CS189A - Capstone

Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/

Project Assignments

- All the teams must be formed today **Monday, January 10th**
  - If you have not done so yet, you have to form your teams and pick a project

- Each team must prepare a two page vision statement about the project describing, due **Friday, January 14th** (23:59 PST)
- A team member from each team is going to present (5 minutes) this vision statement to the class on **Friday, January 14th**
- Each team must form a Google group and must invite the instructor, the TA, and the industry mentors for their project to the group by **Friday, January 14th**

Software Engineering

- In 1968 a seminal NATO Conference was held in Germany.
  - Purpose: to look for a solution to ***software crisis***
  - 50 top computer scientists, programmers and industry leaders got together to look for a solution to the difficulties in building large software systems (i.e., software crisis)
  - The term "***software engineering***" was first used in that conference to indicate ***a systematic, disciplined, quantifiable approach to the production and maintenance of software***
- Three-decades later (1994) an article in Scientific American (Sept. 94, pp. 85-96) by W. Wayt Gibbs was titled:
  - "Software's Chronic Crisis''

Software's Chronic Crisis

Large software systems often

- do not provide the desired functionality
- take too long to build
- cost too much to build
- require too much resources (time, space) to run
- cannot evolve to meet changing needs
  - for every six large software projects that become operational, two are canceled
  - on the average software development projects overshoot their schedule by half
  - 75% of the large systems do not provide required functionality

Dependability and Failures

- It is extremely difficult to build dependable software systems
  - it is almost expected that any software system will have bugs
- There is a long list of failed software projects and software failures
- You can find a list of famous software bugs at:
  http://www5.in.tum.de/~huckle/bugse.html
- I will talk about two famous and interesting software bugs

Ariane 5 Failure

- A software bug caused European Space Agency's Ariane 5 rocket to crash 40 seconds into its first flight in 1996 (***cost: half billion dollars***)

Ariane 5 Failure

- The bug was caused because of a software component that was being reused from Ariane 4

- A software exception occurred during execution of a data conversion from 64-bit floating point to 16-bit signed integer value
  - the value was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus, the conversion failed and an exception was raised by the program

- When the primary computer system failed due to this problem, the secondary system started running.
  - secondary system was running the same software, so it failed too!

Ariane 5 Failure

- The programmers for Ariane 4 decided that this particular velocity figure would never be large enough to raise this exception.
  - Ariane 5 was a faster rocket than Ariane 4!

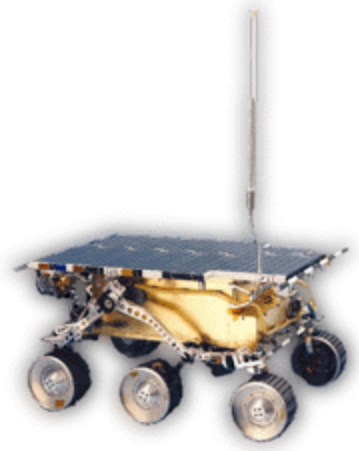
- The calculation containing the bug actually served no purpose once the rocket was in the air.
  - Engineers chose long ago, in an earlier version of the Ariane rocket, to leave this function running for the first 40 seconds of flight to make it easy to restart the system in the event of a brief hold in the countdown.

- You can read the report of Ariane 5 failure at:

  http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

Mars Pathfinder

- A few days into its mission, NASA's Mars Pathfinder computer system started rebooting itself
  - cause: priority inversion during preemptive priority scheduling of threads



Mars Pathfinder

- Priority inversion occurs when
  - a thread that has higher priority is waiting for a resource held by thread with a lower priority
- Pathfinder contained a data bus shared among multiple threads and protected by a mutex lock
- Two threads that accessed the data bus were: a high-priority bus management thread and a low-priority meteorological data gathering thread
- Yet another thread with medium-priority was a long running communications thread (which did not access the data bus)

Mars Pathfinder

The meteorological data gathering thread accesses the bus and obtains the mutex lock

While the meteorological data gathering thread is accessing the bus, an interrupt causes the high-priority bus management thread to be scheduled

Bus management thread tries to access the bus and blocks on the mutex lock

Scheduler starts running the meteorological thread again

Before the meteorological thread finishes its task yet another interrupt occurs and the medium-priority (and long running) communications thread gets scheduled

At this point high-priority bus management thread is waiting for the low-priority meteorological data gathering thread, and the low-priority meteorological data gathering thread is waiting for the medium-priority communications thread

Since communications thread had long-running tasks, after a while a watchdog timer would go off and notice that the high-priority bus management thread has not been executed for some time and conclude that something was wrong and reboot the system

The problem scenario

Meteorological (low)	**Bus Manager (high)**	**Communication (medium)**
Running, gets the lock		
Waiting, has the lock	Running	
Waiting, has the lock	Running, wants the lock	
Running, has the lock	Blocked due to the lock	
Waiting, has the lock	Blocked due to the lock	Running

Since the execution of the Communication thread takes too long, the Bus Manager thread is blocked for a long time. So the system reboots.

Software's Chronic Crisis

- These are not isolated incidents:
  - IBM survey of 24 companies developing distributed systems:
    - 55% of the projects cost more than expected
    - 68% overran their schedules
    - 88% had to be substantially redesigned

Software's Chronic Crisis

30 December 2010 Last updated at 05:02 ET

Skype c
overloa

Server overloads Windows caused net phone firm.

Details of what cau
unusable for millio
have been posted

The two events co
of problems that m
the network under

Skype is assessing
stop the problem r

Apple iOS bug causes iPhone, iPod to miss alarms in 2011

News by *Michael Oryl* on Saturday January 01, 2011.

Sponsored links, if any, appear in green.

MobileBurn.com would like to wish a "Happy New Year!" to those of you with working alarm clocks.

It appears that Apple has another alarm clock bug on its hands. Unlike the Apple iOS Daylight Saving Time bug that impacted users with recurring alarms set, this bug causes any non-recurring alarms set to fire on January 1 or January 2, 2011 to be ignored. We've confirmed this ourselves, and can tell you that it occurs not only on Apple's iPhone smartphones, but also on the iPod touch. It is unclear which versions of iOS contain the bug, but it is being widely reported on Twitter that at least iOS 4.1 and 4.2.1 are affected.

Capability Maturity Model

- Maturity levels used to evaluate software development processes
  - **1) Initial.** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
  - **2) Repeatable.** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
  - **3) Defined.** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

Capability Maturity Model

- Maturity levels used to evaluate software development processes
  - **4) Managed.** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
  - **5) Optimizing.** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Software's Chronic Crisis

- Software product size is increasing exponentially
  - faster, smaller, cheaper hardware
- Software is everywhere: from TV sets to cell-phones
- Software is in safety-critical systems
  - cars, airplanes, nuclear-power plants
- We are seeing more of
  - distributed systems
  - embedded systems
  - real-time systems
- Software requirements change
  - software evolves rather than being built

Summary

- Software's chronic crisis: Development of large software systems is a challenging task
  - Large software systems often: Do not provide the desired functionality; Take too long to build; Cost too much to build Require too much resources (time, space) to run; Cannot evolve to meet changing needs

- Software engineering focuses on addressing challenges that arise in development of large software systems using a systematic, disciplined, quantifiable approach

No Silver Bullet

- In 1987, in an article titled

  "*No Silver Bullet: Essence and Accidents of Software Engineering*"

  Frederick P. Brooks made the argument that there is no silver bullet that can kill the werewolf software projects

- Following Brooks, let's philosophize about software a little bit

Essence vs. Accident

- Essence vs. accident in software development
  - we can get rid of accidental difficulties in developing software
  - getting rid of these accidental difficulties will increase productivity

- For example, using a high level programming language instead of assembly language programming
  - the difficulty we remove by replacing assembly language with a high-level programming language is not an essential difficulty of software development
    - it is an accidental difficulty brought by inadequacy of assembly language for programming

Essence vs. Accident

Essence vs. accident in software development

"*The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms and invocations of functions.* ***This essence is abstract in that such a conceptual construct is the same under many different representations****. ... The hard part of building software is the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation*."

Even if we remove all accidental difficulties which arise during the translation of this conceptual construct (design) to a representation (implementation), still at its essence software development is difficult

Inherent Difficulties in Software

- Software has the following properties in its *essence*:
  - complexity
  - conformity
  - changeability
  - invisibility
- Since these properties are not *accidental* representing software in different forms do not effect them
- The moral of the story:
  - Do not raise your hopes up for a silver bullet, there may never be a single innovation that can transform software development as electronics, transistors, integrated-circuits and VLSI transformed computer hardware

Complexity

- Software systems do not have regular structures, there are no identical parts
- Identical computations or data structures are not repeated in software
- In contrast, there is a lot of regularity in hardware
  - for example, a memory chip repeats the same basic structure millions of times

Complexity

- Software systems have a very high number of discrete states
  - Infinite if the memory is not bounded
- Elements of software interact in a non-linear fashion
- Complexity of the software increases much worse than linearly with its size

Complexity

- Consider a plane that is going into a wind-tunnel for aerodynamics tests
  - during that test it does not matter what is the fabric used for the seats of the plane, it does not even matter if the plane has seats at all!
  - only thing that matters is the outside shape of the plane
  - this is a great abstraction provided by the physical laws and it helps mechanical engineers a great deal when they are designing planes
- Such abstractions are available in any engineering discipline that deals with real world entities
- Unfortunately, software engineers often do not have the luxury of using such abstractions which follow from physical laws
  - software engineers have to develop the abstractions themselves (without any help from the physical laws)

Conformity

- Software has to conform to its environment
  - Software conforms to hardware interfaces not the other way around
- Most of the time software systems have to interface with an existing system
- Even for a new system, the perception is that, it is easier to make software interfaces conform to other parts of the system

Changeability

- Software is easy to change, unlike hardware
- Once an Intel processor goes to the production line, the cost of replacing it is enormous (Pentium bug cost half billion dollars)
- If a Microsoft product has a bug, the cost of replacing it is negligible
  - just put the new download on a webpage and ask users to update their software

Changeability is not an Advantage

- Although it sounds like, finally, software has an advantage over hardware, the effect of changeability is that there is more pressure on changing the software

- Since software is easy to change software gets changed frequently and deviates from the initial design
  - adding new features
  - supporting new hardware

Changeability

- Conformity and Changeability are two of the reasons why reusability is not very successful in software systems
- Conformity and Changeability make it difficult to develop component based software, components keep changing

Invisibility

- Software is invisible and un-visualizable
  - complete views can be incomprehensible
  - partial views can be misleading
  - all views can be helpful
- Geometric abstractions are very useful in other engineering disciplines
  - Floor plan of a building helps both the architect and the client to understand and evaluate a building
- Software does not exist in physical space and, hence, does not have an inherent geometric representation

Invisibility

- Visualization tools for computer aided design are very helpful to computer engineers
  - software tools that show the layout of the circuit (which has a two-dimensional geometric shape) makes it much easier to design a chip

- Visualization tools for software are not as successful
  - there is nothing physical to visualize, it is hard to *see* an abstract concept
  - there is no physical distance among software components that can be used in mapping software to a visual representation
  - UML and similar languages are making progress in this area

Summary

- According to Brooks, there are essential difficulties in software development which prevents significant improvements in software engineering:
  - Complexity; Conformity; Changeability; Invisibility

- He argues that an order of magnitude improvement in software productivity cannot be achieved using a single technology due to these essential difficulties

How Should We Build Software?

Let's look at an example

- Assume we asked our IT folks if they can do the following:
  - Every year all the PhD students in our department fill out a progress report that is evaluated by the graduate advisors. We want to make this online.

- After we told this to our IT manager, he said "OK, let's have a meeting so that you can explain us the functionality you want."

- We scheduled a meeting and at the meeting we went over
  - The questions that should be in the progress report
  - Type of answers for each question (is it a text field, a date, a number, etc?)
  - What type of users will access this system (students, faculty, staff)?
  - What will be the functionality available to each user?

Requirements Analysis and Specification

- Meeting where we discussed the functionality, input and output formats, types of users, etc. is called requirements analysis
  - during requirements analysis software developers try to figure out the functionality required by the client

- After the requirements analysis all these issues can be clarified in a **Software Requirements Specification (SRS)** document
  - maybe the IT folks who attended the requirements analysis meeting are not the ones who will develop the software, so the software developers will need a specification of what they are supposed to build.

- Writing precise requirements specifications can be challenging:
  - formal (mathematical) specifications are precise,
    but hard to read and write
  - English is easy to read and write, but ambiguous

Design

- After figuring out the requirements specifications, we have to build the software

- In our example, we assume that the IT folks are going to talk about the structure of this application first
  - there will be a backend database, the users will first login using an authorization module, etc.

Design

- Deciding on how to modularize the software is part of the **architectural design**
  - it is helpful (most of the time necessary, since one may be working in a team) to document the design architecture (i.e., modules and their interfaces) before starting the implementation

- After figuring out the modules, the next step is to figure out how to build those modules

- **Detailed design** involves writing a detailed description of the processing that will be done in each module before implementing it
  - generally written in some structured pseudo-code

Implementation and Testing

- Finally, the IT folks are going to pick an implementation language (PHP, Java Servlets, etc) and start writing code

- This is the **implementation** phase
  - implement the modules defined by the architectural design and the detailed design.

- After the implementation is finished the IT folks will need to check if the software does what it is supposed to do

- Use a set of inputs to **test** the program
  - when are they done with testing?
  - can they test parts of the program in isolation?

Maintenance

- After they finished the implementation, tested it, fixed all the bugs, are they done?

- No, we (client) may say, “I would like to add a new question to the PhD progress report” or “I found a bug when I was using it” or “You know, it would be nice if we can also do the MS progress reports online” etc.
  - The difficulty of changing the program may depend on how we designed and implemented it

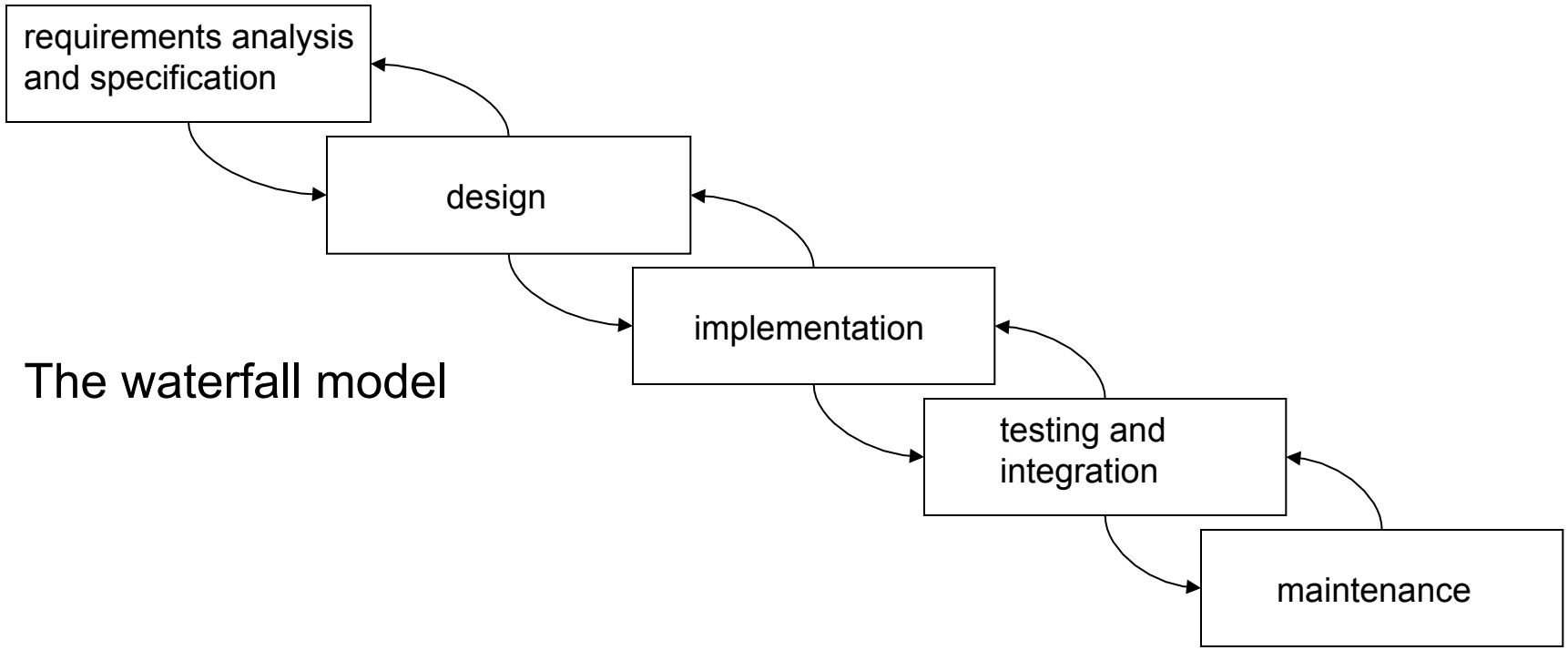
- This is called the **maintenance** phase where the software is continually modified to adopt to the changing needs of the customer and the environment.

Software Process Models

- Software process (software life-cycle) models
  - Determine the stages (and their order) involved in software development and evolution
  - Establish the transition criteria for progressing from one stage to the next

- Software process models answer the questions:
  - What shall we do next?
  - How long shall we continue to do it?

Waterfall Model



The waterfall model

Software product is not only the executable file:
source code, test data, user manual, requirements specification, design specification

Waterfall Model

- Waterfall model is *document-driven*
- Documents
  - requirements specification, design specification, test-plan
  - these documents are crucial in achieving maintainability, traceability and visibility
- Feedback loops between different stages are confined to successive stages to minimize the expensive rework involved in feedback across many stages

Waterfall Model

Problems with waterfall model

- Because of the restricted feedback loops, waterfall model is essentially sequential
  - for example, the requirements have to be stated completely before the implementation starts.
  - it is often difficult for the customer to state all requirements explicitly
  - it is hard to handle changes in the requirements
- A working model of the software is not available until late in the project life-span
  - an undetected mistake can be very costly to fix
  - the delivered program may not meet the customer's needs
- For interactive, end-user applications document-driven approach may not be suitable.
  - for example, it is hard to document a GUI