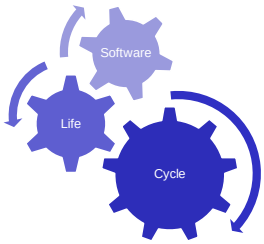


There is no 'I' in team!

CSE403: Software Engineering

David Notkin
Winter 2009

Dealing with complexity

- Breaking large, complex things down to manageable pieces is essential
- In computer science, this is called divide and conquer
  - Based on Latin's divide *et impera*, divide and rule
  - "[I]t often refers to a strategy where small power groups are prevented from linking up and becoming more powerful, since it is difficult to break up existing power structures" [Wikipedia].
- No human can fully conceive of or understand 50MLOC in any real sense
- Step-wise refinement in programming: input, process, output
- CSE degrees as general education, math & science, and computer science components
  - General education as language skills, areas of knowledge, etc.
  - Computer science as required, senior electives, and free electives
- US government as executive, judiciary and legislative branches
- Music as melody, pitch, rhythm, harmony, dynamics, etc.
- …

50MLOC = 50 million lines of code

- 50 lines/page-side ⇒ 1M page-sides
- 1K page-sides/ream ⇒ 1K reams
- 2 inches/ream ⇒ 2K inches
- 2K inches = 167 feet ≈ twice the height of the Allen Center
- 5 words/LOC @ 50 wpm ⇒ 50MLOC/5M min
- 5M min = 83,333 hr = 3,472 days ≈ 10 years

Just to type!
No breaks and
no thinking allowed!

Addressing software complexity

What are/is the …?

- Requirements
- Design
- Implementation
- Testing plan
- …

Who does the …?

- Requirements
- Design
- Implementation
- Testing
- …

- In some sense, two sides of the same coin
- Different approaches, representations, etc. are needed for the artifact-oriented components
- Different skill-sets, knowledge, etc. are needed for the human-oriented components

Software lifecycle and team structure

- These are essentially ways to decompose, respectively, the complex artifact-oriented and human-oriented aspects of the development of large software systems
- There are a multitude of approaches to each: as usual, no single approach to either is best in all circumstances – but that doesn't mean that any approach useful in any situation
- There are weak analogies to management structures: consider matrix structures that try to balance people responsible for particular functions (such as engineering or sales or advertising) with people responsible for particular products

Decomposition is not enough

- "Divide and conquer. Separate your concerns. Yes. But sometimes the conquered tribes must be reunited under the conquering ruler, and the separated concerns must be combined to serve a single purpose." —M. Jackson, 1995
- Put another way, hierarchical (or other) decomposition isn't the whole solution to complexity – the composition of those sub-results into an overall solution is crucial
- Put yet another way, every part may work properly, but the overall system may not – this is *not* a successful outcome

A concrete example

- Logical operations usually work easily in the face of decomposition: for example, we can mechanically build truth tables in propositional logic for non-atomic formulae such as
  ((¬a ∧ b ∧ c) ∨ (a ∧ b ∧ c) ∨ (a ∧ b ∧ ¬c))
- But they don't work so easily for software in general
  - **(scanner ∧ parser ∧ type-checker symbol-table ∧ code-generator ∧ optimizer)** does not a compiler make
  - **¬(P that crashes the Mars Polar Lander)** won't give us a program that does land it safely

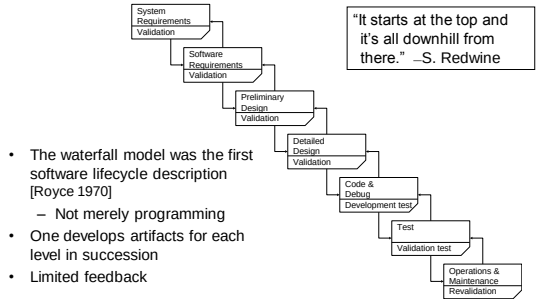
Another concrete example

- Meet with your team on Friday but don't meet again for eight weeks – then see how your project does
- That is, the human tasks must be composed regularly or else they will surely diverge from the overall goals

Reprise

- For activities
  - *What* should we do next?
  - How long should we continue to do *it*?
- For people
  - *Who* should do it?
  - How can *we* communicate with *others* about it?
  - When are *we* done with it?
- These cannot be fully separated, of course

Software lifecycle: classic waterfall



"It starts at the top and it's all downhill from there." –S. Redwine

- The waterfall model was the first software lifecycle description [Royce 1970]
  - Not merely programming
- One develops artifacts for each level in succession
- Limited feedback

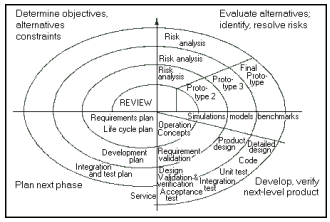
Comments?

Lifecycle stages

- Virtually all lifecycles share
  - Requirements
  - Design
  - Implementation
  - Testing
  - Maintenance
- They may be combined and intertwined in varied ways
- There may be added constraints as well

Spiral model [Boehm]: example

- A disciplined sequence of activities intended to reduce risk
- Each quadrant is a different stage in planning and actions
- The length of the spiral represents the cumulative costs
- One 3/4 turn would a waterfall model



Comments?

Extreme programming: example

- Focus on
  - continuous, customer-oriented change
  - code and simplicity
  - rapid feedback
- Plus practices, rules of engagement, and more

Planning and feed back loops in Extreme Programming (XP) with the time frames of the multiple loops

Comments?

Other software process models

- Agile
- Iterative
- Capability Maturity Model Integration (CMMI)
- Test-driven development (TDD)
- Evolutionary development model
- Model-driven development
- …

Team structures

- Tricky balance among
  - progress on the project/product
  - expertise and knowledge
  - communication needs
  - …
- "A team is a set of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable." – Katzenbach and Smith

Why teams?

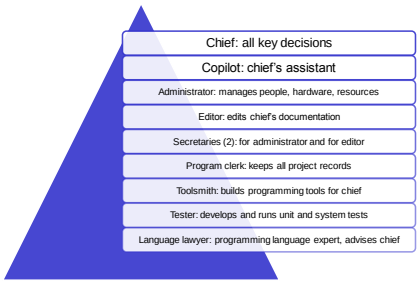
- Benefits
  - Attack bigger problems in a short period of time
  - Utilize the collective experience of everyone
- Risks
  - Personality conflicts
  - Coordination issues
  - Need to establish clear ownership or can have duplication of effort
  - Member can just "go along" instead of sharing potentially great ideas
  - Not taking individual responsibility/accountability because it's a group
  - Need to be careful to have the "right" number

Communication: powerful, costly!

- Communication requirements increase with increasing numbers of people
- Everybody to everybody: quadratic cost
- Every attempt to communicate is a chance to mis-communicate
- But not communicating will guarantee mis-communicating

Surgical/Chief Programmer Team

[Baker, Mills, Brooks]



Microsoft's team structure [microsoft.com]

- **Program Manager**. Leads the technical side of a product development team, managing and defining the functional specifications and defining how the product will work.
- **Software Design Engineer.** Codes and designs new software, often collaborating as a member of a software development team to create and build products.
- **Software Test Engineer.** Tests and critiques software to assure quality and identify potential improvement opportunities and projects.

Toshiba Software Factory [Y. Matsumoto]

- Late 1970's structure for 2,300 software developers producing real-time industrial application software systems (such as traffic control, factory automation, etc.)
- Unit Workload Order Sheets (UWOS) precisely define a software component to be built
- Assigned by project management to developers based on scope/size/skills needed
- Completed UWOS fed back into management system
- Highly measured to allow for process improvement

SCRUM: pigs and chickens

- **Product Owner** represents the customer
  - Ensures that the team maintains a proper business perspective
  - Writes user stories, prioritizes them, etc.
- **ScrumMaster** facilitates
  - Acts as a buffer between the team and distracting influences
  - Ensures that the Scrum process is respected
- **Team** delivers the product
  - Typically 5-9 people with skills to do the work (design, development, testing…)
- **Users** to whom the software will provide value
- **Stakeholders** (customers, vendors) who enable the project and for whom the project will produce the agreed-upon benefit
- **Managers** who set up the environment for the product development organizations
- These roles are far less directly connected to the process

Results-driven structure

- Clear roles and responsibilities
  - Each person knows and is accountable for their work
- Monitor individual performance, hold people accountable
  - Who is doing what, are we getting the work done?
- Effective communication system
  - Available, credible, tracking of issues, decisions
- Fact based decisions
  - Focus on the facts, not the politics, personalities, …

Typical SW team structures

- A person with project management responsibilities
- A person with functional management responsibilities
- Several "developers" in a broad sense: programmers,
- testers, integrators
- A person with lead developer/architect responsibilities

- These could be all different team members, or there could be a large amount of overlap.
- Key: Identify and stress roles **and** responsibilities

Alverson suggests

- Pragmatic Programmer
  - Pragmatic Teams, p. 224-230
- An interview with Patrick Lencioni on "The Five Dysfunctions of a Team"
  - http://www.managementconsultingnews.com/interviews/lencioni_interview.php
- Software Project Survival Guide
  - p.103-107 on team organization

- Also see Stepp's "team dynamics" lecture slides
  - http://www.cs.washington.edu/education/courses/403/08wi/lectures/slides/lecture05-teams.ppt

Questions?