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
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
Software Process Models

• Stages of software engineering: requirements specification, design, implementation, testing, maintenance

• Software process (software life-cycle) models
  – Determine the stages (and their order)
  – Establish the transition criteria for progressing from one stage to the next
Software Process Models

- Stages of software engineering: requirements specification, design, implementation, testing, maintenance

- Software process (software life-cycle) models
  - Determine the stages (and their order)
  - 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?

- Models we’ll discuss: waterfall, spiral, evolutionary: agile/extreme
  - Waterfall (70s, 80s) when all software was “shrink wrapped and shipped”
  - Spiral (late 80s) risk-driven and iterative; Rational Unified Process (UP or RUP)
  - Evolutionary (late 90s, early 00s) as SW becomes increasingly online
The waterfall model

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

Software product is not only the executable file: source code, test data, user manual, requirements specification, design specification, test-plan. These documents are crucial in achieving maintainability, traceability and visibility.
Waterfall Model

Problems with waterfall model

- Because of the restricted feedback loops, waterfall model is essentially sequential
  - for example, requirements must be stated completely before implementation starts
  - it is often difficult for the customer to state all requirements explicitly
  - 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 work
  - for example, it is hard to document a GUI
Risk driven, iterative
BUT: software delivered only after many iterations

Spiral Model (late 80s origin)

Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Develop, verify next-level product
Plan next phase

cumulative cost
progress in each cycle

radial dimension shows the cumulative cost
angular dimension shows the progress in each cycle

Attack the highest risk part (usually obtaining proper user requirements) of the project first, iterate over next highest risk sub-problem
Evolutionary Software Development

- Software is built iteratively and incrementally by first providing an initial version and then improving/extending it based on the user feedback until an adequate system has been developed (late 90s, early 00s origin)
  - Agile software development, extreme programming
  - Triggered by change in application type (consumer, phones, web)
- All activities are executed concurrently with fast feedback among them
- Specifics impacted by application domain and deployment strategy (e.g. cloud/SaaS, web app)
Agile Software Development

Manifesto for Agile Software Development (2001)
available at: http://agilemanifesto.org/

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

That is, while there is value in the items on the right, we value the items on the left more”
Principles of Agile Software Development

• 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**.
Principles of Agile Software Development

- 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.
Extreme Programming

• Extreme programming (XP) is a type of agile software development process proposed by Kent Beck (~late 90’s)

• XP follows the agile software development principles as follows
  – Software is built *iteratively*, with *frequent releases*
  – Each release implements the set of *most valuable features/use-cases/stories* that are chosen by the customer
  – Each release is implemented in a *series of iterations*, each iteration adds more features/use-cases/stories
  – Programmers turn the stories into *smaller-grained tasks*, which they individually accept responsibility for
  – The programmer turns a task into a set of *test cases* that will demonstrate that the task is finished
  – Working as *pairs*, the programmers make the test cases run, evolving the design in the meantime to maintain the simplest possible design for the system as a whole
Scrum

- An evolutionary/iterative/incremental/agile software process implementation
  - See: Scrum and XP from the Trenches -- free online book by Kniberg
Scrum

- An evolutionary/iterative/incremental/agile software process impl.
  - See: *Scrum and XP from the Trenches* -- free online book by Kniberg

- The main roles in Scrum are:
  - Scrum team: Team of software developers
  - Scrum master: Project manager
  - Product owner: Client

- Characteristics of Scrum:
  - Self-organizing teams
  - Product development in two to four week *sprints*
  - Requirements are captured as items in a list of *product backlog*
    - Yours will come from your requirements document (PRD)

- Homework: read the links on webpage under today’s date
Sprint and Scrum: Implementation

• **Sprint planning**
  – Use cases or user stories broken down into tasks
    • Team members assign timings (how long each will take)
    • And pick tasks
  – Tasks = designing, implementing, testing, and demo'ing
    • Includes code review
  – Any new tasks identified enroute go onto backlog for inclusion next time

• **Daily standup**
  • What I did yesterday, what I'm doing today, + any blockers
  • Short/quick so done standing up!

• **Retrospective and end of each iteration (identify ways to improve)** – what worked and what didn't
  • Vote on what to fix in the next sprint
More on Scrum

• More information about Scrum process is available at:
  – www.mountaingoatsoftware.com/scrum
  – www.scrumalliance.org
  – www.controlchaos.com

• Required reading
  – "Scrum/XP From the Trenches" by H. Kniberg. (Free with registration).
CS189A Goals & Requirements

• Four 2-week sprints:
  – Oct 15-29 (PRD v1 – tools, technologies, design, terminology);
  – Oct 29-Nov 8 (use cases/user studies, prototyping, PRD v1);
  – Nov 8-26 (design, prototyping, testing, PRD v2);
  – Nov 26-Dec 6 (prototype demo/pres prep, prototyping and testing)

• Specify what the product will do
  – Vision statement Due Oct 15
  – Product Requirements Document (PRD) (due Nov 1 and Nov 29)
  – Design tools, brainstorming, coding (tests and implementation)

• Build and test an initial prototype
  – Typically teams iterate on these activities until they converge to a working prototype!
Managing people

• Leading a team requires people skills. Unless there is some understanding of people, management will be unsuccessful.
• Poor people management can lead to project failure. 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
Motivating people

• The group leader serves as the external interface of the group, to motivate and guide, but not necessarily allocate work items
  – Motivation means organizing the work and the working environment to encourage people to work effectively.
  – Members choose what they work on, but teams decide priorities together
  – If people are not motivated, they will not be interested in the work they are doing. They will work slowly, be more likely to make mistakes and will not contribute to the broader goals of the team or the organization.

• Motivation is a complex issue but it appears that there are different types of motivation based on:
  – Basic needs (e.g. food, sleep, etc.);
  – Personal needs (e.g. respect, self-esteem);
  – Social needs (e.g. to be accepted as part of a group).
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
What’s Next

• This Friday
  – Pitch night – 12-13 companies come to try to sell you on their project
    • They will be your mentors for the quarter
  – Consider the metrics on which the Capstone winners will be based
  – Have fun!
  – 3:30-7pm in HFH 1104

• Monday
  – Form teams, submit your requests, decisions by Tuesday

• Thursday (discussion section) - mandatory attendance starts
  – Setup team, project, meeting schedule
  – Draft vision statements and send to mentors (due by end of class of next class period)