CS189A - Capstone

Chandra Krintz
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~ckrintz/

https://capstone.cs.ucsb.edu/cs189a/cs189a_sched.html
Software process activities

1. Software specification
   – Customers and engineers define the software that is to be produced and any constraints on its operation

2. Software design
   – Software spec is designed and prototyped

3. Software implementation, validation, and testing
   – Software is programmed and checked to ensure that it is what the customer requires

4. Software maintenance and evolution
   – Software is maintained (bug fixes, upgrades) and modified to reflect changing customer and market requirements
• Discussion/debate on the functionality, input and output formats, types of users, etc. is called requirements analysis
  – Product managers and/or software developers try to figure out the functionality required by the client
  – Functional and non-functional requirements
• Discussion/debate on the functionality, input and output formats, types of users, etc. is called requirements analysis
  – Product managers and/or software developers try to figure out the functionality required by the client
  – Functional and non-functional requirements

• Writing precise requirements specifications can be challenging:
  – Formal (mathematical) specifications are precise, but hard to read/write
  – English is easy to read and write, but ambiguous
  – Today’s solutions employ a combination of
    • IEEE Software Requirements Specification (SRS), Product Requirements & Design (PRD) – combined with system modeling, user stories, case studies
    • Should be a “living document” that evolves over time
      – Starts with a vision statement
(2) Software Design

- Product managers/owners do not develop the software
  - Software developers use requirements doc to understand what to build

- Sketch out the functionality in the requirements specification

- Model the system and its components
  - Context, interactions, structural, behavioral
  - User interfaces, user experience
  - Use cases, sequencing, event response, system state, classes/objects

- Define software architecture: **drawings, evolving docs, coding**
  - Components with interfaces (application programming interfaces: APIs)
  - High level and low level
    - Dependencies, modules, alternatives
    - Patterns
  - Prototype components -- **mock out** / **simulate** missing pieces
(3) Implementation and Testing

- Decide on technologies to incorporate/integrate/reuse

- Implement modules defined by architectural design & detailed design
  - Typically as prototypes that evolve over time into production-quality SW

- As part of prototyping and evolving testing happens *concurrently*
  - That requirements are met, assumptions are held, bugs are minimized
  - Be **defensive!** Prevent cases that you haven't considered from ever executing (assert! exit! return error!)

- Use a set of inputs/actions to **test** the program
  - When are you done with testing?
  - Test parts of the program in isolation
  - Unit tests, functional tests, integration tests
Validation, Verification and Testing

- Reviews, walkthroughs, inspections
- Software testing:
  - black-box vs. white-box; functional vs. structural
  - random testing, exhaustive testing
  - domain testing, boundary conditions
  - coverage criteria: statement, branch & path coverage, condition coverage, multiple condition coverage
  - unit testing, stubs, drivers
  - integration& testing: top-down vs. bottom-up integration and testing
  - regression testing
(4) Maintenance & Evolution

• We finished implementation, tested it, fixed all the bugs, are we done?

• No, we (client) may say, “I would like to add …” or “I found a bug when I was using it” or “You know, it would be nice if we could also …” etc.
  – **Ease of changing depends on how SW is designed and implemented**

• Phase in which the software is continually modified to adapt to the changing needs of the customer and the environment

• **At some point, the software’s lifetime ends**
  – It is decommissioned, deprecated (APIs) and/or no longer supported
  – Typically this is a business decision
SW Specification (1): Requirements Analysis and Documentation

• Discussion/debate on the functionality, input and output formats, types of users, etc. is called requirements analysis
  – Product managers and/or software developers try to figure out the functionality required by the client
  – Functional and non-functional requirements

• Writing precise requirements specifications can be challenging:
  – Formal (mathematical) specifications are precise, but hard to read/write
  – English is easy to read and write, but ambiguous

  Today’s solutions employ a combination of
  • IEEE Software Requirements Specification (SRS), Product Requirements & Design (PRD) – combined with system modeling, user stories, case studies
  • Should be a “living document” that evolves over time
    – Starts with a vision statement
2-Page Vision Statement

- PDF via email to TA
  - Project Title / Name (can change)
  - Team name, members names/emails
  - Team lead
  - What the project is about?
    - What problem the project is solving *(what is innovation, the science, and new core technical advance)*?
    - Why the problem is important
    - How the problem is solved today (if it is)
  - Identify the outcome of the project
  - Define initial project milestones: specification, design, prototyping
  - How do you plan to articulate and design a solution
    - List the implementation platform and technologies will plan to use to develop the solution
    - List initial milestones and **how you plan to achieve them**
Capstone Award Judging Criteria

5pt **Science**: Has the project the demonstrated application of important, interesting, or new aspects of Computer Science? (e.g. Use of machine learning, non-trivial algorithms, solid distributed system design techniques)

5pt **Practice**: Did the project adhere to techniques that represent the state of best practice in industry throughout the development of the system (e.g. repo workflows, test-driven development, issue tracking, or use of static or dynamic analysis tools)

5pt **Scope**: Has the team attacked a problem of significant (but appropriate) scale and complexity. Does the problem require the development of significant new code and/or the integration of complex exciting parts that are not normally made to interface to on another? Was the project able to complete the goals that it set for itself?

5pt **Teamwork and Presentation**: Do all the members of the team contribute significantly (in their own ways)? Does the team take the process seriously and communicate effectively with one another and the mentors? Is the project presented both in written and spoken form in a way that is compelling and impressive? Has the team developed an impressive demo?
Team Building

• Requires people skills. Unless there is some understanding of people, team will be unsuccessful and can fail. **Keys:**
  
  • Consistency
    – Team members should all be treated in a comparable way without favourites or discrimination.
  
  • Respect
    – Different team members have different skills and these differences should be respected.
  
  • Inclusion
    – Involve all members and ensure that everyone’s views are considered
  
  • Honesty
    – Be honest about what is going well and what is going badly in a project
Teamwork

- Most software engineering is a group activity
  - The development schedule for most non-trivial software projects is such that they **cannot be completed by one person working alone**.

- A good group is **cohesive** and has a **team spirit**. The people involved are motivated by the success of the group as well as by their own personal goals.

- **Group interaction** is a key determinant of group performance.

- **Flexibility** in group composition is limited
  - Lead must do the best they can with available people.

- **Good communications** across team is essential for success
  - Promotes trust & understanding
CS189A F18 Next Steps

• Today (Monday):
  – Intro to SWE and vision statement
  – Discuss and choose teams (5 members)
  – Identify **group leader** and **scribe**
    • **Lead**: motivator, picks up all loose ends, settles debates/makes decisions
    • **Scribe**: records scrums, retrospectives, sprint planning, mentor/TA meetings
  – 1:45pm send selection email to instructor

• Tuesday: Contact mentors to introduce yourselves
  – Setup weekly meeting times with team and company mentors

• Thursday: draft/send vision statement to mentors for feedback
  – Send to TAs (by end of discussion)
    • Github repo, project description sentence, google doc & group setup and
    • Send invites (github,shared docs) to TAs, instructor, team, mentors

• **Vision statement due next Monday by end of class**
CS189A: Claiming a Project

- Only **complete** emails dated after **1:45pm** on Monday Oct 8 considered
  - Email to Chandra (ckrintz@gmail.com) with
    - Subject: 189a project selection
    - Sent by group leader (or his/her representative)
    - List of group member names, **identify lead and scribe**
    - A picture of each group member for public posting
      - The file name must be: LASTNAME.png
      - Please use 512x512 resolution and png file format
        - Contact the TA if you need help with any of this
    - List all company participants in order of preference

- **FCFS assignments + algorithm**
  - You are **not** likely to get your top preferences
  - One team per project/company!

- Team-mentor pairings announced Tuesday morning