SEG4910/11 – **Projet génie logiciel en fin d'études / Software Engineering Capstone Project – 2023 cohort**

Course notes

Timothy C. Lethbridge

Some slides derived from notes by Liam Peyton

Tim Lethbridge, Ph.D., P.Eng.

- Professor at Uottawa; full-time since 1994
  - Software Engineering
    - Usability, software tools, knowledge engineering, code generation
- Former software developer at Nortel and the Government
- Researcher with GM, IBM, Boeing, Ericsson and smaller companies

- Current research focus:
  - Model-Oriented Programming (Umple) and UX

Cours Bilingue ….

- Vous pouvez travailler en français
- Vous pouvez faire des présentations et les documents pour le projet en français
- Vous pouvez poser ou répondre aux questions en classe en français

  - En général, je communiquerai en anglais car il y a des étudiants unilingues
  - Mais si vous me posez une question en français, je répondrai en français

Everything's online / Tout est en ligne

- Brightspace
  - Videos of sessions / schedule
- Microsoft Teams
  - Course announcements
  - Chat / video with each team
- Website
  - http://www.site.uottawa.ca/~tcl/seg4910-11/
  - Class schedule
  - These notes (updated for Jan 2023)

Most of you have teams, some have finalized projects

- We have been working on projects for 2 months prior to the course
- If you don't have a project, see listings on Microsoft Teams
  - Contact other students who say they are looking, and post your own name
  - Email clients if a project is still available
  - Direct message students and the prof

One Project – 2 Courses

- 1 Project
  - customer (meet regularly, once every week or two)
    - Type 1: Customer has problem
    - Type 2: Open market – customer represents a user in the market
  - Groups: 2-5 students (possible exceptions)
  - Typically there is a group leader (can take turns)
  - Workload: 3-4 weeks per person per semester
    - (12 weeks at 12-15h/wk per student)

- Start in 4910, finish in 4911
  - Must have same project, same customer(s), same group for both courses, otherwise you have to retake the entire sequence 4910 / 4911

SEG4910 attends in the same time slots as SEG4911

- SEG4911 students are finishing their project
- SEG4910 will learn from their presentations

- Only some timeslots will be used
- See the schedule to know when to attend

Team roles / Rôles d'équipe

- Shared among students / Partagé entre étudiants
  - Project Manager — Gestionnaire de projet
  - Business Analyst — Analyste d'affaires
  - QA manager — Responsable de l'AQ
  - Architect — Architecte
  - Build Manager — Gestionnaire de build
  - Lead Developers — Développeurs principaux

  - But everybody does some design and coding
    - tout le monde fait du design et du codage

The Real Customer / Le vrai client

- Someone who wants to or would be willing to use the product after you have finished the project
- Quelqu'un qui veut ou serait prêt à utiliser le produit une fois que vous avez terminé le projet

- The customer follows the work from concept to deployment over 2 semesters
- Key to Agile approach
  - The customer sees and comments on your work every week or so, hence is 'on site' virtually

Deliverables to professor

- ASAP: In your Teams conversation (created when your team is formed)
  - Project title and brief description to be approved by the prof
  - Name and email of customer if you have not selected from the list of projects provided by the prof

- Week 2 of 4910: Github/Gitlab repo invitation or link posted to Teams conversation, then project initial overview in the Wiki
  - Some leeway will be allowed for groups that show they are working hard on getting going by communicating to the prof

Deliverables to professor

- At week 7 and 12: Each individual posts a link in Teams to a URL listing their commits
- By week 7: Each individual has completed self-evaluation (Prof will send URL)
- End semester in 4910; mid semester in 4911: Presentation(s)
- End semester in 4911: Motivational demo
- Attending class most weeks (80%) according to schedule
- Ongoing. Commits; PRs; issues; Wiki updates

Project initial overview

- Must be finalized by SEG4910 week 2 – on your wiki or Readme.md
  - Outline
  - Team members and their roles
  - Objectives (benefit to customer, key things to accomplish, criteria for success)
  - Expected/anticipated architecture
  - Anticipated risks (engineering challenges, etc.)
  - Legal and social issues.
  - Initial plans for first release, tool setup, etc.

  - Put the above on the Wiki and privately send me your customer's name, title and contact info
    - Send by Team channel for your group if channel is set up

Example recent very successful projects (1/2)

- Machine learning from network data to detect malware (several)
- Making dev tools available on the web
- Managing sets of 3D printers
- Analyzing video of sports to automate stats (company started … then more 491x)
- An app to help medical specialists diagnose strokes (used by patients and doctors)

Example recent very successful projects (2/2)

- App to help local farmers distribute produce directly to consumers
- Co-op navigator mobile app
- Math educational tool
- Crowdsourcing tool for legal cases (3 years)
- Generator of Swagger from software
- Airport information system
- Warehouse robot routing

Example recent causes of poor grades (1/4)

- Working full time at paid employment (sometimes not even letting teammates know)
- Writing trivial amounts of simple code
- Not struggling hard to solve a technical problem (”I tried” is not enough in the capstone)
- Not searching effectively online for solutions or experimenting

Example recent causes of poor grades (2/4)

- Stopping work for weeks and blaming midterms
- Trying to catch up at week 8 (or week 10)
- Arguing with clients in an unprofessional way
- Not testing enough, so clients and the prof find too many problems
- Not trying out enough with real users
- Not polishing the product

Example recent causes of poor grades (3/4)

- Spending weeks trying to learn a language in depth (best to dive in and learn as you go)
- Trying to get away with letting team-mates do more than their fair share
- Not delivering what they have promised to teammates, on time
- Trying to do too much or too little, given team size

Example recent causes of poor grades (4/4)

- Making beautiful looking graphics … *only*
- Not contacting the prof quickly enough when a problem starts to get out of hand
- Focusing on the business side of a startup, and not producing software
- Building something with little innovation
- Making something that won't realistically be able to attract any real users

Overall grading scheme
Schéma de notation global

- FINAL_MARK=(**Default team grade** *
  Complexity adjustment factor)
  + Individual factors

- NOTE_FINALE=(**Note d'équipe par défaut** *
  Facteur d'ajustement de la complexité)
  + Facteurs individuels

Overall grading scheme
Schéma de notation global

- *Complexity adjustment factor* is usually 1.0 or close
- Le facteur d'ajustement de la complexité est généralement de 1,0 ou proche

- Individual factors are normally 0, but can be positive or negative
- Les facteurs individuels sont normalement 0, mais peuvent être positifs ou négatifs

Default team grade (out of 100)
Note d'équipe par défaut (sur 100)

- (25%) Customer satisfaction / Satisfaction du client
  This is key / C'est la clé
- (20%) Professionalism and project management / Professionnalisme et gestion de projet
- (10%) Presentation/demo / Présentation/démo
- (25%) Design
- (20%) Communication

Grades for customer satisfaction worth 25% (1/2)

5% Problem solved?

- 0= not at all; 1=partly; 2=considerably; 3=mostly; 4=almost fully; 5= fully; 6=exceeds expectations for quality; 7=exceeds expectations for quality and functionality

5% Their perception you have been working hard

5% Reaching out and meeting

- Has the whole team met, or tried to meet with customers regularly?
- Obtaining requirements, testing prototypes

Getting grades for customer satisfaction (2/2)

5% Have you listened to them

- Did you respond to customer input, including explaining why some things were not done?
- Have you adjusted the project to meet their needs, as the requirements change?

5% Concretely delivered

- 4910: Do you have a great prototype?
- 4911 (middle) Do you have a minimum viable product?
- 4911 (near end) Is the product on market or in production, with knowledge transfer?

Grades for design worth 25%

- Have you made design decisions that allow for flexibility, scaling, security and maintainability?
- Do you have good data schema and APIs
- Have you used the right frameworks?
- Is code written well?
- Does it work properly?
- Is the UX good (easy to learn, speedy response, has needed features, gives good feedback, etc.)
- Is there evidence of enough tests to prove it works in all reasonable cases?

Grades for professionalism and project management worth 20%

- Have you followed agile process with sprints, testing, etc.
- Have you managed the balance among schedule (fixed), scope (carefully and quality (must keep high)?
- Has your team collaborated well?
- Have you made steady progress including getting going quickly?
- Have you dealt with problems well?
- Have you avoided or dealt with any ethical problems?
- Have you interacted with the customer well?
- Have you stuck to a schedule that allows it to be in production and maintained by others at the end of semester 2?

Grades for communication worth 20%

- Do you have a good record of requirements at a basic level?
- Have you recorded meeting minutes and design decisions (on the wiki)
- Can the prof and others understand your architecture quickly?
- Is your code well commented?
- Have you participated in class discussion?

Complexity adjustment factor (multiplied by group grade; default 1.0)

- Rewards hard work + ensures you are not trying to get an easy ride on an overly-simple project

- 1.1-1.2: Exceptional complexity in some of: architecture, algorithms, number of features, lines of code, tests, UX challenges, technical challenges
  - And good progress has been made
  - Judged relative to team size

Complexity adjustment factor continued

- 1: Project seems the right size for a team with each student working 12-15h/week
- 0.7-0.99: Small size; not sufficiently challenging; not developed far enough
- <0.7: Trivial or considerably incomplete

Individual adjustment factors (added; normally zero)

- A. Up to +10 points: Exceptional individual work as judged by clients, teammates or professor. No more than 50% of team members may receive such marks

- B. Up to +10 points from team members who have had to work hard to make up for other members that have lost points from C and D below

Individual adjustment factors

- C. up to -15 points. Student is > 15 days late delivering critical promised work to the team with insufficient justification.
  - Professor must be notified at 10-day point, and will impose penalty at 15-day point.

- D. up to -10 points. Quantity and quality of commits as assessed by the professor and TA are lower than expected of a 4th year student.

Individual adjustment factors

- E. -5 each. Missing/late individual deliverables to the prof (self-evaluation, URL with personal commits, absent from presentation, excess absence from class)

- F. -5. Failure to disclose ethical issues (e.g. conflict of interest, working full time)

Agile work

- Use a **Git repository**
  - Github (preferred), Gitlab, Bitbucket. Give the prof and TA access
- Deliver **tests** with commits where possible
- Use **issue tracking** for all user stories, features and bugs
- Use a **Wiki or text files** in repo for requirements, design.,
  - Put meeting minutes, progress logs here
  - Do not use non-repo files (e.g. Googhle docs)
- Use a **group communication** channel (Teams, or Slack)
- Set up **automated building** where possible

- **Deliver in increments** when commits are ready at intervals of no more than a month, starting in Week 5 at the latest
- Continuous integration
  - Each person commits their changes and the build runs

Legal Issues

- Academic Fraud
  - Using others work without acknowledgement
  - Misrepresenting results, participation
- IP
  - You own your work but can relinquish your ownership (companies may insist on this)
- Non-Disclosure Agreements
  - Sometimes needed – discuss with professor
- Paid-For Work
  - I do NOT recommend it. You must discuss with me.
  - All team members must be equal

Work schedule

- Schedule times each week to work with your team
  - Use the timeslots when you are not attending class
- After the pandemic, you can use project room on STE 2nd floor
  - I will need to sign a form to get you access

A real project driven through to completion (1)

- Your capstone should be **in production** by the time you are done
  - Being used by the customer for a month or two at least
  - Or on the market with several update cycles and some downloads
- …With a plan for ongoing maintenance by somebody

A real project driven through to completion (2)

- This means
  - Keep it small enough
  - Focus on high-value requirements
  - Ensure there is automated testing
  - Ensure it is
    - Maintainable
    - Installable
    - Flexible
    - Usable, etc.

A real project driven through to completion (3)

- You will lose marks at the end if
  - The project is ‘sort of done’ but will likely be abandoned
  - You only submit a functioning system at the very last minute without much chance for people to seriously use it and find new issues
- There can be exceptions for ‘proof of concept’ work, but this must be pre-approved by me and the customer

Avoid *blindly* doing what customers 'say' they want

- Many real requirements emerge 'in use' and through usability testing
- Requirements ALWAYS change
- Ask multiple potential users and experts (including the profs and TAs) and follow their advice
- Use your creativity and inventiveness to make your project better and more flexible

Beware of architectures/ frameworks that make response time slow

- The ‘modern web’ isn’t always the best
- Fast response time is one of the most important qualities

Best Practices Address Root Causes

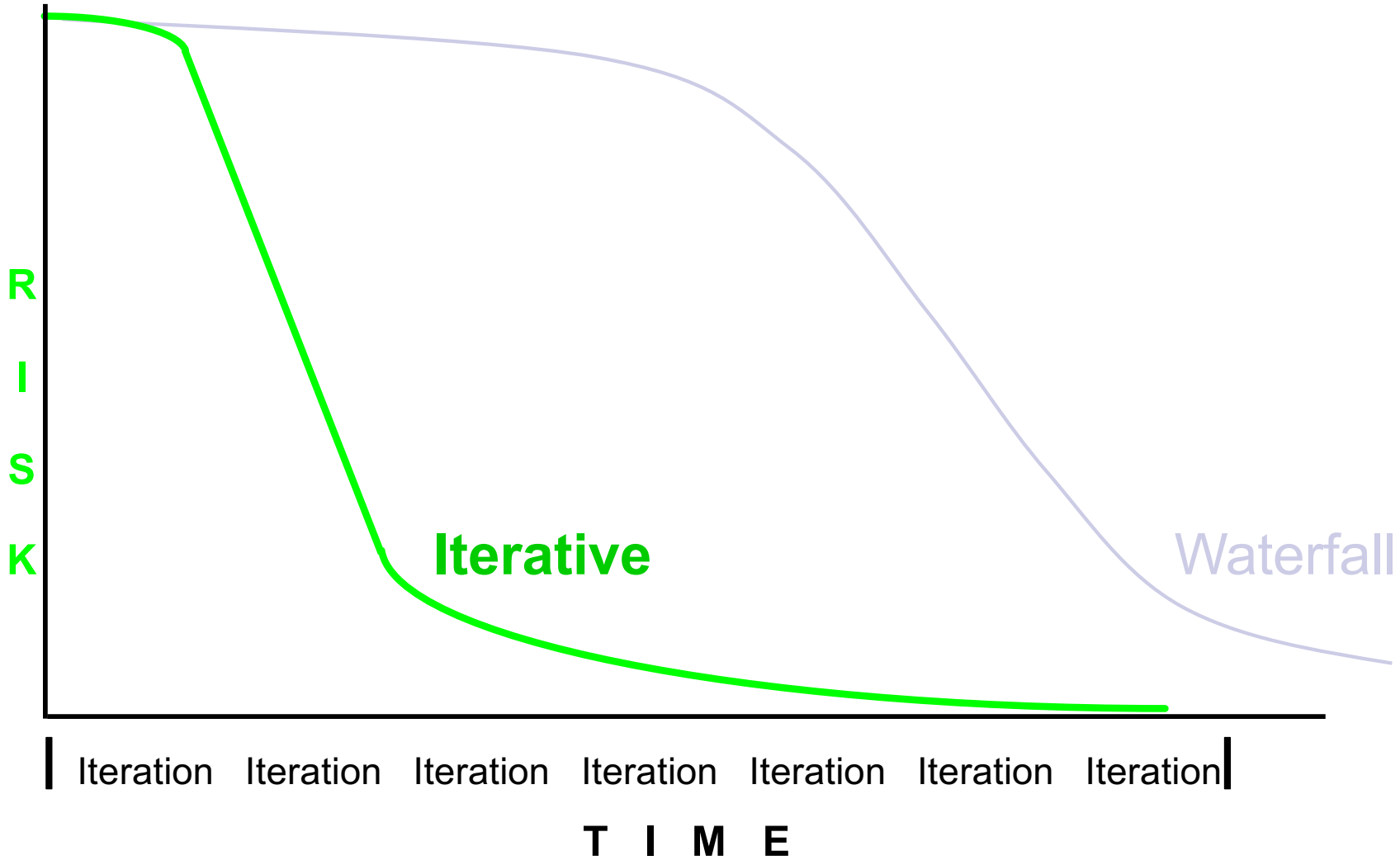
Root Causes

- Insufficient requirements
- Ambiguous communications
- Brittle architectures
- Overwhelming complexity
- Subjective assessment
- Undetected inconsistencies
- Poor testing
- Waterfall development
- Uncontrolled change
- Insufficient automation

Best Practices

- Develop iteratively using agile methods
- Manage requirements using agile methods
- Use component architectures and frameworks
- Model the software visually – consider Umple
- Test driven development
- Version control with pull requests and review
- Continuous integration

Iterative Development Accelerates Risk Reduction



Iterative Development Characteristics

- Critical risks are resolved before making large investments
- Initial iterations enable early user feedback
- Testing and integration are continuous
- Objective milestones provide short-term focus
- Progress is measured by assessing implementations
- Partial implementations can be deployed

Analysis & Iterative Development

- SCRUM (Ken Schwaber)
  - http://www.scrumalliance.org/learn_about_scrum
  - 2-4 week sprints (customer releasable), prioritized feature backlog
  - See separate slide deck
- Extreme Programming (Ken Beck)
  - 3 week iterations, tests and data created and agreed to by customer before coding begins

The Agile Manifesto

http://agilemanifesto.org/principles.html

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

Design in an Agile Environment

Some ‘agile’ proponents downplay design

But

- Design is not the same thing as documentation
- All the manifesto downplays is comprehensive documentation

How should design be expressed in an agile environment?

What is the minimum the *team* needs to properly structure the next sprint?

- Use cases/stories & test cases
- UI sketches / storyboards
- Class diagrams
- APIs
- Non-obvious Architecture Elements:
  - Components, layers, dependencies, patterns and mechanisms

Agile Design 2

All of the above for the current sprint only

- i.e. don't design much beyond the current sprint except the list of user stories (backlog)
- Update your design in each sprint
  - As you tackle new user stories
  - As you refactor

Avoid recording design elements that can be rapidly found by looking in the code or using tools

Agile Design 3

What information will *other software engineers* need when they try to modify your software?

- All of the above plus:
  - How is the code laid out?
    - Readme files in each directory
    - Headings at the top of each file
    - Possibly: Package diagram with textual descriptions
  - How to make anticipated types of changes
- Create a high-level ‘developers guide’ for everything that can’t be put on code

Agile Design 4

Beware of *maintaining* design elements that will have to be changed whenever code is changed

- Storyboards (put in ‘old’ folder after use?)
- Use cases (perhaps just keep a list, not details after they are implemented?)
- UML diagrams (can they be generated from code or can you use a tool that generates code from diagrams such as Umple)

Agile Design 5

As you create or maintain a document (i.e. a wiki page)

- Ask yourself if the effort of creating and maintaining it is worth it?
  - Will anyone actually read it (again)?
- Will the cost of deleting it or throwing out detail be more than the cost of not being able to find information?

Agile Release Management

Key: Releasing something functional to the user by the end of every sprint

Methods

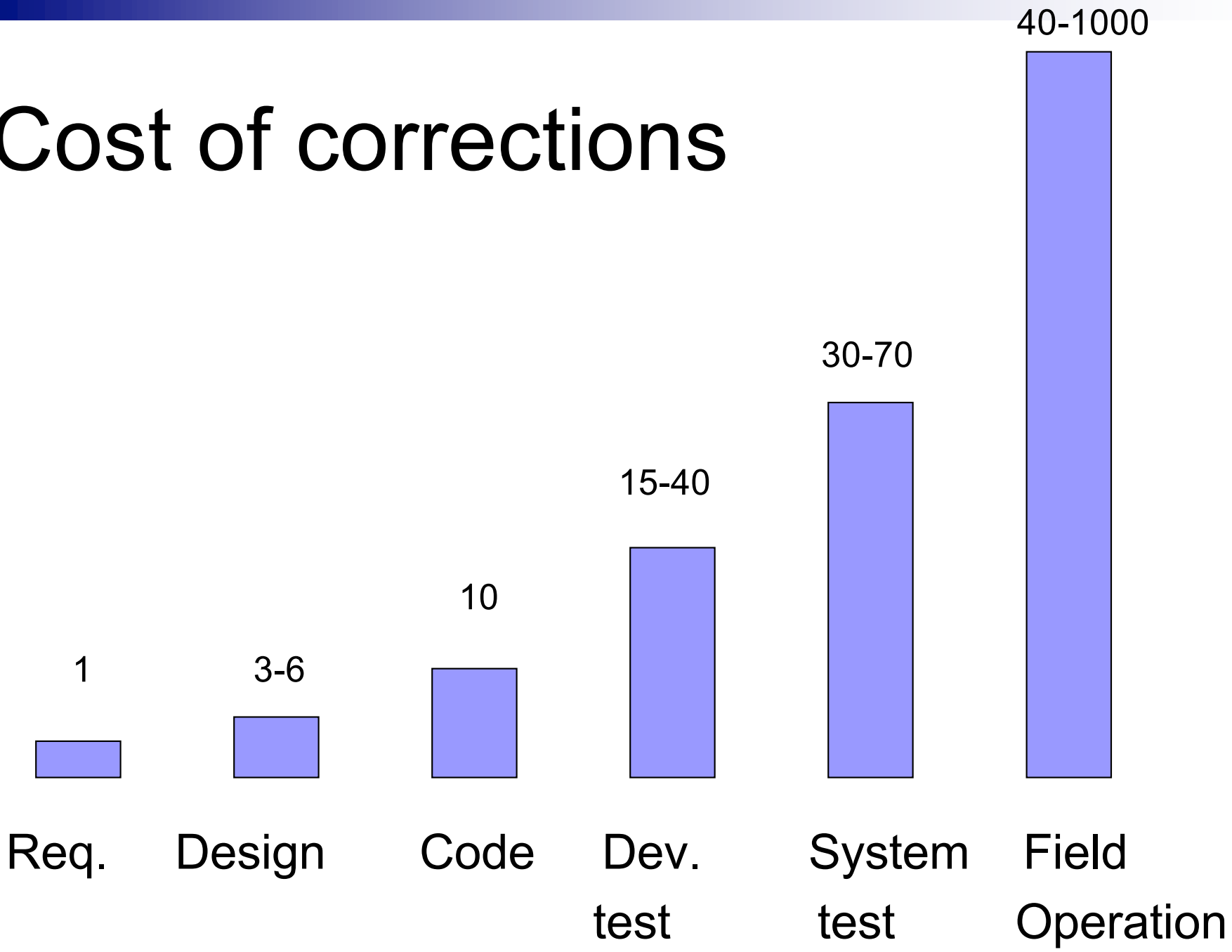
- Test-driven driven development
- Managing quality vs. scope vs. time
  - Drop features from the release to ensure quality requirements and deadline are met

Agile Release Management 2

Methods continued

- Continuous integration
  - Every submit to the repository triggers a build
- Issuetracking
  - With measurement of how well you are doing at reducing the backlog of bugs

Cost of corrections



Example Sources of Defects (1)

- Misinterpretation of customer/user real needs
  - Poor communication / lack of iteration/prototyping
- Unanticipated scenarios never accounted for
  - Changing operating system, dependency version or regulations
  - Feature interactions
  - Legal issues, licensing issues
- Not considering enough cases
  - Not enough use cases
  - Not enough types of data, etc
- Shortcuts to save time
  - Skipping tests, comments, requirements

Example Sources of Defects (2)

- Hacking or coding without design
  - Agile development still means being disciplined
- Excess complexity
  - Complex logic
  - Classes or methods with too many lines
- Not understanding code or data before making changes
- Weakness in user interface
- Bad database / bad class diagram
  - E.g. Not normalized

Tracking and continual improvement

- Have a goal to reduce the number of high priority defects
  - Track over time
  - Remove defects before adding features
- Data mine for common causes
  - Improve processes to reduce common causes
- Show graphs of accomplishments
  - Needed by mid-SEG4911