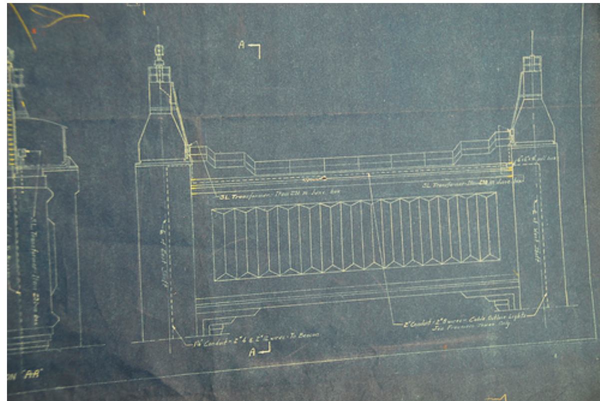
The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

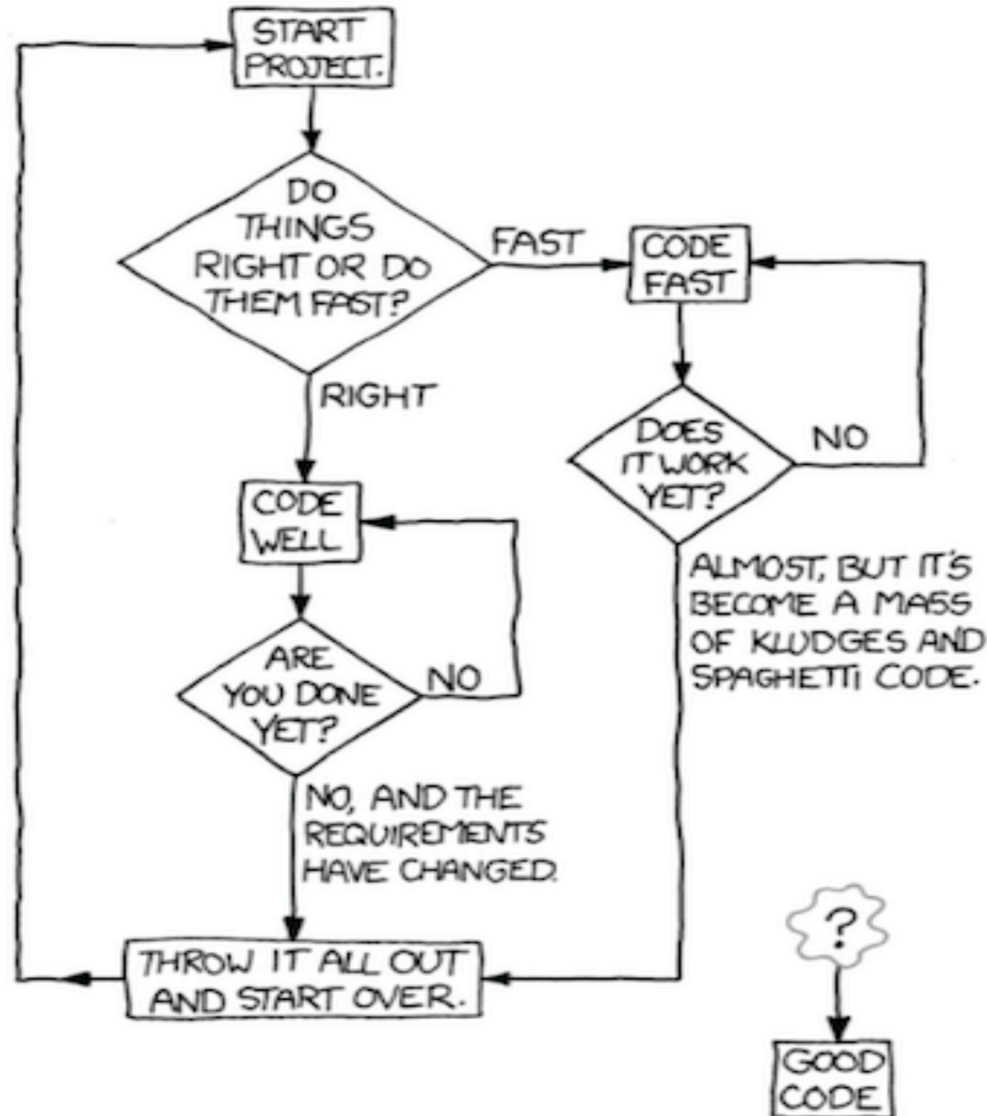
software development lifecycle (sdlc) models & agile methods

how did that happen?



- by analogy with civil engineering, where you design first, then do construction
- in software, there is no "construction" it's *all* design
- used to be called coding

sdlc (2)



sdlc (3)

- what is a software development process?
- what is the lifecycle of a software project?

- will talk about "agile" later. first, we'll talk about "disciplined" or is it "traditional?" or is it "sturdy?" or is it "planned?" or is it…

sdlc (4)

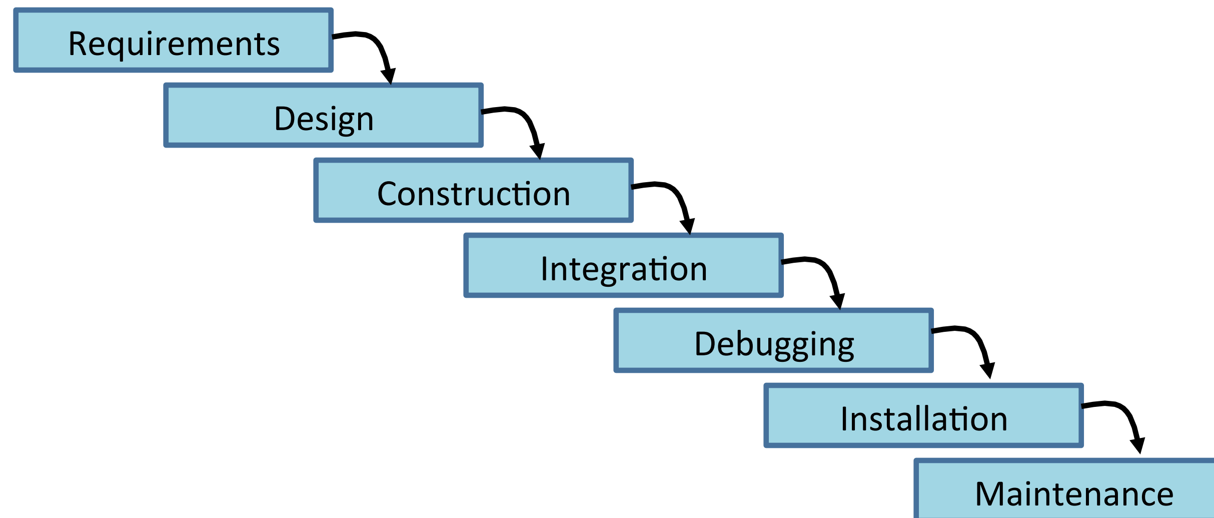
- tend to talk about sdlc in terms of a dichotomy
  - "agile" vs. well…um…"not agile"
  - or, "planned" vs. "continuous"
  - others tend to (incorrectly) think that the deployment method implies the process
    - saas == agile
    - installed == traditional
- think more in terms applying the process on an individual feature, or an aggregate

example feature workflow

goal of sdlc

- what's the goal of a good sdlc?
  - passes all the tests (external quality attributes)
  - good design/architecture (internal)
  - good user experience (quality in use)
  - process quality (can process help ensure product quality)

waterfall



- move from one phase to the next only when its preceding phase is completed and perfected.
- first mentioned by Royce in 1970 as an example of a flawed, non-working model for software development.
- US department of defence projects attempted to entrench this model by requiring their contractors to produce the waterfall deliverables and then to formally accept them to a certain schedule (US military standard DoD-2167)
  - there was a unwieldy process for going back and amending previous deliverables

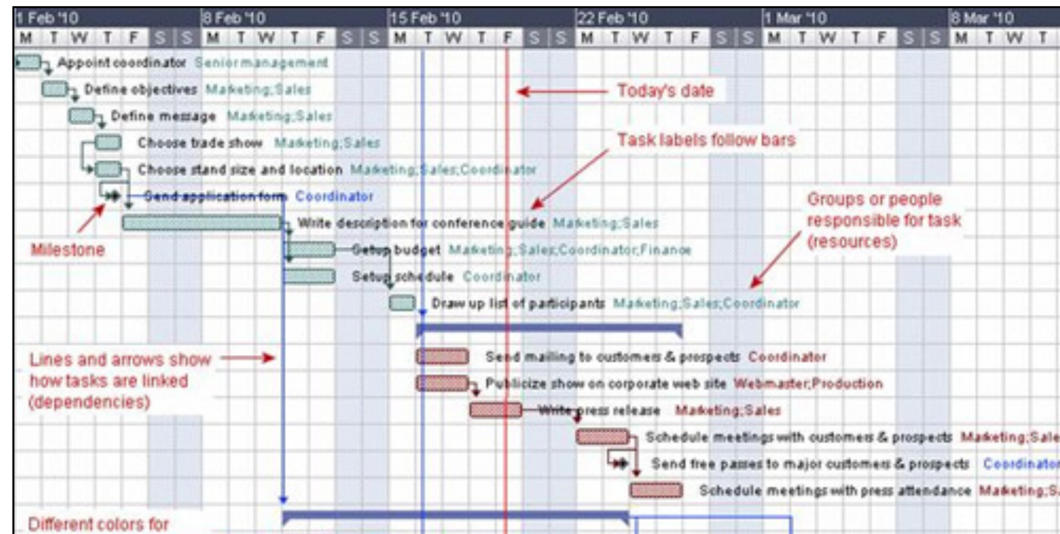
waterfall (2)

problems

- static view of requirements – ignores volatility
- lack of user involvement once specification is written
- unrealistic separation of specification from design
- doesn't easily accommodate prototyping, reuse, etc.

waterfall (3)

more problems



- often tracked with Gantt charts!
  - printed and taped up on the wall
  - out of date immediately
  - difficult to move tasks between developers
    - must assign all tasks before starting!
  - start writing in changes – disaster mess!

Bohem's cost of change

- *Software Engineering Economics* – Barry Boehm, 1981
  - data from waterfall-based projects in 1970s at IBM
  - acknowledged "architecture-breaker" flawed assumptions
  - small project – 1:4, large project – 1:100
  - also known as "software rot"

v-model

Level of abstraction

system requirements

software requirements

preliminary design

detailed design

code and debug

unit test

component test

software integration

acceptance test

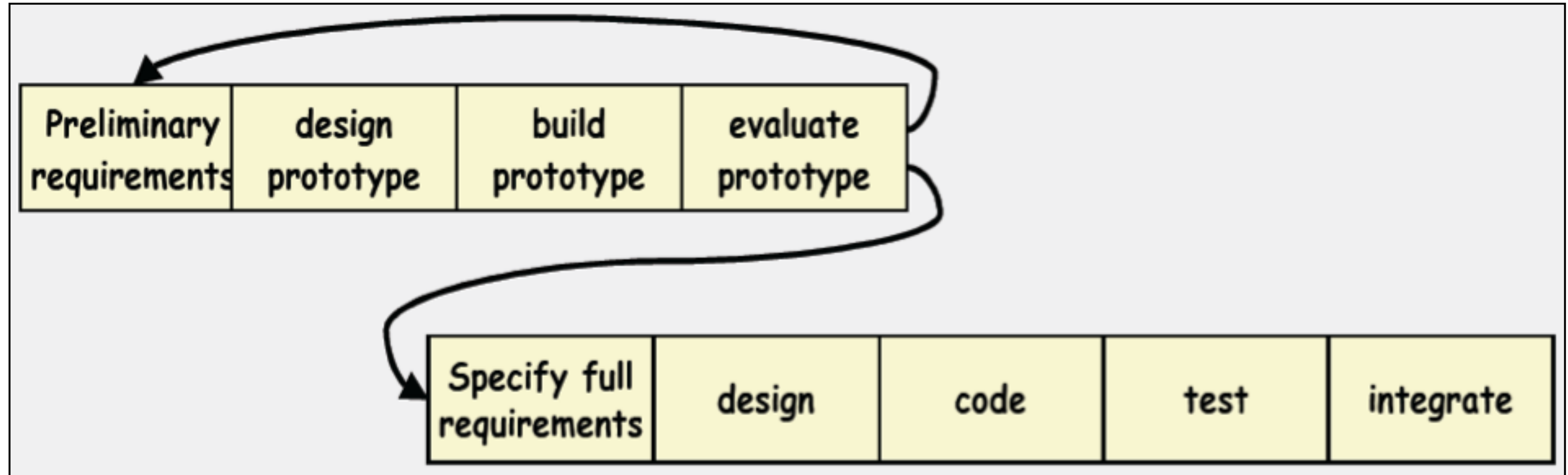
system integration

"analyse and design"

"test and integrate"

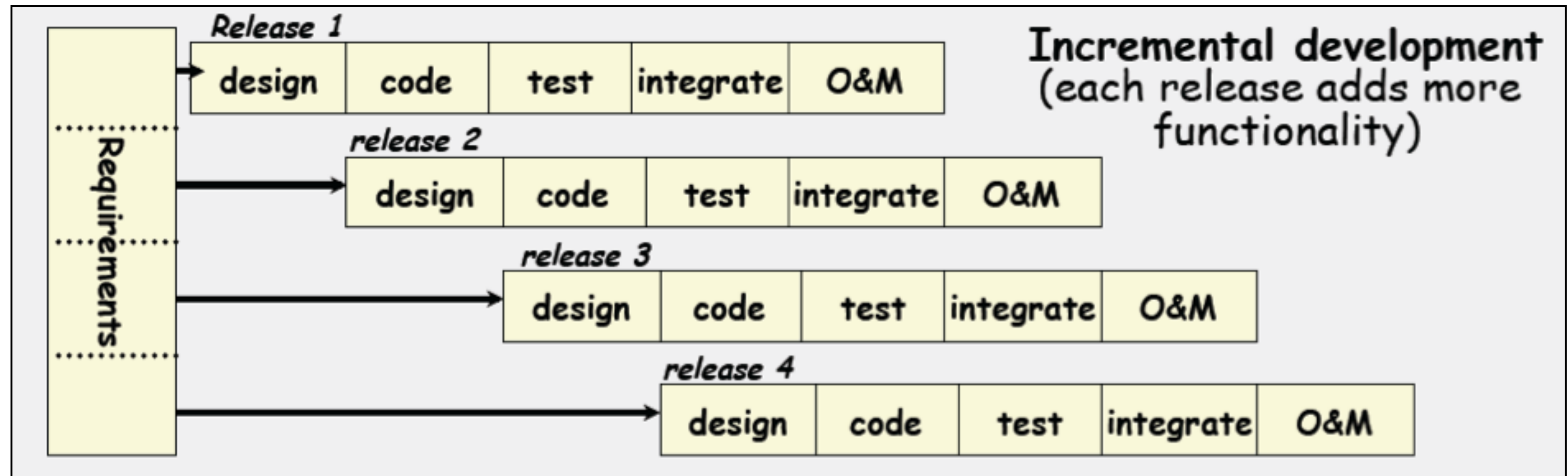
time

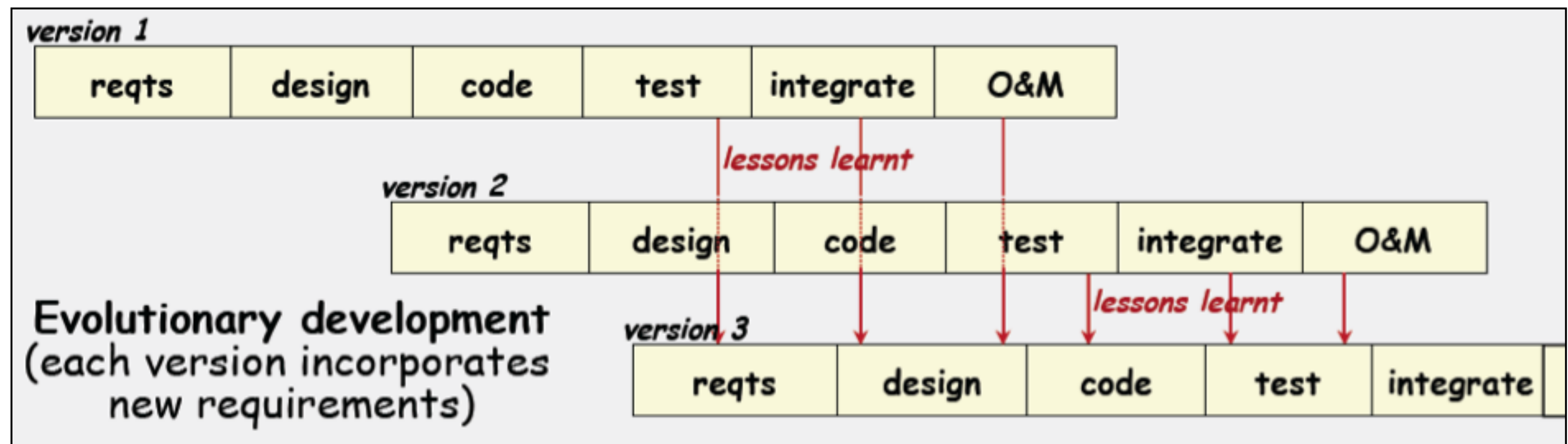
rapid prototyping



- prototyping used for:
  - understanding requirements for the user interface
  - determining feasibility of a proposed design
- problems:
  - users treat the prototype as the solution (or boss thinks it's done!)
  - prototype is only a partial specification

phased lifecycles





spiral model

Determine goals, alternatives, constraints

Evaluate alternatives and risks

Plan

Develop and test

constraints$_4$

constraints$_3$

Constr-aints$_2$

alternatives constraints

alternatives$_4$

alternatives$_3$

Altern-atives$_2$

risk analysis$_4$

risk analysis$_3$

risk analysis$_2$

risk analysis$_1$

budget$_4$ budget$_3$ budget$_2$ budget$_1$ prototype$_1$ prototype$_2$ prototype$_3$ prototype$_4$

requirements, lifecycle plan

development plan

integration and test plan

implementation plan

concept of operation

software requirements

software design

detailed design

code

unit test

system test

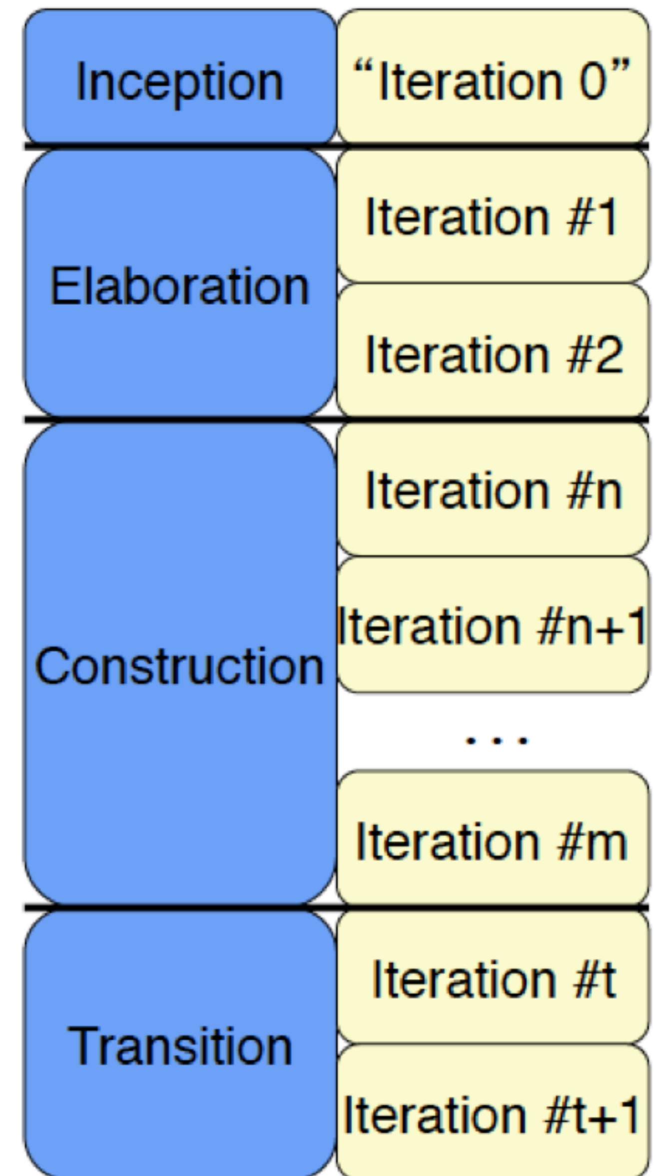
acceptance test

validated requirements

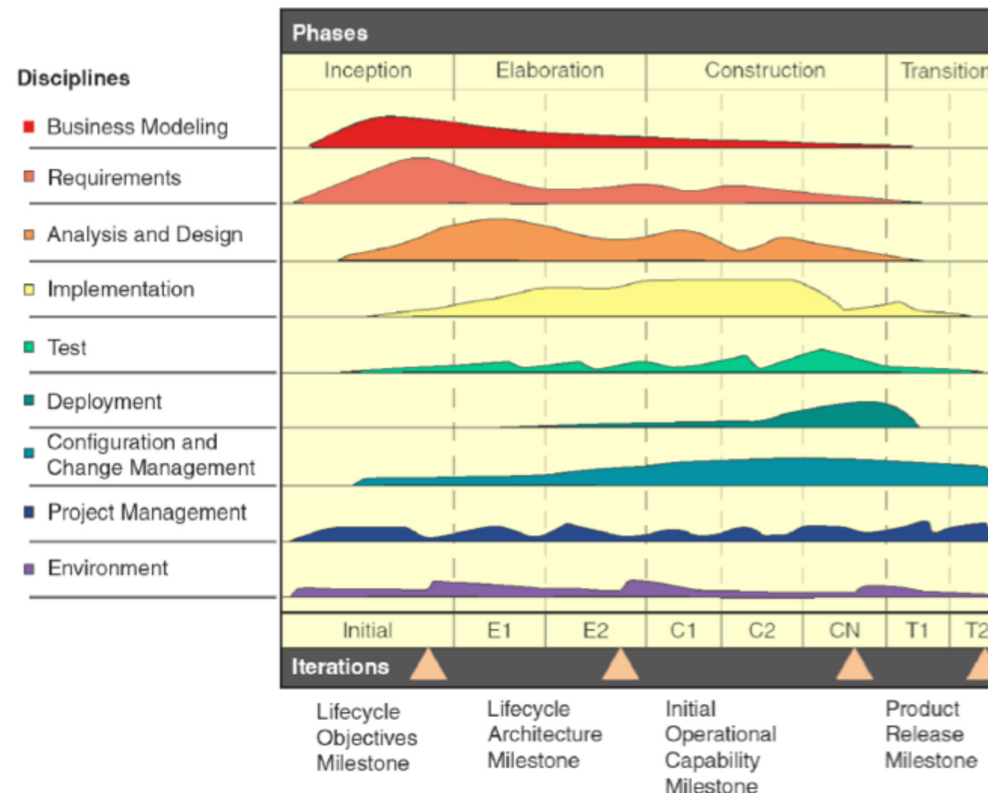
validated, verified design

rational unified process

- inception
  - establish scope
  - build business case
  - stakeholder buy-in
- elaboration
  - identify & manage risks
  - work out architecture
  - focus on high risk items
- construction
  - iterate & build operational version
  - develop docs & training material
- transition
  - fine-tune
  - resolve config, install & usability issues



rational unified process (2)



- framework created by Rational, acquired by IBM in 2003
- four phases:
  - inception: business planning, requirements gather
  - elaboration: mitigate risks, use cases, dev. plan, architecture, prototypes
  - construction: development, unit tests, QA
  - transition: user acceptance testing, training

agile methods

agile

- refers to a group of software development methodologies created as a reaction against the heavily regulated, regimented, micro-managed use of the waterfall model ("pure waterfall").
- developed in the mid-1990's as "lightweight methods". most popular ones to survive are:
  - scrum – 1995
  - extreme programming (XP) - 1996
- "agile" term was first used in 2001.

agile manifesto

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

12 agile principles

- 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.

12 agile principles (2)

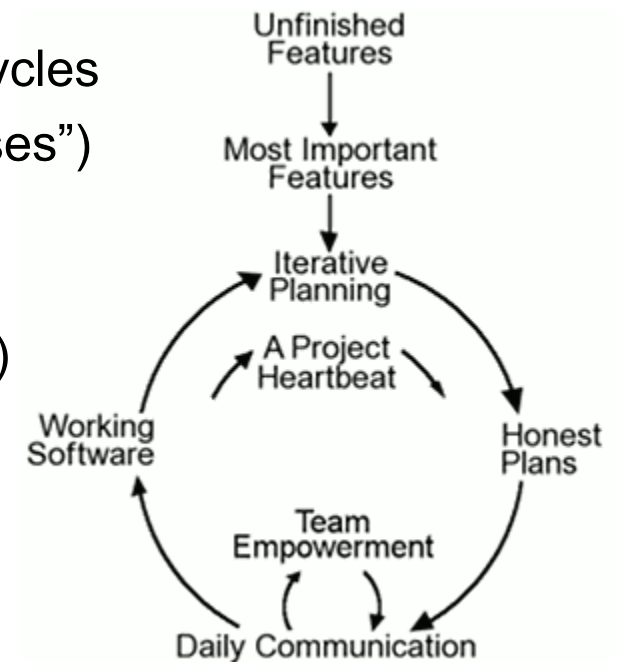
- build projects around motivated individuals. give them the environment and support they need, and trust them to get the job done.
- the most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- working software is the primary measure of progress.
- agile processes promote sustainable development. the sponsors, developers, and users should be able to maintain a constant pace indefinitely.

12 agile principles (3)

- 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 (XP)

- XP = eXtreme Programming (Beck 1999)
- frequent "releases" in short development cycles
- manage by features ("user story" / "use cases")
  - release planning / iteration planning
- continuous integration
- pair programming (continuous code review)
- unit testing of all code
- avoiding programming of features until they are actually needed
- simplicity and clarity in code
- frequent communication (customers and coders)
- expecting changes in the customer's requirements as time passes and the problem is better understood
- coding standard
- collective code ownership
- sustainable pace



XP alternate

Collect User stories

Planning game

Write test cases

code

integrate

test

Release

Each cycle:
approx 2 weeks

XP alternate (2)

Collect User stories

Planning game

Write test cases

Suggested Feature (Task & Sub-Task) Workflow for TPA R&D

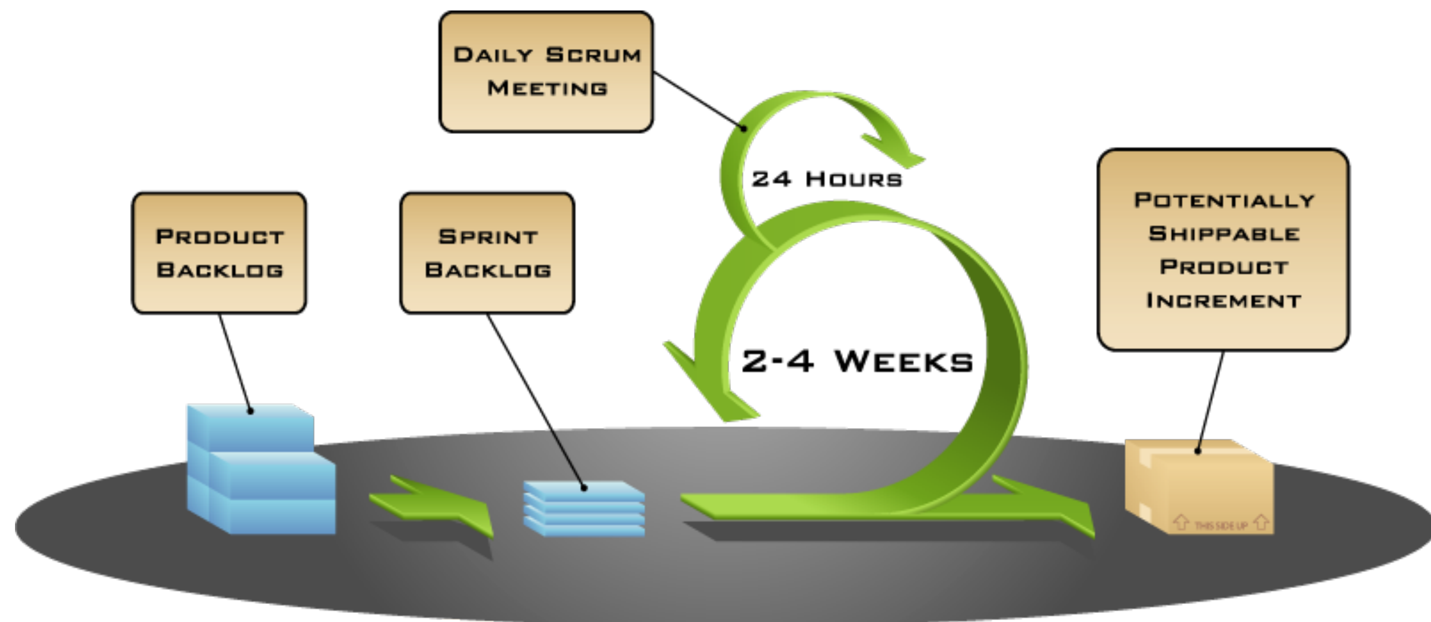
PM/R&D
Backlog
R&D
PM/R&D
Trash
In-Plan
MUST have A/B-List attribute & size. SHOULD have developer assigned.
R&D
Started
MUST have a developer assigned.
R&D
Finished
R&D
Delivered
R&D
Accepted
R&D
Closed

Release

Each cycle: approx 2 weeks

scrum

- scrum (Schwaber & Beedle 2001)
- product owner, team, scrum master
- “sprints” 2-4 weeks
- “stories” are described and sized in “units” or “points”
- team commits to number of “points” they can do in next sprint
- product owner picks stories accordingly
- product owner tests stories and gives feedback after each sprint



scrum fable

- in 2011 this fable was removed from the scrum process
  - pigs (committed): project owner, scrum master, development team
  - chickens (involved): customers, executive management
  - rooster: struts around offering unrequested, uninformed & unhelpful opinions
  - analogy is breakfast – bacon & eggs

"Agile" vs "Sturdy"

Iterative	⟷	Planned
Small increments	⟷	Analysis before design
Adaptive planning	⟷	Prescriptive planning
Embrace change	⟷	Control change
Innovation and exploration	⟷	High ceremony
Trendy	⟷	Traditional
Highly fluid	⟷	Upfront design / architecture
Feedback driven	⟷	Negotiated requirements
Individuals and Interactions	⟷	Processes and Tools
Human communication	⟷	Documentation
Small teams	⟷	Large teams

personal experience

- feature-driven development is not in question.
  - almost nobody believes in pure waterfall
  - written reqs/specs/design for *entire* release ≈ waterfall
  - written requirements/spec/design per feature when necessary ≠ waterfall
    - advocated where necessary in agile
- continuous integration, keeping the code in good shape at all times & automated architectural regression testing? **yes!**
- full unit tests? **usually impractical**
- pair programming? **sometimes, maybe**
- frequent communications? **yes!**
  - involving stakeholders? **yes (if they will attend!)**
- simple design with constant re-factoring? **yes, mostly**
  - but too extreme to *never* design for the future.

personal experience (2)

- commit only to next sprint? **not practical**
- use of "points" as opposed to a time unit? **no**
  - everyone outside of development will not trust it
- coding standards and collective code ownership? **yes**
- eliminate final test phase? **not practical**
  - reduce it with code/test iterations within the coding phase
- use working software as the primary measure of progress? **yes, for the most part**
  - for big-bang releases, I advocate:
    - feature demos during the development process.
    - independent function testing during the coding phase.
    - reflect on release plan when a feature is done by above def'n.
    - relentlessly plan and manage to dcut (= feature complete)

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

personal experience (3)

- welcome changing requirements? **can’t avoid**
  – but within a planning framework. cannot welcome all changes without considering the impact on the end-dates.
- sustainable development? **yes**
  – but unrealistic without careful planning
- the best architectures, requirements, and designs
  emerge from self-organizing teams? **not convinced**
- beware: it’s easy to proudly claim agile but actually be doing cowboy development!

which process is the best?

- all processes have their pros and cons, but only in the context of a given project.
  - does continuous deployment make sense for the next version of microsoft office?
  - what process is best for an x-ray machine?
  - space shuttle avionics – hal/s developed specifically for shuttle
    - completely independently developed primary and backup systems!
  - curiosity rover software, installed in flight! and then upgraded on mars!
- again, depends on the nature of the project

summary

- do these things, and you are doing well!