The Discipline of Software Engineering

A discussion on basic best practices in software engineering

Mark Doernhoefer
The MITRE Corporation
mdoernho@mitre.org
8/26/2014

The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

Approved for Public Release; Distribution Unlimited. Case Number 14-2258.
The literature abounds with examples of software development projects that are late, over budget, unreliable, and/or vulnerable to attack

- The Standish Group, authors of the Chaos Study, report that 61% of projects are challenged or failed\textsuperscript{[1]}
- Capers Jones estimates large project cancellation rate at 31%, costing over $35M per project\textsuperscript{[2]}
- The Risks Digest chronicles software failures on a daily basis\textsuperscript{[3]}
- Healthcare.gov was launched with programming errors and performance problems despite spending millions in development

The root cause of some of these problems can be traced to management and budget issues including poor planning and cost estimation, but some shortcomings can be attributed to software project managers and the software engineering staff failing to follow good software engineering practice, in short, a failure to apply a disciplined software engineering process.
Why is this Important?

There are so many studies attempting to quantify the cost of software failures. They don’t agree on percentages but they generally agree that the number is at least 50 to 80 billion dollar range annually.\[4\]

--- Dan Galorath
Galorath Inc.

Software is on or impacting the critical path in almost 25% of major systems in DoD’s ~$1.5 trillion acquisition portfolio.\[5\]

--- Lloyd K. Mosemann II
2013 IEEE STC Keynote Address

Unless you've had your head buried under a rock for the past year, you've noticed high-profile software systems are crashing left and right -- costing companies millions in maintenance costs and emergency patches. But as the glitches keep coming, we’re starting to see the true cost of these glitches rise to the surface: CIOs’ job security.\[6\]

--- Vincent DeLaroche
CEO, CAST
Recent Observations

I don’t know objectively whether we are progressing, regressing, or standing still. Within the context of the special issue on software quality, I think the assessment is probably pretty dismal, as nearly all remnants of high-quality software have been removed from the marketplace and any processes that created high quality—in training, in education, and among developers—is also gone. How many courses are there on “preventing software errors”? Especially compared with the number of courses on a specific language, such as JavaScript, that is error-prone by design? Or, in the acquisition community, how many courses are they are on how to buy high-quality software?[6]

-- Stan Rifkin
Letters to the Editor
IEEE Software March/April 2014

It’s like déjà vu all over again.
-- Yogi Berra
Of Course you Can Just Give Up

- Takipi* offers a product to monitor error trending in production code and provide assistance with debugging applications in production
  - Specific support for finding invalid casts, null pointers, out of bounds indexes, illegal arguments, threading errors, etc.

- Whatever happened to debugging the code before delivery?
  - Why is a product like this needed?

*See: http://www.takipi.com/

If the world was perfect, it wouldn’t be.
-- Yogi Berra
But There is a Better Way – Discipline

- Lack of discipline – failure to follow the processes required by standards and best practice
  - Process has gotten a bad reputation as simply a means to achieve a certification such as CMMI or ISO
  - Process is seen as heavyweight, bureaucratic overhead that impedes progress
    - In fact, process should form the basis for a disciplined approach
  - The result is an incomplete and inconsistent application of software engineering development standards and best practice
Red Team Experience

- MITRE is often asked to “Red Team” a program or project in trouble
  
  You can observe a lot by watching.
  -- Yogi Berra

- All too often these Red Teams find the same issues in different programs with different sponsors
  - Most of these projects have already experienced cost overruns, schedule slips or system failures
  - Many of the root causes of the problems are either failure to account for an IEEE 12207 process or failure to follow best practice in an IEEE 12207 process area

- When interviewed, the most common reason why a process area was not addressed is that the program management felt that the process did not apply to their project

  This point of view is not new...
The Little Book of Bad Excuses

- Published in 1998 by the Software Program Manager’s Network (SPMN)
- Written by the members of the Airlie Software Council
- Provides a tongue-in-cheek documentation of the common lapses in software discipline
- Example excuses (all numbered in the book):
  - On requirements: We can't ask the user; we're too far down the road.
  - On process: I don’t need good process because I have good people.
  - On testing: We have formal reviews to find problems.
  - On CM: CM only applies to documents.
  - On CM²: CM only applies to source code.
- The book is as relevant today as it was in 1998 – we’re seeing the same things

We made too many wrong mistakes.

-- Yogi Berra
What Do You Mean by “Discipline”? 

Discipline can refer to a field of study and a behavior whereby participants conform to a set of rules or practices and can also refer to punishment for not following the rules.

**dis·ci·pline** [dis-uh-plin] noun
1. training to act in accordance with rules; drill: military discipline.
2. activity, exercise, or a regimen that develops or improves a skill; training: A daily stint at the typewriter is excellent discipline for a writer.
3. punishment inflicted by way of correction and training.
4. the rigor or training effect of experience, adversity, etc.: the harsh discipline of poverty.
5. behavior in accord with rules of conduct; behavior and order maintained by training and control: good discipline in an army.

We are using just two of the definitions: The discipline of software engineering and the ability to act in accordance with the standards and best practices in the field of software engineering.
Where is this Discipline Defined?

- IEEE Standard 12207 Systems and software engineering — Software life cycle processes
  - Provides a simple, straightforward description of the processes necessary for any software development
  - Only describes the processes and how the processes apply to the overall development effort
  - IEEE 12207 will not tell you how to develop software

- The Capability Maturity Model Integration (CMMI)® provides models for best practices in development, acquisition, and services related to software engineering
  - Further describes process goals and practices
  - Provides a basis for measuring process results

- The Information Technology Infrastructure Library (ITIL)® also provides processes, procedures, and tasks that support software service development and management

CMMI® is a registered trademark of Carnegie Mellon University
ITIL® is a registered trademark of AXELOS Limited
Where are Best Practices Defined?

- The Software Engineering Body of Knowledge (SWEBOK) provides pointers to best practices in software engineering literature
  - The SWEBOK Guide correlates best practices to IEEE 12207 processes
  - Contained in the works of Pressman, McConnell, Brooks, Sommerville, Page-Jones, Boehm, Wiegers, Clements, etc.

- Educational institutions use standardized software engineering instruction
  - Software Engineering 2004 Curriculum (undergrad)
  - Graduate Software Engineering 2009 (GSwE 2009)

- We have case studies that document best practice
  - Software Engineering Institute[8]
  - Sommerville web site[9]
IEEE 12207: A Closer Look (1 of 2)

- The Standard Software life cycle processes
  - Section 6 System Life Cycle Processes
  - Section 7 Software Specific Processes
- Clause 7.1.1 Software Implementation Process
  - Software Requirements Analysis Process (7.1.2)
  - Software Architectural Design Process (7.1.3)
  - Software Detailed Design Process (7.1.4)
  - Software Construction Process (7.1.5)
  - Software Integration Process (7.1.6)
  - Software Qualification Testing Process (7.1.7)
- Each process section lists the tasks and activities conducted when executing the process as well as the expected outcomes
  - Whether you know it or not, you will be executing all of the processes listed as you develop your application
  - It is better to plan your processes instead of making it up as you go along
Three important aspects:

1. **IEEE 12207** defines software development activities that are always performed and discusses the individual tasks associated with those activities.

2. **IEEE 12207** *does not* describe how to perform the tasks or provide recommended methodologies, techniques, strategies, tools, ...
   - For any specific project it is necessary to:
     1. Determine the nature of the system to be developed
     2. Determine where you are starting (new development, reuse, COTS integration, ...)
     3. Identify the techniques most appropriate for the project
     4. Determine the life cycle model (i.e., the scheduling and interrelationships among the activities and tasks) to be used
     5. Document this in your plans
     6. Adhere to the plan, but change it when process improvements are necessary.

3. **Do not** automatically adhere to a predefined process defined in a corporate generic plan. Evaluate corporate standards for suitability to the system under development. Just because your generic process might be CMMI 5, does not mean it is best for your system.
Software Implementation Process (7.1.1)

Outcomes:
- an implementation strategy is defined;
- implementation technology constraints on the design are identified;
- a software item is realized; and
- a software item is packaged and stored in accordance with an agreement for its supply.

- Failure to plan for the processes listed in IEEE 12207
  - Development processes made up as the project progresses
  - Critical later phases of the development life cycle such as deployment and operation unaccounted for

- Failure to scope the project in terms of cost and schedule
  - Expectation mismatch between developer and customer
  - Failure to allocate resources to complete the project

*If you don’t know where you are going you might not get there.* -- Yogi Berra
Outcomes:

a) the requirements allocated to the software elements of the system and their interfaces are defined;
b) software requirements are analyzed for correctness and testability;
c) the impact of software requirements on the operating environment are understood;
d) consistency and traceability are established between the software requirements and system requirements;
e) prioritization for implementing the software requirements is defined;
f) the software requirements are approved and updated as needed;
g) changes to the software requirements are evaluated for cost, schedule and technical impact; and
h) the software requirements are baselined and communicated to all affected parties.

- Failure to record and manage requirements
  - Misunderstanding about what the software is supposed to do
  - Inability to write test cases based on required functionality
  - Inability to assess impact of requirements changes and/or control scope creep
Software Architectural Design Process (7.1.3)

Outcomes:
- a) a software architectural design is developed and baselined that describes the software items that will implement the software requirements;
- b) internal and external interfaces of each software item are defined; and
- c) consistency and traceability are established between software requirements and software design.

- Failure to document and review the design
  - System functions not fully allocated to system components
  - Misunderstanding as to whether a specific function will be performed by hardware or software
  - Lack of an architectural software design leads to lack of cohesion and increased coupling
  - Late discovery of interface needs
Software Detailed Design Process (7.1.4)

Outcomes:

a) a detailed design of each software component, describing the software units to be built, is developed;

b) external interfaces of each software unit are defined; and

c) consistency and traceability are established between the detailed design and the requirements and architectural design.

- Failure to document and review the design
  - Leads to poor software architecture, increased redesign and refactoring during Construction
  - Often results in functional duplication across large programs
  - Design reviews perfunctory, “check the box” activity
  - Design not baselined and understood for future development work
  - Lack of design detail and traceability hampers testing
Software Construction Process (7.1.5)

Outcomes:
- a) verification criteria are defined for all software units against their requirements;
- b) software units defined by the design are produced;
- c) consistency and traceability are established between software units and requirements and design; and
- d) verification of the software units against the requirements and the design is accomplished.

- Failure to plan for construction
  - No coding standards
  - No unit testing standards
  - Lack of peer reviews or code walkthroughs
  - Unclear as to when, how, or what governs completion of coding
  - Lack of traceability of requirements to code causes FCA issues
Software Integration Process (7.1.6)

Outcomes:

a) an integration strategy is developed for software units consistent with the software design and the prioritized software requirements;
b) verification criteria for software items are developed that ensure compliance with the software requirements allocated to the items;
c) software items are verified using the defined criteria;
d) software items defined by the integration strategy are produced;
e) results of integration testing are recorded;
f) consistency and traceability are established between software design and software items; and
g) a regression strategy is developed and applied for re-verifying software items when a change in software units (including associated requirements, design and code) occur.

- Failure to plan for integration
  - Excessive time spent in testing and re-testing because the work was not planned
  - Incomplete integration testing causes surprises during qual testing
Software Qualification Testing Process (7.1.7)

Outcomes:

a) criteria for the integrated software is developed that demonstrates compliance with the software requirements;
b) integrated software is verified using the defined criteria;
c) test results are recorded; and
d) a regression strategy is developed and applied for re-testing the integrated software when a change in software items is made.

• Failure to define qualification testing
  ◦ Lack of clarity on how to test and how to record test results
  ◦ Lack of resources (testbed, test data, etc.)
  ◦ Tests not traced to requirements, so certain features may not be tested
  ◦ Weak testing leads to unreliable delivered software
The Prescription: Practice Basic Software Engineering Discipline

- Learn and use the process standards
  - In particular, IEEE 12207 Section 6, System Life Cycle Processes and Section 7, Software Specific Processes
- Study the industry best practice
  - CMMI and SWEBOK references
- Determine applicable processes and practices
- Plan your development and document your plans
  - Program Management Plan
  - Systems Engineering Plan
  - Software Development Plan
- Monitor your development and measure your results
  - Institute continuous process improvement
  - Change the processes that don’t work
- Enforce the process
  - All process deviations must be accompanied by solid rationale
Practical Considerations

- This is *not* a recommendation to establish heavyweight, documentation intensive processes
  - It is a recommendation to plan your process up front, before you begin development
  - The activities should be designed with lean in mind
  - Process should not slow you down, it should speed up your development and improve your ability to estimate cost and schedule
- Documentation should be viewed as a byproduct of a disciplined approach
  - Do not develop documentation unless there is a use or potential use for the document
  - Avoid duplication of documentation
- Apply continuous process improvement throughout the life cycle
The Bottom Line

- If you think you are special: You’re not
  - See the Little Book of Bad Excuses
- Every departure from a disciplined process adds risk
  - Can you get lucky? Sure. Play the lottery.
- Do you have to plan everything out in fine detail?
  - No, but you do need a plan and you need to plan for resources
  - You also need to communicate that plan and perhaps even train the workforce

The future ain’t what it used to be.
--- Yogi Berra

- Software process discipline must be applied across the entire software life cycle because....
It ain’t over til it’s over.
Selected Detailed Disciplinary Infractions and References
Requirements Discipline

- Failure to record requirements
  - Unable to document system capabilities
  - Can’t develop test cases tied to a system requirement
  - Can’t do a functional configuration audit
  - Can’t establish a functional baseline

- Failure to manage requirements
  - No systematic approach for approving requirements changes
  - No analysis of impact for prospective changes
  - Can’t recall why functional changes were made
  - Scope creep

- Other issues
  - Requirements management tool used as a repository, not as an engineering tool
  - Vague, poorly written requirements
Software Design Discipline

- Failure to document the design
  - “Back of the envelope” or mental notes of software architecture
  - Can’t identify system components and their interfaces
  - Can’t establish an allocated baseline
    - No way of ensuring all functions mapped to software components

- Failure to review the design
  - Hard to achieve reuse of common design patterns
  - Critical for concurrent systems

- Failure to manage design changes
  - Lack of impact analysis for proposed design changes
  - “Moving Target” syndrome
Code and Unit Test Discipline

- Lack of coding standards
  - Hinders maintainability and understandability
  - No uniformity in the product
  - Limits the use of design patterns, which can be a real time saver
- No unit testing guidance provided to developers
  - Without test coverage guidelines, low level testing is likely to be incomplete
  - Testing tends to take longer as testers have to determine how much testing is enough for each unit
- Lack of peer review during code and unit test
  - Studies have shown that peer review is the most effective way of capturing defects within the construction phase, particularly when assisted by static analysis tools\(^2\)[10][11]
Software Testing Discipline

- Failure to develop test cases that trace to requirements
  - Test cases don’t test the function as specified
  - “Test to pass” philosophy versus testing to identify failures
    - Results in poor software reliability when exposed to unexpected conditions

- Failure to test the system under expected load
  - Results in unexpected behavior and performance issues when software is deployed in a production environment
  - Typically error handling induces additional load, can lead to cascading failures

- Software deployed although it failed formal testing
  - “The formal test was only to characterize the performance of the software”
  - There should be no issues discovered during the formal test event
Necessary Support Processes

- No Configuration Management (CM)
  - CM not planned in advance, CM process evolves with the evolving product
- No Quality Assurance (QA)
  - Independent set of eyes inspecting your product and process provides fresh insights into issues
- No risk management
  - Risk mitigation activities not realistic or not tracked for accomplishment
- No deployment strategy
  - How do you start up the new software while retiring old systems?
  - How to you manage data migration?
- No maintenance strategy
  - How do you patch the software in the field?
- No Continuity Of Operations (COOP) or disaster planning
References (1 of 2)


References (2 of 2)


