Översikt

- Implementation
  - Vad är en bra implementation och hur når vi dit?
- Quality Assurance
  - Hur säkerställer vi kvalitet?

Planering

- F1
  - Repetition Design
  - Responsible Programmer
  - TDD
- F2
  - Jan-Erik – Impl+QA översikt etc…
- **F3 (Idag)**
  - **Good Implementation - Clean Code**
  - **Quality Assurance i agil process**

Good implementation?

Bad implementation?

Good implementation - ilities

Completeness
Correctness
Traceability
Performance
Readability
Maintainability

Good implementation

Complete
Correct
Traceable
Performant
Readable
Maintainable

Good enough!

Complete *enough*
Correct *enough*
Traceable *enough*
Performant *enough*
Fast *enough*
Readable *enough*
Maintainable *enough*

→ **Customer Value (Cheap enough)**

Complete and Correct

Completeness – gör **allt** i kravspec
Correctness – löst enligt kravspec

Några problem med det?

Det beror på! (ständiga konsultsvaret)

Levande specifikation → Ja!

Up-Front-Specification → Nej!

Complete and Correct

Målet med BDD/ATDD/Specification By Example

En kontinuerligt exekverande kravspec!

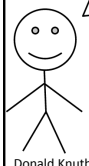
Traceability

Process och verktyg!

T.ex. Wiki + Jira + Subversion

Performance

We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%



Donald Knuth

Det innebär dock inte att prestanda kan testas sent!

Readability
Maintainability

Lätt att
- Förstå
- Ändra
- Använda
- Testa
- Integrera

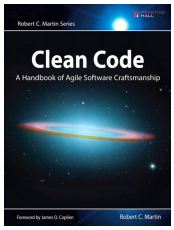
→ Clean!

Good implementation

Complete
Correct
Traceable
Performant
Readable
Maintainable

Clean Code

What is it?
Some excerpts from:



Clean Code

Be proud!
Cleverness in code is never a thing to be proud of.

Four principles of simple design by Kent Beck:
(1) Tests pass
(2) Exposed intent (good names)
(3) DRY
(4) Small.

Clean Code – Hur?

Ubiquituous Language
Design principles
Coding guidelines
Continuous Design – **refactor**!
TDD!
Patterns
Pair programming

Coding guidelines

Size – must fit on screen!
Naming
Indentation
Commenting
File naming
Etc…

Size

Method must fit on screen

Class must fit on screen

Refactor till you drop!

Method not more than 5 lines

Comments

Comments is a smell

Be proud!

Comments are never a thing to be proud of.

Comments does not compile!

Never

- Repeat Code
- Explain line
- Keep TODO, FIXME, etc

OK

- External refs
- Summary
- Code intent

Refactor

- Bad naming
- Duplication
- Long method/Class
- Bad Design
  - Not SRP
  - Not DRY
  - etc…
- Slow
- etc…

Rename
Extract method
Extract class
Move
Substitute Algorithm
…

Boy Scout Rule

Broken Windows

Teknisk skuld

Metafor skapad av Ward Cunningham (Wiki, XP, Fit, design patterns, …)

Två lösningar
1. Snabb, men kommer att göra förändring svårare längre fram
2. Tar längre tid, men renare design och tydligare kod.

Lösning 1 skapar en skuld, som måste betalas av.
Om man inte betalar av så blir räntan till slut för hög.

Teknisk skuld

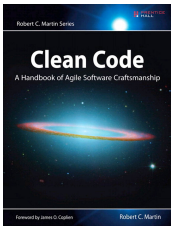
Saknade tester är INTE teknisk skuld i ursprungsmeningen! Det är bara dumt, inte skuld!

Code Quiz!

Code Quiz!

Hittas på https://github.com/jonananas/teaching

Läs!



Quality Assurance

- Syfte: Produce High Quality Software
  - Enligt specifikation
  - Redo att användas

Quality = Good impl?

Complete
Correct
Traceable
Performant
Readable
Maintainable

Quality Assurance

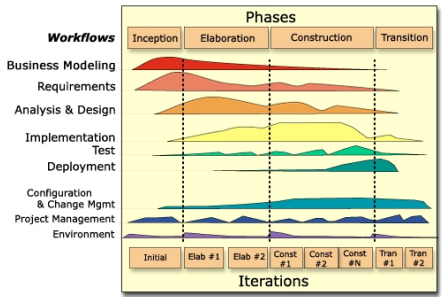
- Syfte: Produce High Quality Software
  - Enligt specifikation
  - Redo att användas

Metod	Risk
Up-Front specification	Hög
BDD	Låg
Late Deploy	Hög
Continuous Deployment	Låg

Quality Assurance

- Traditionellt syfte:
  - Hitta defekter så de kan åtgärdas
  - Bedöma kvalitet, törs vi release:a?
    - (Låta testavdelningen ta smällen om det skiter sig)
- Syfte i en agil utvecklingsprocess
  - Kontrollera att processen funkar, hittar vi fel (tex buggar) så måste processen förändras!

RUP – Rational Unified Process



Quality Assurance

- Traditionell approach:
  - Punktinsatta manuella verifieringar av artefakter (dokument, kod, etc.)
  - Stor manuell insats innan vi gör release.
  - Manuell acceptanstest när produkten är klar.
- Agil approach:
  - Kontinuerligt fokus på kvalitet:
    - Acceptanstester specificeras i förväg och automatiseras.
    - TDD
    - Frekventa releaser med Demo
    - Manuell exploratory testing

Quality Assurance

- ”The best way to obtain quality in a product is to put it there in the first place”, pp 200 i boken

Principer i Lean Software Development

1. Minimera slöseri (Eliminate Waste)
2. **Bygg in kvalitet** (Build quality in)
3. Skapa Kunskap (Create knowledge)
4. Ta beslut så sent som möjligt (Defer commitment)
5. Leverera snabbt (Deliver fast)
6. Respektera människor (Respect people)
7. Optimera helheten (Optimize the whole)

Lean: Slöserier

- Tid mellan hittad defekt och åtgärd

Lean: Bygg in kvalitet

- En bugg som upptäcks i testfasen tyder på fel i utvecklingsprocessen
  - Att testa för att hitta buggar är slöseri
  - Att testa för att undvika buggar är ett måste
- Kod som är svår att underhålla är av låg kvalité → utvecklingsprocessen
- Kod som är svår att testa är för komplex → utvecklingsprocessen

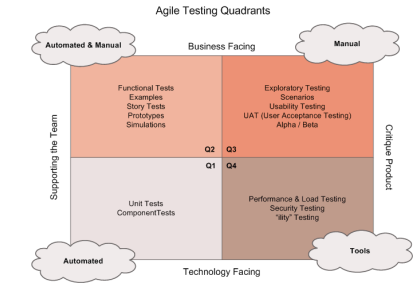
QA Techniques

- **Testing**
- Inspection and Review
- Formal Methods
- Static Analysis

Testing

- Acceptance Testing
- UI Testing
- Conformance Testing
- Configuration Testing
- Performance Testing
- Stress Testing

Testing



http://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/

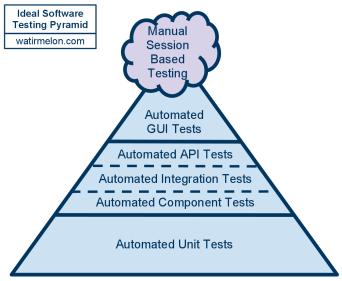
Test Automation Pyramid

http://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid

Mike Cohn

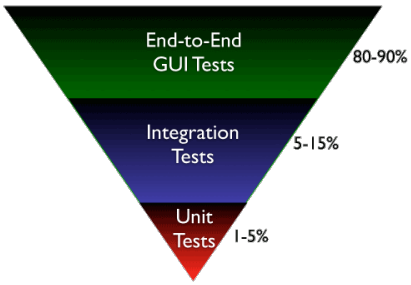
http://jamescrisp.org/2011/05/30/automated-testing-and-the-test-pyramid/

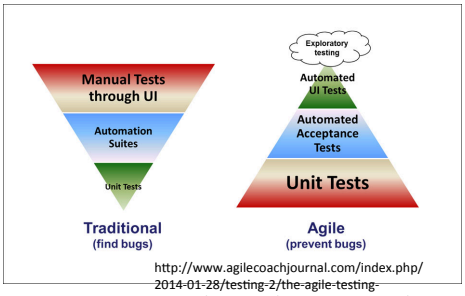
Test Automation Pyramid



http://watirmelon.com/tag/testing-pyramid/

Ett annat sätt…





http://www.agilecoachjournal.com/index.php/2014-01-28/testing-2/the-agile-testing-pyramid/attachment/agiletestingpyramid-3/

Imorgon 15:15 MC413

Markus Örebrand, OmegaPoint

Små misstag, stora säkerhetshål
Utgår från några vanligt förekommande exempel på osäker programmering, knyter an till OWASP Top 10 och spekulerar i möjliga exploits.