CS 314, Colorado State Univ. Software Engineering Notes 1: Software Products & Development Processes

James M. Bieman
with contributions from Geri Georg

What Is Software?

- The non-physical manifestation of information:
  Books, music, movies, your genetic code, your bicycle lock combination, the Linux operating system (or any program).
- The software media (flash drive, DVD, etc.) is not software.
  Do we even need media?

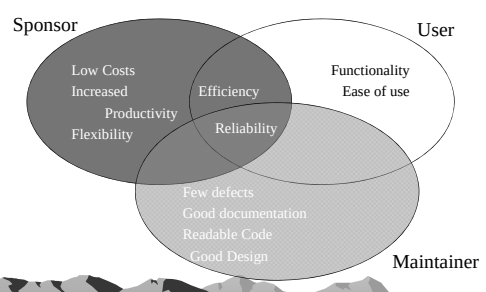
Computer Software

- Executable code
- Non-executable software:
  - A problem statement.
  - A requirements document.
  - A software design.
  - A software test plan & associated documents.
  - Source code.

Nature of Software

- Malleable: Software is easily modified, but "correct" modification is difficult.
- Software creation is human-intensive, using engineering and not manufacturing skills.
- Software does not wear out, but its requirements and the environment change.

Quality to Whom?



Software Qualities

- Correctness: a program is correct with respect to a formal specification.
- Reliability: probability that a program will not fail over a specified time period.
- Robustness: a program behaves reasonably under stress.
- Performance: efficient use of resources.
- Safety: do no harm.

Software Qualities (2)

- Portability: the ease of transferring software from one platform to another.
- Usability: ease of use.
- Maintainability: ease of maintaining.
- Reusability: SW unit's reuse potential.
- Usefulness: does it do something useful?

 1-7

Pragmatic Constraints

- Software must be completed within time and $ constraints.
- Software must work with existing software.

 1-8

Quality Requirements in Different Application Areas:

- Information systems: data integrity, security, data availability, transaction performance, usability.
- Distributed systems: system reliability, tolerance to network partitioning, fault tolerance.
- Embedded systems: response time, reliability, safety, usability.

 1-9

Long Ago: Expensive Hardware, Few Users.

- Focus on writing computer instructions.
- Formal and well understood problems.
- Programs written by user.
- Small gap between problem & solution.
- Small gap between user and computer.

IBM 7090, 1959
- $2.9 million.
- ~147 KB core storage.
- CPU Cycle time: 2.18 microSecs.
- Tape drives; no hard drive.
- Used by NASA to control space flights.
- Used in the movie Dr. Strangelove.

 1-10

Now: Cheap Hardware, Many Users.

- Focus on maintaining systems & defining requirements (informal problems).
- Programs written by programmers, not users.
- Large gap between problem & solution.
- Large gap between user and computer.

Apple iPhone 6
- $199 – 849 (unlocked).
- 1 GB RAM, main memory.
- CPU 64-bit A8 cycle time 2 GHz dual-core.
- 1334-by-750-pixel resolution.
- 16-128 GB Disk.
- 4.55 ounces.

 1-11

Conflict

The informal domain of humans versus the formal domain of computers.

Typical and "wicked" applications:
- Point-of-sale terminals and support systems.
- Income tax program for individuals or tax professionals.
- Digital dashboard for automobiles.
- Natural language interface for physical and cyber navigation.
- Driverless car.

 1-12

Common Requirement: Solve Problems in "Human Domain"

Key requirements can only be expressed informally.

- Point-of-sale: must reflect retail environment.
- Tax program: must reflect complex and changing tax laws, be correct (but specifications are not formal), and easy to use.
- Electronic dashboard: safety critical, and must be ergonomic.
- Natural languages are informally specified.
- Driverless car combines navigation with safety concerns.

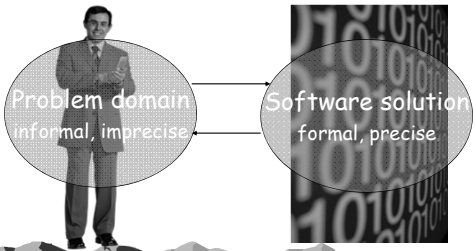
Computer Solution: A Formal System

- Computers are formal systems: machine language follows precise rules.
- Programs are formal: compile into machine instructions.

Running programs precisely execute discrete commands.

The Software Problem

We need formal solutions to informally described problems.



Software Development Myths [Pressman]

- Management myths:
  - Problems solved by standards & tools.
  - When schedules slip add more people.
  - All programmers are equal in ability.
- Software customer myths:
  - Change is easily accommodated.
  - A general statement of need is sufficient to start coding.

Software Development Myths (2)

- Developer myths:
  - The job is done when the code is delivered.
  - Project success depends solely on the quality of the delivered *program.*
  - You can't assess software quality until the program is running.

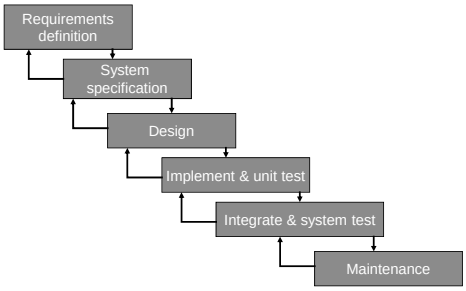
What Is a Software Process?

- What software developers do, their activities or tasks:
  - Requirements identification.
  - Specification.
  - Design.
  - Validation.
  - Evolution.

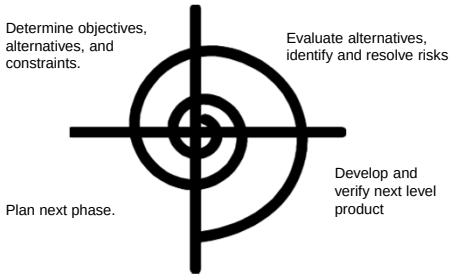
"Code & Fix" Model

- Write code, then test & debug.
- Problems:
  - Ignores requirements analysis & design.
    - Errors not corrected until after coding.
    - Software does not satisfy needs.
  - Code becomes unstructured after a number of fixes.
  - Debugging is difficult. Why?

Waterfall Process Model



Boehm's Spiral Model



Iterative Development Process

Many flavors of iterative development, for example the "Rational Unified Process (RUP)", which is a UML model-driven process.

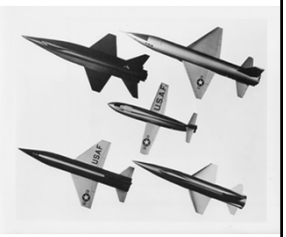
- Inception iterations: early interactions with stakeholders.
- Elaboration iterations: finalize requirements, define software architecture.
- Construction iterations: develop initial running system.
- Transition iterations: complete product release.

Each iteration type includes some portion of requirements, analysis, design, implementation, and test.

Iterative Development Models

- Use case model.
- Analysis model.
- Design model.
- Deployment model.
- Implementation model.
- Test model.

Each model depicts a different view of a system.



Models of competing X-15 designs surrounding the earlier Bell X-1A

Agile Processes

- Minimize risk by focusing on small increments of work.
- Typical cycle time: 1 week – 1 month.
- Priorities re-evaluated after each cycle.
- Aim is for flexibility – agility.
- Example agile processes:
  - Scrum
  - Extreme Programming

Scrum

- Roles
  - ScrumMaster
  - Product Owner
  - Team: group that does the work.
- Sprint
  - 1 week to 1 month cycle.
  - Constant length "timeboxed".
  - Some deliverable produced at the end of a sprint.

https://www.youtube.com/watch?v=XU0llRltyFM
https://www.youtube.com/watch?v=9TycLR0TqFA

Scrum Meetings

- Sprint planning: start of a sprint.
- Daily Scrum: very brief.
- Sprint review; retrospective.
- All meetings have strict time limits.
- The Scrum Master provides coaching and expertise at all points of the process and for all the artefacts of the process.

Sprint Planning Meeting

- Product Owner prioritizes user stories in the product backlog to be implemented during the sprint.
- Team commits to the stories they will implement during the sprint.
- Team Members decompose the committed stories into tasks with estimation times (or at "grooming" meeting).
- Team Members split user stories that are still large, and estimate their story points – a comparative effort estimate.
- Team Members and the Product Owner develop acceptance criteria for each user story so that everyone understands what is needed to declare the story "done": "Done" means acceptance criteria are met.
- Meeting input: **product backlog**, a prioritized set of user stories.
- Meeting O**utcome**: the **sprint backlog** – a set of user stories and all tasks needed to complete them that the team has committed to complete for the sprint.

Daily Scrum Meeting

Each member of the team states:

- Tasks they completed since the last meeting.
- Tasks they plan to complete before the next meeting.
- Obstacles encountered: team members volunteer to help if they are working on a lower priority user story.

Meeting i**nputs:** artefacts to the daily scrum from the task board.

Meeting **output:** the states of each task in the sprint: to do, doing, and done. Shown on a task board.

Scrum Review Meeting

Sprint **review meeting input:** a working system that with all the committed stories of the sprint implemented and tested.

- Team demos the user stories implemented during the sprint.
- Team conducts a private review of what did/didn't work, and decides how to improve the next sprint.

Review O**utput**: a set of improvements that will be made to the next sprint.

User Stories

User Story Template:
<title>
As a <type of user>,
I want to <do something>
so that I <get a benefit>

Sources:
http://agile2007.agilealliance.org/downloads/handouts/Smits_495.pdf
https://www.rallydev.com/sites/default/files/guide-userstories-v1.pdf
https://msdn.microsoft.com/en-us/library/hh273055

Priority:
<decided by the produc owner>

Story points:
<estimate of a story"s relative "size" – NOT lines of code (see next slide)>

Acceptance Criteria:
<whatever is needed to let the Product Owner know the user story was implemented as intended>

User Stories

- Acceptance criteria: a simple pass/fail test for each element of the criteria.
- Product owner assigns priorities to user stories.
- Team members assign (often tentative) story points to the story.
- As the user story moves to a higher priority, it is split into smaller, well-understood stories with less tentative story point estimates.
- User stories that reach the top of the product backlog are included in the next sprint
  - The story must be small enough that implementation tasks (and its acceptance criteria) can be identified, along with accurate time completion estimates.

When there is too much unknown about some part of a user story. Define a "Spike".

Spike

- A story or task aimed at answering a question or gathering information.
- Required to answer a technical question or solve a design problem in order to do a user story estimate.
- The spike is given an estimate and included in the sprint backlog.

Estimating User Story Points

- Story size estimates are ordinal.
  - Humans are good at comparing.
  - Estimation can be done quickly.
- Use Fibonacci scale: 1, 2, 3, 5, 8, 13, 21, …
  Separation between numbers makes it easer for team members to agree.
- Elements for point estimation:
  - Complexity
  - Effort
  - Doubt

Planning Poker

- Card Deck: Cards are inscribed with a number in the Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21, etc.
- Team members each choose a card that represents their best guess of the user story difficulty.
- Everyone turns over their cards at once.
- High and low card owners argue.
  Repeat process until estimates converge.

Project Manager or Scrum Master may serve as moderator.

Scrum Master (from Wikipedia definition)

- Accountable for removing impediments to the ability of the team to deliver the product goals and deliverables.
- Not a traditional team lead or project manager, but acts as a buffer between the team and distracting influences.
- The scrum master ensures that the scrum process is used as intended.
- The scrum master helps ensure the team follows the agreed scrum processes, often facilitates key sessions, and encourages the team to improve.
- referred to as a *team facilitator*

Ultimate Scrum Master:
https://www.youtube.com/watch?v=P6v-I9VvTq4&list=PLIXxHp9iBs-m6t1S6kmxeqigQBju86eFF

Decomposing a story into Tasks

- Tasks should be small enough that the estimated time needed for them should be 1-10 hours
- Break the user story into as many tasks as needed so that if all are completed the acceptance criteria will be met.
  - Defining test data, designing test approaches (e.g. unit test, GUI tests, integration tests, …) must be included as tasks.
- Sometimes "non-functional" requirements (e.g. performance or security-related) are included as split user stories and have their own set of related tasks.
- Regularly occurring tasks related to acceptance criteria for every user story can basis of the "definition of done" (e.g. all tests pass and there are no outstanding defects).

Sources:
https://www.rallydev.com/toolkits/iteration-planning-toolkit, Testing in the Iteration slides

Scrum Measures

Improving the team's ability to estimate:

- Velocity
  - Number of story points the team completes per iteration.
    - Initially, the team may underestimate story points.
    - Estimations will improve as a project progresses, and a more accurate team velocity will emerge.
- Capacity
  - The number of hours available to work on story tasks.
    - Since each user story has a set of tasks with hour estimations this number is used to decide which user stories to commit to for the sprint.
    - As the project progresses team task estimation times will become more realistic, and lead to a better match with committed user stories.

Extreme Programming

- Incremental iterative development aimed for small improvements in each cycle.
- Continuous unit and regression testing.
- Pair programming.
- On-site customer as part of the development team.
- Refactoring.
- Simplicity – don't do more than necessary.

Lots of Interest in Extreme Programming, but …

- Detailed specifications and designs are not written up:
  - UML diagrams are done on the "white board" and not saved as documentation.
  - Code is the only documentation.
- A customer representative is part of the project team.
- Programmers work in pairs.
- Design activity takes place "on the fly".
  Start with the simplest solution; add complexity only when required due to test failures.

May not be suitable for large projects.

Software Failures

- IRS Automated Income Tax Form Processing System (Sperry 1980's).
- SDI Star Wars software.
- Ariane-5 Rocket.
- Therac-25 Accidents.
- Year-2000 bug.
- London Ambulance Service Fiasco.
- Colorado Benefits Management System (2004).
- MS Zune failure on December 31, 2008.

…

IRS System

- Inadequate performance, cost overruns.
- 1985: $90 m. added to $103 m. spent.
- Late refunds.
  - IRS pays $40.2M in interest to taxpayers & $22.3M in overtime wages.
- 1996: Still no improvement! No master plan; only a 6,000 page technical document.

SDI "Star Wars" Software (1983 -)

Goal: use ground and space-based systems to protect the US from attack by nuclear ballistic missiles.

- 10M+ lines of code.
- Huge testing job. But.
  - Cannot tested under operational conditions.
  - Tests must be done via simulation.
- Required reliability: fewer than 1 failure in 10^9 hrs. of operation.
  - To demonstrate this reliability, the system must run for more than 10^9 hrs. without failure.

Ariane-5 Rocket launched on June 4 1996.

- Veered off course after ~40 sec.
- Destroyed by remote control.
- Reason: incorrect requirement spec.
- $500 million worth of equipment lost.
- Future economic loss: Ariane held more than half of the world's launch contracts.

The Therac-25 Accidents

- Therac-25: a computerized radiation therapy machine.
- Accelerates electrons to create high-energy beams to destroy tumors with minimal impact on healthy tissue.
- June 1985 - Jan. 1987: 6 known accidents involving massive overdoses; some resulting in deaths.
- Accidents caused by faulty software.

Year 2000 "Bug"

- Only 2 digits to store year data.
  "We didn't think that anyone would be using this in 2000."
- Result: *potential* for failures in all SW computing dates after 01-01-00.
  - Financial software failures.
  - Embedded system failures.
  - The entire economic system!!
  - Panic!

Y2K Repair

- Must scan all code looking for dates.
- Determine all dependencies.
- Make changes.
- Test, test, test.

Observations

- Y2K was a risk but was over-hyped.
  Why? It's a bug that is easy for reporters to understand.
- Although there were few real failures, social effects were significant:
  Many people planned for a shutdown of the economic & industrial infrastructure.
  Y2K bug supported existing millenium fears.

London Ambulance Service (LAS) Fiasco

- Largest ambulance service in the world.
- The LAS computer aided dispatch system replaced a manual system.
- The system failed when it went on line in 1992:
  - Overloaded by normal use.
  - Multi-hour delays in responses to emergency calls.
  - Ambulance communications failed and ambulances "disappeared".

LAS Fiasco Causes

- Winning contractor's experience was only for administrative systems.
- No independent quality assessment.
- Concerns not followed up.
- System did not match prior manual process.
- Lack of voice control & flexibility.
- Need for perfect information.
- Poor interface:
  - Failure to identify duplicated calls.
  - Exception messages scrolled off of screen.
  - …
- Memory leaks.
- …

 1-49

Colorado Benefits Management System

- Replaced prior system for reviewing and approving applications for benefits.
- Failed immediately after put in operation in 2004.
  - Required 17 screens to process 1 case.
  - Each screen took up to 24 minutes to load.
  - System would time out.

  Poor clients!, poor caseworkers!
- No provision for a rollback to prior working system!!

 1-50

CBMS Fiasco Causes

- No load testing.
- No usability testing.
- Extreme work hours by contractors.
- Contractor's engineers had little experience.

The mistakes were avoidable with good software engineering practices.

 1-51

Zune Failure 12/31/2008

- From Wikipedia:

  "At approximately midnight Pacific Standard Time, on the morning of December 31, 2008, many first generation Zune 30 models froze. Microsoft has stated that the problem is with the internal clock driver written by Freescale and the way the device handles a leap year

  …

  a third party analysis of the clock driver's source code revealed an infinite loop in the way the clock driver calculates years based on a given number of elapsed days."

 1-52

Affordable Care website rollout

- Inexperienced project management.
- Waterfall process: couldn't test until it was all implemented.
- Multiple contractors.

 1-53

Toyota acceleration accidents

- Recall of 9 million vehicles (model years 2009-2011).
- Blamed on floormat problem.
- Suspicious software:.

  Possible race conditions due to global variable references by device drivers.

 1-54

James M. Bieman and Geri Georg
2016

Software Successes

- Phone systems: wired & wireless.
- Banking & securities systems.
- Embedded systems.
- Medical systems.
- Entertainment.

. . .