Lecture 2

Processes and DevOps

SOEN 6441, Summer 2018

René Witte
Department of Computer Science
and Software Engineering
Concordia University

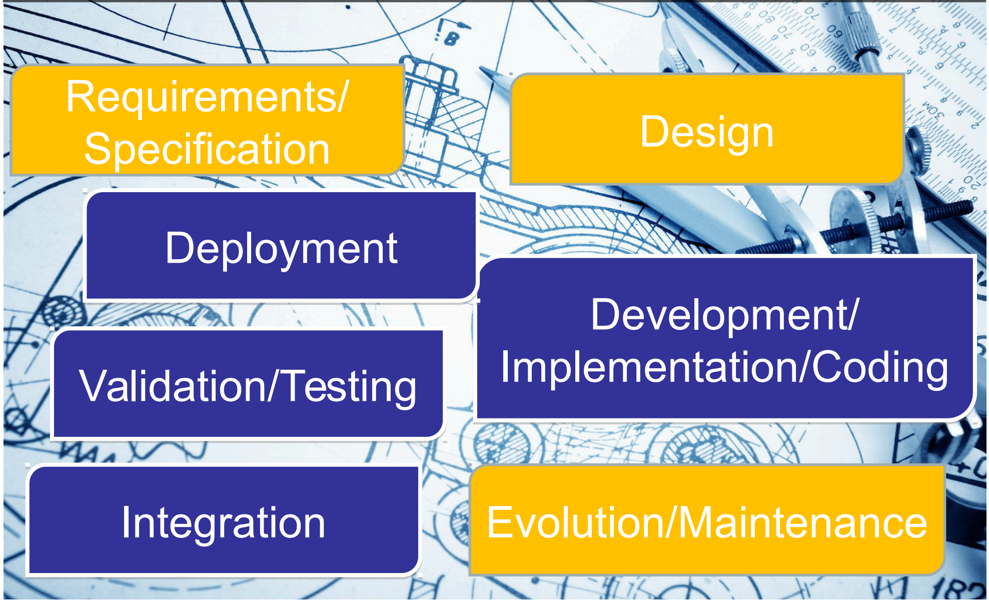
Outline

1. **Software Development Process**

2. **DevOps**
   - Why DevOps?
   - What is DevOps?
   - DevOps Practices
   - DevOps Consequences

3. **Notes and Further Reading**

Main software engineering activities



Requirements/ Specification

Design

Deployment

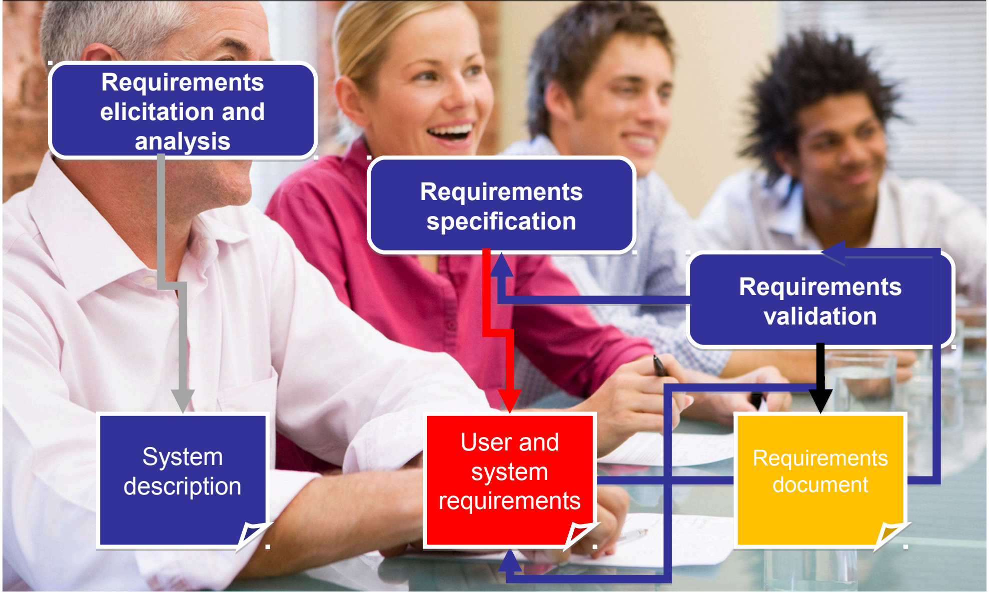
Development/ Implementation/Coding

Validation/Testing

Integration

Evolution/Maintenance

Requirement/Specification



Requirements elicitation and analysis

Requirements specification

Requirements validation

System description

User and system requirements

Requirements document

Requirement/Specification

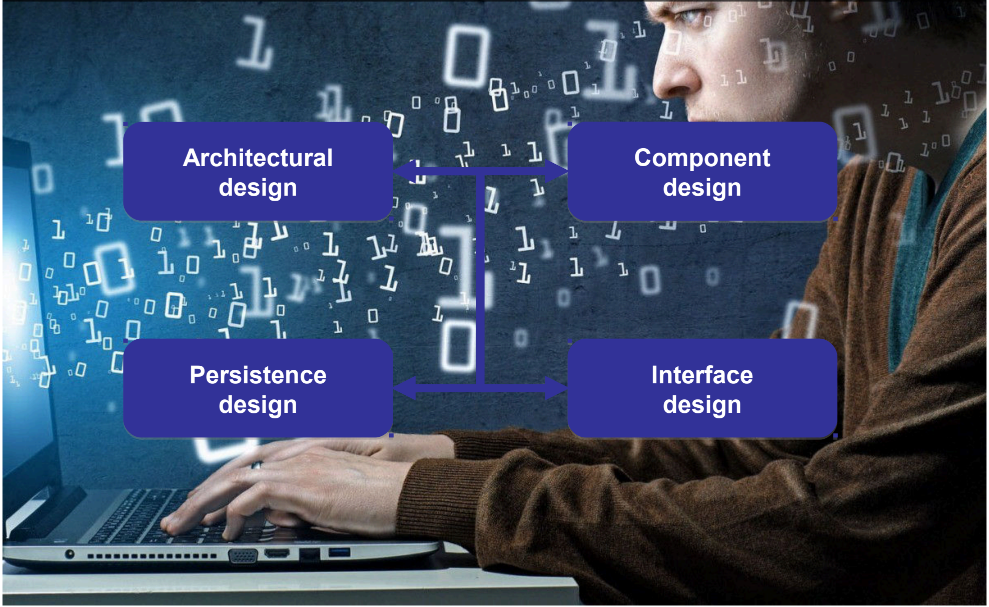
- Software Elements Analysis
  - Extracting the requirements. Customers typically know what they want, but not what software should do. Demonstrating live code may help reduce the risk that the requirements are incorrect.

- Scope Analysis
  - An analysis of the scope of the development should be determined and clearly stated. Certain functionality may be out of scope of the development project as a function of cost or as a result of unclear requirements

Requirement/Specification

- Specification is the task of precisely describing the software to be written.
- Most successful specifications are written to understand and fine-tune applications that were already well-developed, although safety-critical software systems are often carefully specified prior to application development.
- Specifications are most important for external interfaces that must remain stable.

Design



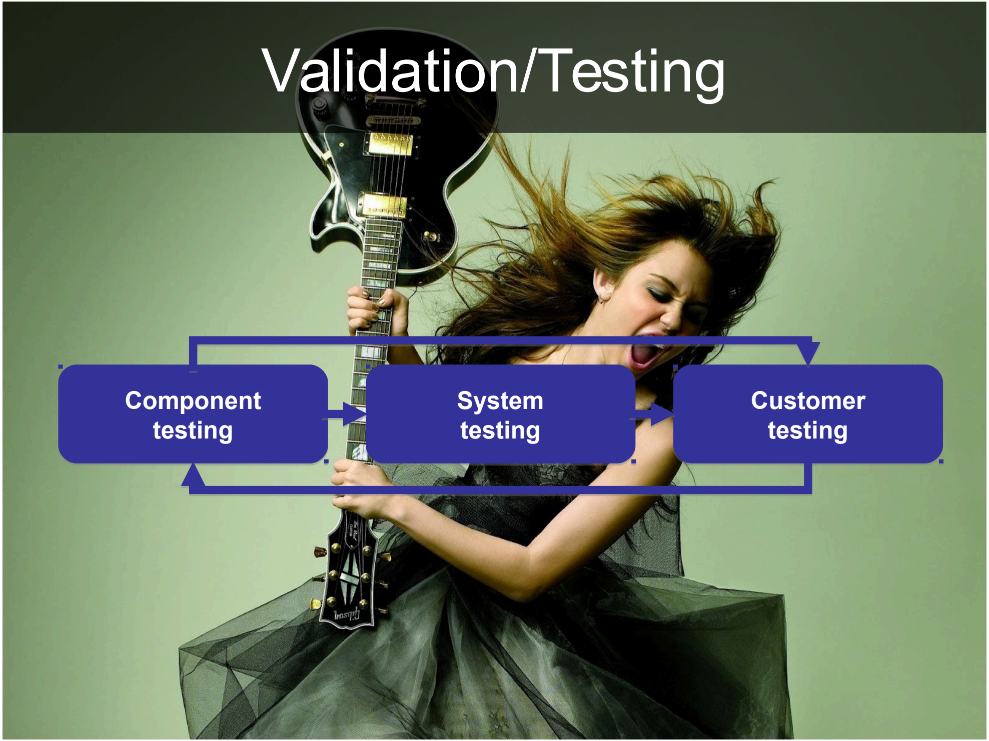
Architectural design

Component design

Persistence design

Interface design

Validation/Testing



Component testing

System testing

Customer testing

Validation/Testing

Component testing

- Done by the person who writes the code
- Often considered as part of coding

System testing

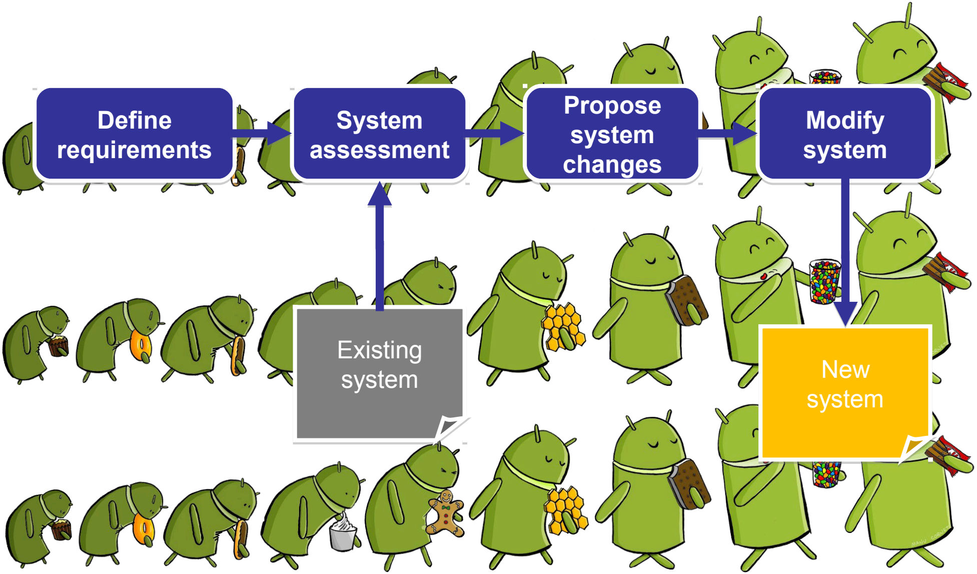
- Feature testing and performance testing
- Different levels of system testing
- Regression testing

Validation/Testing

Customer testing

- Acceptance testing
- Field testing

Evolution/Maintenance



Define requirements

System assessment

Propose system changes

Modify system

Existing system

New system

Evolution/Maintenance

- Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more time than the initial development of the software.
- A small part of that is fixing bugs. Most maintenance is extending systems to do new things, which in many ways can be considered new work.

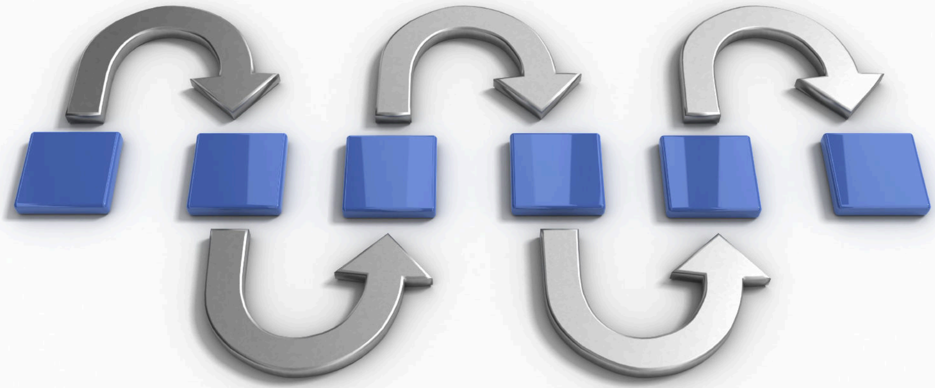
Other activities

- Coding
  - Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion.
- Deployment
  - After the code is appropriately tested and approved, it is moved into production environment i.e. is made available for business use.

Other activities

- Documentation
  - An important (and often overlooked) task is documenting the internal design of software for the purpose of future maintenance and enhancement. *Documentation is most important for external interfaces.*
- Software Training and Support
  - A large percentage of software projects fail because the developers fail to realize that it doesn't matter how much time and planning a development team puts into creating software if nobody in the client organization ends up using it.

Activities are connected to produce a software engineering process



Building vs. growing

Building software

The “building” metaphor: planning; specification as blueprint; components; assembly; scaffolding; etc. But the idea that planning *preceded* construction remained.

Growing software rather than build it.

Start with a very simple system that runs but has minimal functionality and then add to it and let it grow.

Process 0: The code-and-fix model

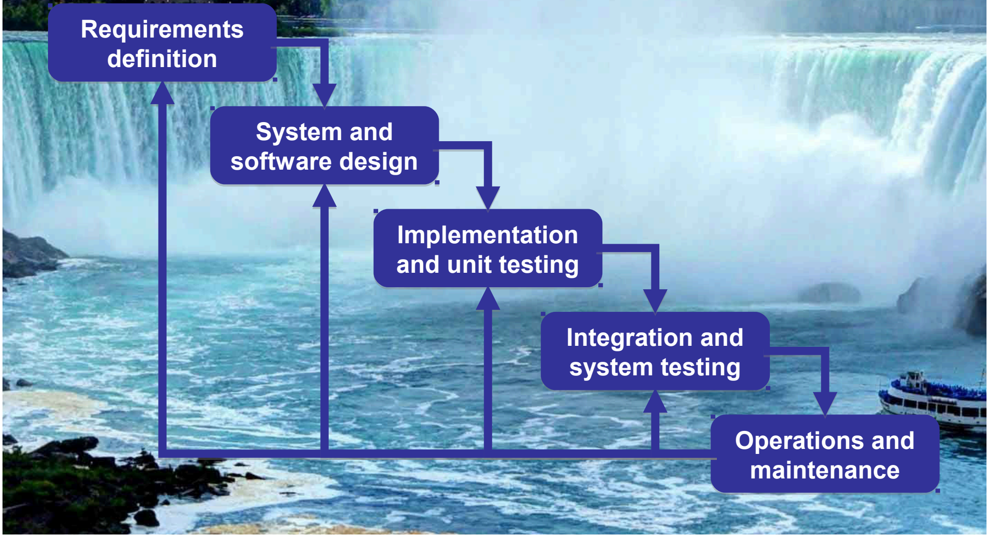
- The basic model used in the earliest days of software development contained two steps:
  1. Write some code.
  2. Fix the problems in the code.

Process 0: The code-and-fix model

- After a number of fixes, the code became so poorly structured that subsequent fixes were very expensive. This underscored the need for a design phase prior to coding.
- Frequently, even well-designed software was such a poor match to users' needs that it was either rejected outright or expensively redeveloped. This made the need for a requirements phase prior to design evident.
- Code was expensive to fix because of poor preparation for testing and modification. This made it clear that explicit recognition of these phases, as well as test and evolution planning and preparation tasks in the early phases, were needed.

Process 1: Waterfall

Process 1: Waterfall



Requirements definition

System and software design

Implementation and unit testing

Integration and system testing

Operations and maintenance

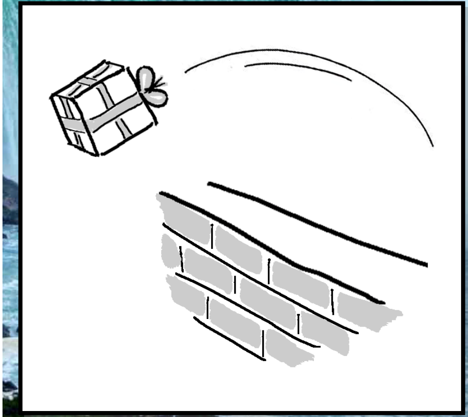
Pros of the Waterfall process

Rigid and formal process, fits well for:

-Embedded systems

-Safety-critical systems

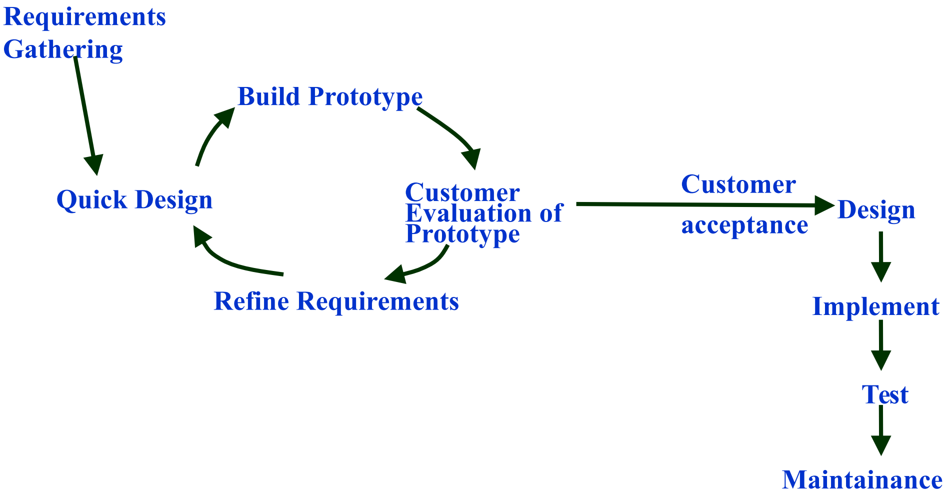
Cons of the Waterfall process



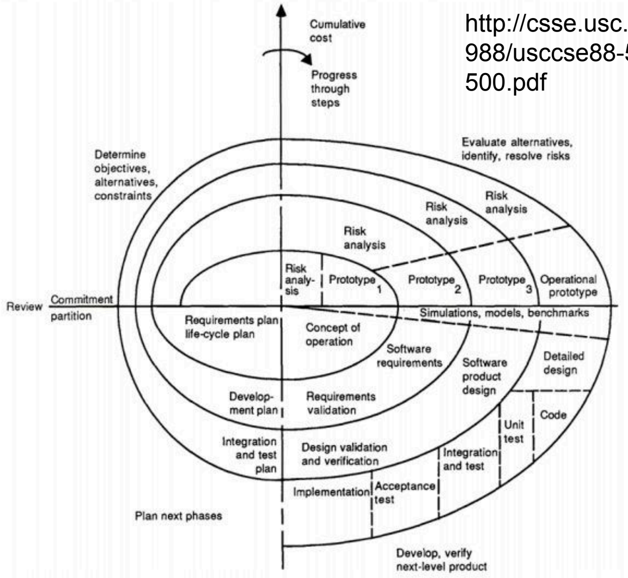
Activities are isolated:

-Late-changing requirements require a lot of rework!

Process 1.5: Prototype model

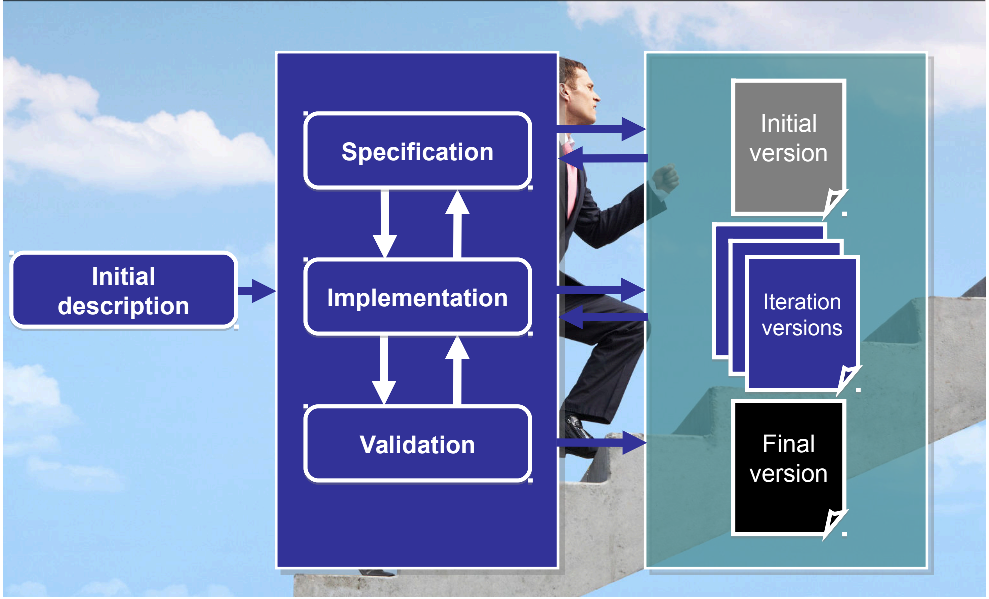


Process 2: Spiral model

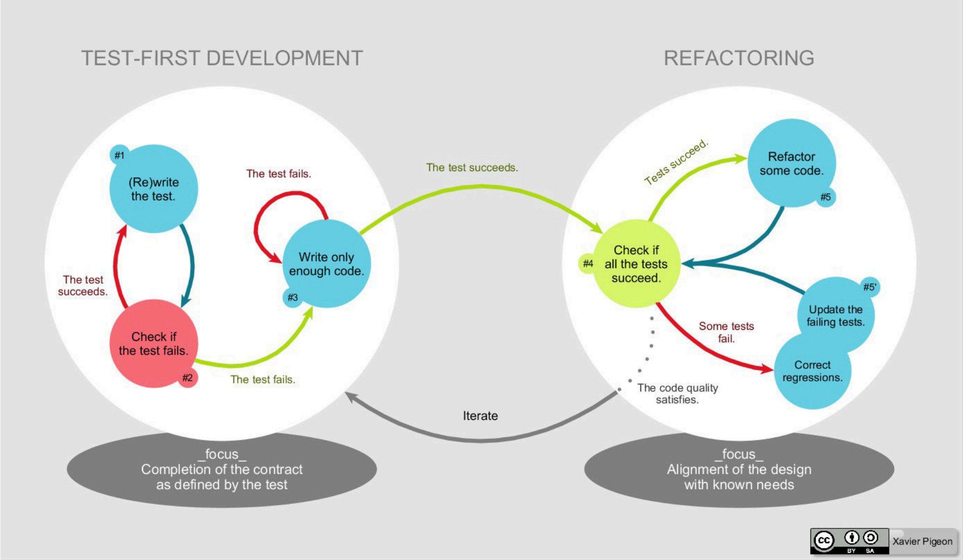


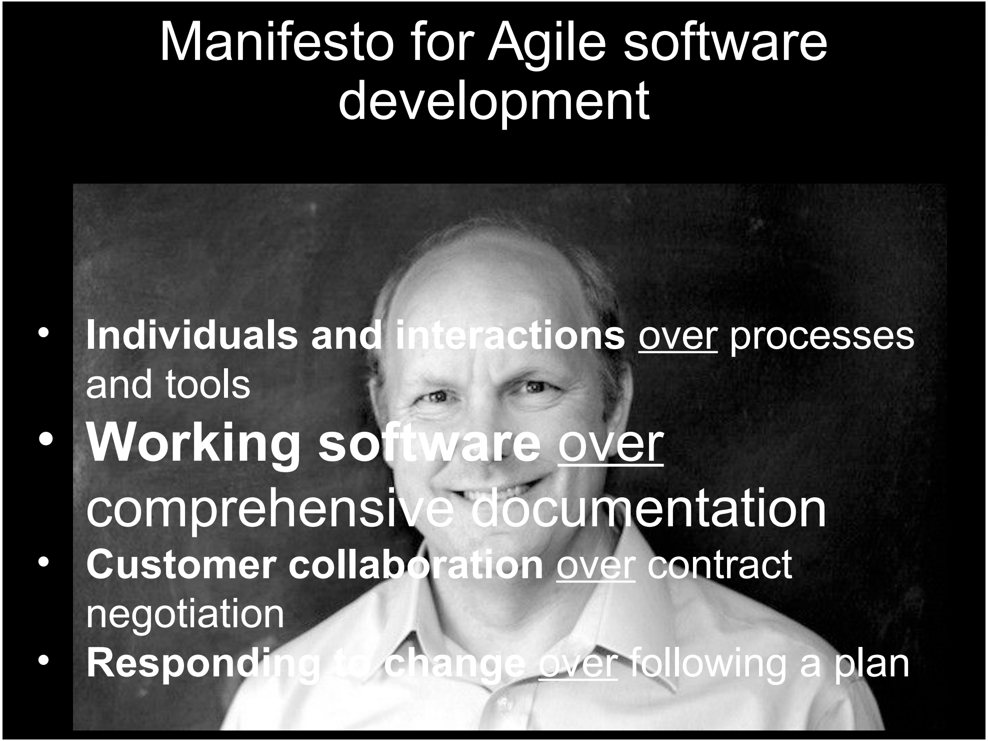
http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

Process 3: Incremental Development



Process 3.5 Test-Driven development





Manifesto for Agile software development

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Outline

1 Software Development Process

2 DevOps

- Why DevOps?
- What is DevOps?
- DevOps Practices
- DevOps Consequences

3 Notes and Further Reading

Why DevOps?

- Developers and operators do not pursue the same goals.

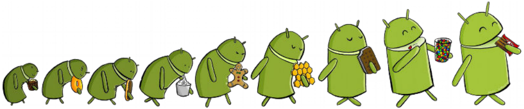
Software Development

Design and specification

Coding

Testing

Release engineering



Evolution

Software Operation

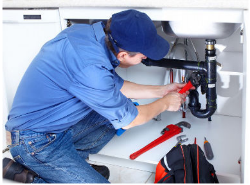
Monitoring

Troubleshooting

Capacity planning

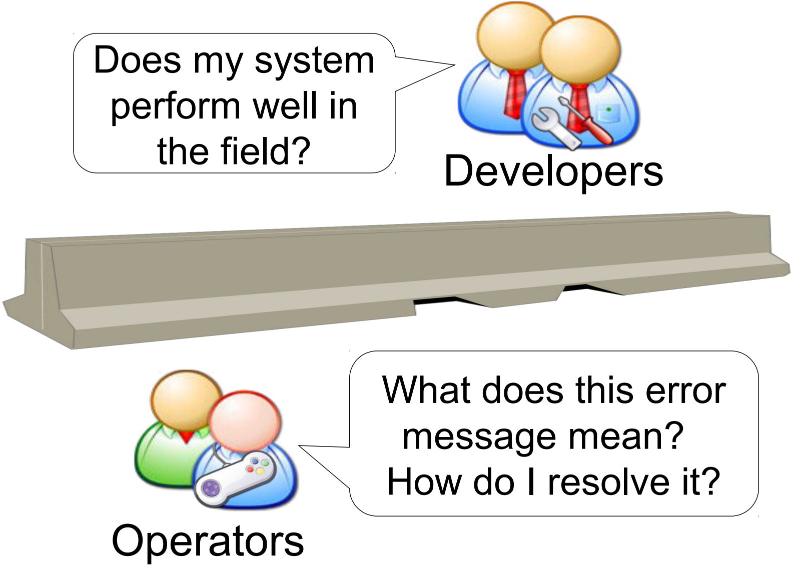
Anomaly detection

Q&A



Configuration Tuning

There is a gap between software developers and operators



Does my system perform well in the field?

Developers

Operators

What does this error message mean? How do I resolve it?

What problem is DevOps trying to solve?

- Poor communication between Dev and Ops
- Opposing goals
  - Devs want to push new features
  - Ops want to keep the system available
    - ▭ Leads to slow release schedule
- Different cultures
- Limited capacity of operations staff
- Developers have limited insight into operations

Why companies care?

- IBM
  - “IBM has gone from spending about **58%** of its development resources on innovation to about **80%**”
  - http://devops.com/blogs/ibms-devops-journey/?utm_content=12855120
- Paddy Power (Ireland):
  - “The cycle time from a user story's conception to production has decreased **from several months to 2 to 5 days.**
  - “Previously, approximately 30% of the workforce was fixing bugs. Now, **bugs are so rare** that the teams no longer need a bug-tracking system.”

DevOps motivation

- Organizations want to reduce time to market for new features, without sacrificing quality
  - Requires culture change & business-IT alignment
- DevOps practices will influence…
  - the way you organize teams
  - the way you build systems
  - even the structure of the systems that you build
- Unlikely that Devs are able to “throw their final version over the fence” and let operations worry about running it

What is DevOps?

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.

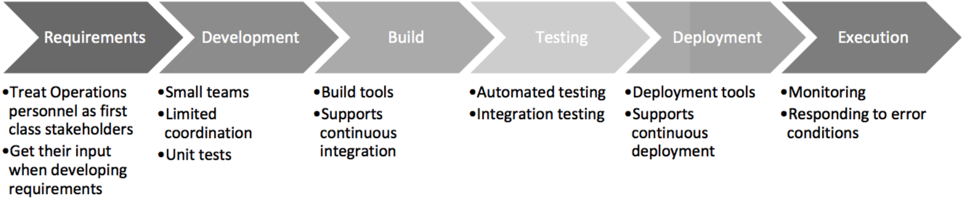
DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

DevOps is also characterized by operations staff making use many of the same techniques as developers for their systems work.

What does that mean?

- Quality of the code must be high
  - Testing & test-driven development
- Quality of the build & delivery mechanism must be high
  - Automation & more testing
  - A must when deploying to production 25x per day (etsy.com)
- Time is split:
  - From commit until deployment to production
  - From deployment until acceptance into normal production
  - Means *testing* in production
- Achieving that starts before committing

DevOps Practices



Requirements	Development	Build	Testing	Deployment	Execution
•Treat Operations personnel as first class stakeholders •Get their input when developing requirements	•Small teams •Limited coordination •Unit tests	•Build tools •Supports continuous integration	•Automated testing •Integration testing	•Deployment tools •Supports continuous deployment	•Monitoring •Responding to error conditions

- Treat Ops as first-class citizens throughout the lifecycle – e.g., in requirements elicitation
  - Many decisions can make operating a system harder or easier
  - Logging and monitoring to suit Ops
- Make Dev more responsible for relevant incident handling
  - Shorten the time between finding and repairing errors

DevOps Practices

- Use continuous deployment, automate everything
  - Commits trigger automatic build, testing, deployment
- Enforce deployment process is used by all
  - No ad-hoc deployments
  - Ensures changes are traceable
- Develop infrastructure code with the same set of practices as application code
  - “Infrastructure as Code” : using IaaS APIs, etc., to automate creation of environments
  - Misconfiguration can derail your application
  - Ops scripts are traditionally more ad-hoc

DevOps Consequences

Architecturally significant requirement:

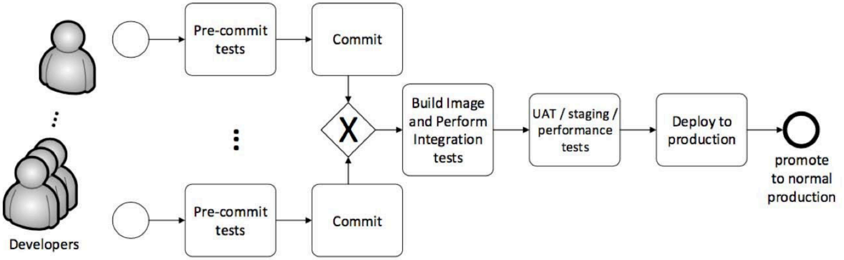
Speed up deployment through minimizing synchronous coordination among development teams.

- Synchronous coordination, like a meeting, adds time since it requires
  - Ensuring that all parties are available
  - Ensuring that all parties have the background to make the coordination productive
  - Following up to decisions made during the meeting

DevOps Consequences

- Keep teams relatively small
  - Amazon’s “two pizza rule”: no team should be larger than can be fed with two pizzas
  - Advantages: make decisions quickly, less coordination overhead, more coherent units
- Team size becomes a major driver of the overall architecture:
  - Small teams develop small services -> Microservices
  - Coordination overhead is minimized by channeling most interaction through service interfaces:
    - Team X provides service A, which is used by teams Y and Z
    - If changes are needed, they are communicated, implemented, and added to the interface.

Continuous Deployment Pipeline


Pre-commit tests
Commit
Build Image and Perform Integration tests
UAT / staging / performance tests
Deploy to production
promote to normal production
Pre-commit tests
Commit
Developers

- Developer wants to commit code
- Pre-commit tests are executed locally. If successful:
- Code is committed
- Committed code is compiled, Unit tests are run. If successful:
- Code is built & packaged
  - Result can be a machine image or template (assuming virtualization). If successful:
- Integration tests are run. If successful:
- Acceptance / performance tests are run. If successful:
- The new software/service is deployed to production

Outline

1. Software Development Process
2. DevOps
   - Why DevOps?
   - What is DevOps?
   - DevOps Practices
   - DevOps Consequences
3. Notes and Further Reading

Reading Material

Supplemental

- [Som16, Chapters 2, 3] (Software Processes)

References

[Som16] Ian Sommerville.
Software Engineering.
Pearson, 10th edition, 2016.
http://software-engineering-book.com.