Approach to Large Jobs: Each Iteration



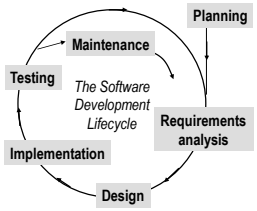
Software Engineering

Modern Approaches



Eric Braude and Michael Bernstein

Chapter 5: Quality in Software Process



Phases relevant to this chapter are shown in bold

Learning goals of this chapter

- What are principles of managing quality?
- How do you plan for "quality?"
- What are inspections and how do you carry them out?
- How do Quality Assurance personnel carry out reviews and audits?
- How do you measure and improve software processes?
- In what way does CMMI assess organizational quality?
- What does a software quality plan look like for a case study?

Quality in the Software Process

- Quality principles
  - overarching quality guidelines
- Quality planning
  - quality plan defines overall approach to quality
- Inspections
  - peer processes focused on quality
- Reviews and Audits
  - external quality assessment
- Defect management
  - identification, tracking and resolution of defects
- Process improvement
  - continuous upgrading of process effectiveness
- Organizational quality
  - engineering competence levels (e.g. CMMI/MPS.Br)

Quality Principles

- Focus continuously on quality
- A quality assurance process must be defined
- The organization must follow its quality assurance process
- Find and repair defects as early in the development process as possible

Defect Repair Cost

Reason (by one estimate):	If defect found ...	
	… soon after creation	… at integration time
Hours to ..		
.. detect	0.7 to 2	0.2 to 10
.. repair	0.3 to 1.2	9+
Total	**1.0 to 3.2**	**9.2 to 19+**

Needed from a Quality Plan

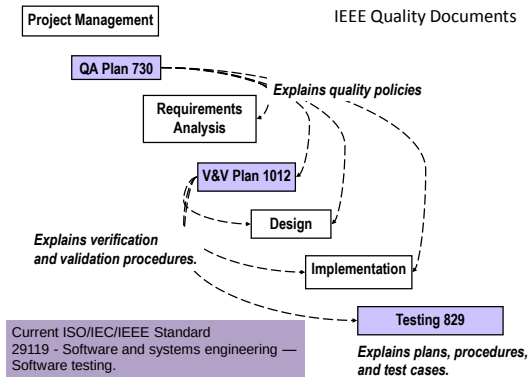
- **Who** will be responsible for quality?
  A person, a manager, a group, an organization, etc.
- What **documentation** will be generated to guide development, verification and validation, use and maintenance of the software?
- What **standards** will be used to ensure quality?
  Documentation standards, coding standards, etc.
- What **metrics** will be used to monitor quality?
  Product and process metrics.
- What **procedures** will be used to **manage** the quality process?
  Meetings, audits, reviews, etc.
- What kind of **testing** will be performed?
- What quality assurance **techniques** will be used?
  Inspections, proofs of correctness, tests, etc.
- How will **defects** be handled?

QA Personnel

QA Person vs. Developer Person-Hours

- 1 QA person per 3-7 developers
- Excludes developer testing
  counted as developer time
- Includes post-developer testing
- Ideally performed by external QA personnel

IEEE Quality Documents



Current ISO/IEC/IEEE Standard 29119 - Software and systems engineering — Software testing.

IEEE Quality Assurance Plan

IEEE 730-2002 Software Quality Assurance Plan
Table of Contents *1 of 2*

IEEE 730-2014 - IEEE Standard for Software Quality Assurance Processes

IEEE Quality Assurance Plan (cont.)

IEEE 739-2002 Software Quality Assurance Plans
Table of Contents *2 of 2*

Inspections

- Quality technique - focus on reviewing the details of a project artifact (requirements, designs, code etc.)
- Note: *not just code*
- Purpose: assure artifact's correctness by seeking defects.
- A meeting of inspectors is held at which defects are identified.

Inspections – Guiding Principles

- Peer process
- Specified roles – strong moderator
- Artifact readiness
- Adequate preparation
- Defect *detection* instead of defect *repair*
  - don't design on the fly
- Use of checklists
- Metrics collection
- Record action items for follow up
- Time limit

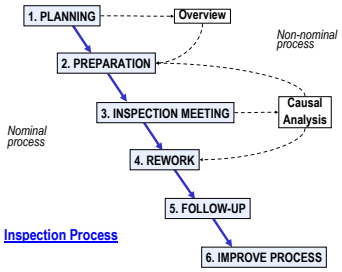
Inspection Roles

- Moderator
- Author
- Recorder
- Reader
- Inspector

Inspection Metrics

- Number of defects discovered, by severity and type
- Number of defects discovered by each category of stakeholder inspecting the artifact
- Number of defects per page reviewed
- Review rate (number of pages/hour)

Inspection Process



Inspection Process

Inspection Costs

Time/Costs per 100 Non-Comment Lines of Code

one company's estimates

Planning	**1 hr**	**×**	**(1 person)**
[Overview	*1 hr*	×	*(3-5 people)]*
Preparation	**1 hr**	**×**	**(2-4 people)**
Inspection meeting	**1 hr**	**×**	**(3-5 people)**
Rework	**1 hr**	**×**	**(1 person)**
[Analysis	*1 hr*	×	*(3-5 people)]*
Total:	**7 - 21 person-hours**		

Options for QA Reviews

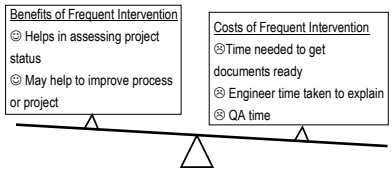
- Participate in all meetings
  Including formative sessions and inspections
- Review all documents
  Participate in all inspections
  (but do not attend all meetings)
- Attend final reviews and review all completed documents
- Review select completed documents
  But do not participate otherwise

Options for QA Audit

- Audit at unrestricted random times
  - Includes visiting with engineers
  - Includes inspecting any document at any time
- Audit random meetings
- Audit randomly from a specified list of meetings
- Audit with notice
- No auditing

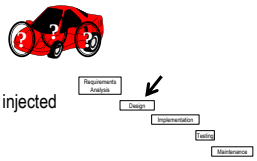
QA Intervention

Frequent vs. Occasional QA Intervention



Overall Defect Classification

- **Severity**
  How serious ☹
- **Priority**
  Order in which defects will be repaired
- **Type**
  What kind of problem
- **Source**
  - Phase during which injected



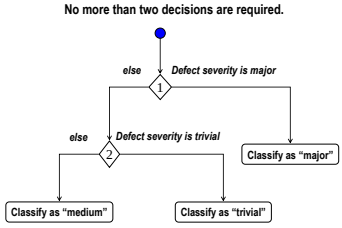
Defect Severity

Triage Severity Classifications

- **Major**
  Causes a requirement to be unsatisfied
- **Medium**
  Neither *major* nor *trivial*
- **Trivial**
  Defect, but doesn't affect operation or maintenance

Defect Severity Triage

Triage Decision Method Applied to Defect Severity Classification



IEEE Severity Classification

IEEE Severity Classification (1044.1)

- **Urgent**
  Failure causes system crash, unrecoverable data loss; or jeopardizes personnel
- **High**
  Causes impairment of critical system functions, and no workaround solution does exist
- **Medium**
  Causes impairment of critical system functions, though a workaround solution does exist
- **Low**
  Causes inconvenience or annoyance
- **None**
  None of the above

Common Defect Types

Common Defect Types Across All Artifacts

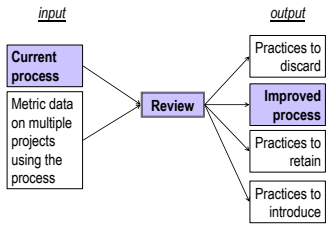
- **Omission**
  Something is missing.
- **Unnecessary**
  The part in question can be omitted.
- **Non-conformance with standards**
- **Inconsistency**
  The part in question contradicts other part(s).
- **Unclassified**
  None of the above

Defect Tracking

Defect Tracking								
Name	**Description**	**Discov-ering enginr.**	**Respon-sible enginr.**	**Date opened**	**Source**	**Seve-rity**	**Type**	**Status**
Check-out flicker	Checkout screen 4 flickers when old DVDs are checked out by hitting the Checkout button.	Kent Bain	Fannie Croft	1/4/04	Inte-gration	Med	GUI	Being worked begun 2/10/04
Bad fine	Fine not correct for first-run DVD's checked out for 2 weeks, as displayed on screen 7.	Fannie Croft	April Breen	1/4/06	Re-quire-ments	High	Math	Not worked yet
…	…	…	…	…	…	…	…	Tested with
…	…	…	…	…	…	…	…	Resolved
…	…	…	…	…	…	…	…	…

Process Improvement

The *Process Improvement* Meta-Process



Measure Process Effectiveness

Process →	Waterfall	Waterfall + Incremental	Pro-cess U	Pro-cess V
Average over 10 projects:				
Major defects identified within first 3 months per 1000SLOC in delivered product	1.3	0.9	0.7	2.1
Development cost per detailed requirement	$120	$100	$85	$135
Developer satisfaction index (1 to 10=best)	4	3	4	3
Customer satisfaction index (1 to 10=best)	4	6	6	2
Cost per maintenance request	$130	$140	$95	$165
Variance in schedule on each phase: 100 × *actual duration - projected duration / projected duration*	+20%	+70%	-10%	+80%
Variance in cost:: 100 × *actual cost - projected cost / projected cost*	+20%	+65%	-5%	+66%
Design fraction: *total design time / total programming time* Humphrey: Should be at least 50%.	23%	51%	66%	20%

A process for Gathering Process Metrics

1. **Identify & define** metrics team will use by phase; include
   ... time spent on 1. research, 2. execution, 3. review
   … size (e.g. lines of code)
   … # defects detected per unit (e.g., lines of code)
   include source
   … quality self-assessment of each on scale of 1-10
   maintain bell-shaped distribution
2. **Document** these in the SQAP
3. **Accumulate** historical **data** by phase
4. Decide **where** the metric data will be **placed**
   - as the project progresses SQAP? SPMP? Appendix?
5. **Designate engineers** to manage collection by phase
   - QA leader or phase leaders (e.g., design leader)
6. **Schedule reviews** of data for lessons learned
   - Specify when and how to feed back improvement

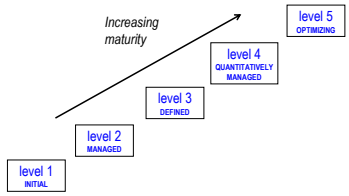
Requirements Document: 200 detailed requirements	Meeting	Research	Execution	Personal Review	Inspection
Hours spent	0.5 x 4	4	5	3	6
% of total time	10%	20%	25%	15%	30%
% of total time: norm for the organization	*15%*	*15%*	*30%*	*15%*	*25%*
Self-assessed quality 1-10	2	8	5	4	6
Defects per 100	N/A	N/A	N/A	5	6
Defects per 100: organization norm	*N/A*	*N/A*	*N/A*	*3*	*4*
Hours spent per detailed requirement	0.01	0.02	0.025	0.015	0.03
Hours spent per detailed requirement: organization norm	0.02	0.02	0.04	0.01	0.03
Process improvement	**Improve strawman brought to meeting**		**Spend 10% more time executing**		**Project Metric Collection for Phases**
Summary	**Productivity: 200/22 = 9.9 detailed requirements per hour**				

CMMI

- Software Engineering Institute (SEI) 1980s
- Measure process capability and maturity
- Two kinds of assessments: *staged* and *continuous*
- Builds on a long history of use that includes case studies and data that demonstrate return on investment

CMMI Staged

***CMMI* Model for Organization with Staged Processes**



CMMI Maturity Levels

Level, Title, and Summary	Expected Outcome	Characteristics
1. INITIAL		
Undefined; ad hoc	Unpredictable; depends entirely on team individuals.	Organizations often produce products, but they are frequently over budget and miss their schedules.
2. MANAGED	Preceding level plus:	
Measurement and control	Project outcomes are qualitatively predictable	Respect organizational policies Follow established plans Provide adequate resources Establish responsibility and authority Provide training Establish configuration management Monitor and control: Take corrective action Evaluate process and product relative to plans: Address deviations

CMMI Maturity Levels

3. DEFINED	Preceding level plus:	
Processes standardized	Projects consistent across the organization; qualitatively predictable	Establish process objectives Ensure process objectives met Establish orderly tailoring Describe processes rigorously Be proactive
4.QUANTI-TATIVELY MANAGED	Preceding level plus:	
Processes measured	Metrics available on process; quantitatively predictable	Set quantitative goals for key sub-processes Control key sub-processes with statistical techniques. Identify and remedy variants
5. OPTIMIZING	Preceding level plus:	
Improvement meta-process	Processes improved and adapted using process metrics	Establish quantitative process improvement objectives Identify and implement innovative process improvements Identify and implement incremental process improvements Evaluate process improvements against quantitative objectives.