Software Engineering
Introduction

Department of Computer Science, Ben-Gurion University
Fall 2019

Based on Presentation by
Nurit Gal-oz, Department of Computer Science Ben-Gurion university
Mira Balaban Department of Computer Science Ben-Gurion university
Igor Potapov: http://www.csc.liv.ac.uk/~igor/COMP201
Ian Sommerville: http://www.comp.lancs.ac.uk/computing/resources/IanS/SE7/Presentations/index.html
Alexander Borgida: http://www.cs.rutgers.edu/~borgida/431/
מנولة

Foundations of Software Engineering

https://www.cs.bgu.ac.il/~fsen191/Main
What is software?

- **Computer programs** and associated documentation

- **Software products** may be developed for a particular customer or may be developed for a general market

- **Software products** may be
  - **Generic** - developed to be sold to a range of different customers
  - **Bespoke** (custom) - developed for a single customer according to their specification
What is Engineering?

- Application of a systematic, disciplined, quantifiable approach to
  - structures,
  - machines,
  - products,
  - systems,
  - processes.

What is Software Engineering?

- Software Engineering is that form of engineering that applies
  - a systematic **disciplined quantifiable** approach
  - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines as necessary
  - and sometimes just plain invention

to

creating, developing, operating and maintaining cost-effective, reliably correct, high-quality solutions to software problems

[Definition by Berry 92]

**SE** requires

- the **identification of a problem,**
- a **computer** to carry execute a software product, and
- a **user environment** (composed of people, tools, methodologies, etc.)
Software Engineering - Motivation

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed
The task of software engineers?

**Software engineers** should

- adopt a *systematic* and *organised* approach to their work
- use appropriate tools and techniques depending on
  - the problem to be solved,
  - the development constraints and
  - the resources available
Origin of Software Engineering?

- **Software engineering** deals with the development of high-quality software systems.

- **Software engineering** is *abstract* – no physical limitations:
  - → leads to **unlimited complexity**.

- **Software engineering** – coined in 1968: within discussion of **software crisis**.
  - The NATO conference, 1968 in Garmisch, Germany
  - In response to the perception that computer programming had not kept up with advances in computer hardware.
THE SOFTWARE CRISIS
HELP WANTED!

One of the most significant developments in the computer industry during the 1960s was the perceived shortage of skilled “computer people.”

The programmer personnel crisis is the first of the many “software crises” that were proclaimed over the next several decades.

What is the “software crisis”?

- Major projects are meaningfully late.
- Software costs more than predicted.
- Software is unreliable.
- Software is difficult to maintain.
- Poor performance.
- While hardware costs were decreasing, software costs were rising.

→ requires techniques to control the complexity of large software systems.
Standish Group’s CHAOS 2009 Report

- **Successful** - projects delivered on time, on budget, with required features and functions.
- **Challenged** - late, over budget, and/or with less than the required features and functions.
- **Failed** - cancelled prior to completion or delivered and never used.
Standish Group’s CHAOS 2014 Report

**Resolution of Large Software Projects**
- Challenged: 52%
- Failed: 42%
- Successful: 6%

**Causes of Downtime**
- Application Bug or Error: 42%
- Operator Error: 5%
- Hardware Failure: 6%
- Infrastructure Software Failure: 17%
- Environment & Other: 18%
- Network: 6%
- Viruses: Hacker, etc.: 8%

*The above chart shows application bugs to be the leading cause of downtime from Standish Group’s 20 years of primary research.*
NASA’s Mars Climate Orbiter Project

• In 1999 a Mars space probe from NASA was ‘lost’

• Two teams involved in the space probe development were using different systems of measurement
  • Lockheed Martin engineering team used English units of measurement while the NASAs team used the more conventional metric system for a key spacecraft operation

• Reasons:
  • There had been inadequate consideration of the entire mission and its post-launch operation as a total system
  • Communications and training within the project had been inconsistent
  • There was no complete end-to-end verification of navigation software and related computer models
HealthCare.gov

Now
HealthCare.gov

Two Years ago

FAILURE TO LAUNCH

HealthCare.gov

Find health coverage that works for you

Get quality coverage at a price you can afford. Open enrollment in the Health Insurance Marketplace continues until March 31, 2014.

APPLY ONLINE  APPLY BY PHONE

SEE PLANS AND PRICES IN YOUR AREA  SEE PLANS NOW
מה קורה איתנו

נייטל ופריקים חכונה, ד”ר سمיר تومر, המשכל, 2014, מסקיע

19
אירוטים מביכים וオスפים הקשורים בחקוכה

יוני 1996 - תליית "אריאן" מתפוצצת 40 שיני לתח לישון
- הסיבה: המרת עָּלֶג מֶּסֶרֶר בָּה מַשְׁתַּנְגָּה מסגוג real (64 ביט) למסגוג מסגוג integer (16 ביט)
- http://www.around.com/ariane.html

פברואר 1991 – טיל פיתוח אמריקאי מתפוצץ טייל סקואד ייראק, הורגמ 28 הרוגים
- הסיבה::setext:

ספטמבר 1999 – תליית מתרק לאקלים האמיון אובדת חלול
- הסיבה: שיווק מעורב בייתווד משקל

נובמבר 1983 – ההרשה בנקובר קורס אっちי שלמד צוות מ-1000-520 זוכות פוטנציאליות
- הסיבה: כלוק "קיוו" במקומ "נייגו" בسفرרה הרבעית אっちי, הפקודה והשגרה והסכם והשגרה של המדר
- http://www5.in.tum.de/~huckle/Vancouver.pdf

1995 – רשת סלקום (בראשית דרחה) מעבלי מתפרעים תקשות optimizations דרדר
- הסיבה: געולים י더 בדומן הקמיה שלית במקשוריים מסוג "אלפב" של מטאורולוגיה
- http://catless.ncl.ac.uk/Risks/17.26.html#subj1

מקורות נוספים:
- http://www5.in.tum.de/~huckle/bugse.html
- http://catless.ncl.ac.uk/Risks

מקון: רופא, עדין חמור, ש𝕄יהט ב荙נסטז מוןנה 2014, הספרי
Why Software FAILS

HUNDREDS OF BUGS

TROUBLESHOOTING

IT DOESN'T WORK

YOU SAID IT WILL... WHY NOT

WORK ON ALL BROWSERS

YOU ARE NOT LISTENING

THIS IS NOT WHAT WE PAID FOR

WHAT IS WRONG WITH YOUR DEVELOPER

I HATE THIS

OVER THE BUDGET

TOO MUCH MONEY

IT DOESN'T WORK ON MY LAPTOP

SPINS

NO ONE IS COMING TO MY SITE

WE ARE NOT ROAD

THIS IS CONFUSING

WHY CAN'T THIS WORK

WE CAN'T CODE IT THAT WAY
- Inaccurate estimates of needed resources
- Badly defined system requirements
- Poor reporting of the project’s status
- Unmanaged risks
- Poor communication among customers, developers, and users
- Use of immature technology
- Inability to handle the project’s complexity
- Sloppy development practices
- Poor project management
- Stakeholder politics
- Commercial pressures

Programming in the Small and Programming in the Large

Programming in the small

One programmer
understands everything from top to bottom.
Main problem is the
development of algorithms

Programming in the large

System is developed by a
large teams of programmers
Main problems are management of details and
communication between software subsystems
Defines software system in terms of components and interactions among those components
Why Software Engineering is Needed?

- Software development is hard!
- Important to distinguish
  - “easy” systems (one developer, one user, experimental use only) from
  - “hard” systems (multiple developers, multiple users, products).
- Experience with “easy” systems is misleading:
  - One person techniques do not scale up.

- Analogy with bridge building:
  - Over a stream = easy, one person job.
  - Over a big River … ? (the techniques do not scale)
Why is SE hard?

• **real world problems:**
  - Requirements are not clear and “acceptability” is defined by user satisfaction
  - For example: “software to help people control nuclear reactors”

• **wicked problems:**
  - define and solve concurrently
  - no unique definition or solution
  - new problem, not previously encountered
  - many stakeholders, with different goals
Success Factors

- **Type of Development**
  - Methodologies, Process, Staff Skill, Tools and delivery
  - Highly skilled **executive sponsor**
  - Highly **engaged users** who thoroughly know their subject Matter
  - Focus on **high value items** instead of providing on low value items just for the sake of political appeasement
  - **Quick Decision making** and **cooperative peers** who can ensure working together as team to achieve a common goal
**BUT:** The Software Werewolf

- Software is like a werewolf—it looks normal until the moon comes out and it turns into a monster
  - Missed deadlines
  - Blown budgets
  - Buggy software
- We want the *silver bullet* to kill the monster
  - something to make software costs drop as rapidly as computer hardware costs do.

*Frederick P. Brooks: No Silver Bullet: Essence and Accidents of Software Engineering*
There Is No Silver Bullet!

Frederick P. Brooks: “As we look to the horizon of a decade hence, we see no silver bullet.

There is no single development, in either essence, technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity”

*Frederick P. Brooks, No Silver Bullet: Essence and Accidents of Software Engineering*
Essential Difficulties of SE: [Brooks]

- **COMPLEXITY:**
  - software is more complex for its size than any other human construct;
  - no two parts are alike (vs. car, microchips,…);
  - science advances by simplifying, while software cannot ignore/simplify details of real world

- **CONFORMITY:**
  - among {hardware, software, people, organizations} it is software which is chosen to *bend or adapt* because it is more malleable, last to arrive on the scene, usually only one developed on site,…

- **CHANGEABILITY:**
  - once delivered, most engineered products (hardware, cars, buildings) are rarely changed because the cost to change would be a large fraction of the cost to make. The (unfortunate) *perception* is that software is cheap to change. And pressure to change comes from successful use, and aging hardware platform.

- **INVISIBILITY:**
  - since it has no physical reality, software is not properly visualizable with diagrams, etc. in the way in which houses, circuits, etc are.

*Frederick P. Brooks: No Silver Bullaet: Essence and Accidents of Software Engineering*
Complexity

- Software is more complex for their size than an other human construction.
- Many problems of developing software products derive from this essential complexity and its nonlinear increases with size.
  - [http://www.informationisbeautiful.net/visualizations/million-lines-of-code/](http://www.informationisbeautiful.net/visualizations/million-lines-of-code/)
- The increase in size makes communication among team members hard.
- Problems are: product flaws, cost overruns, schedule delays.
- Are not only technical problems, management problems too.
Complexity

What is the maximum number of digits for the year?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$

$\approx 10^{255}$

\[
\begin{align*}
&= 12 \times 2 \times 60 \times 60 \times 24 \times \\
&12 \times 31 \times 100 = 77,137,920,000 \\
&= 360 \times 2 \times 60 \times 60 = 2,592,000 \\
&= 360 \times 2 \times 60 \times 60 = 2,592,000
\end{align*}
\]

$\approx 10^{21}$

Source: Professor Shmuel Tomer, Technion, Winter 2014, Hebrew

What is the maximum number of digits for the month?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$

What is the maximum number of digits for the day?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$

What is the maximum number of digits for the hour?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$

What is the maximum number of digits for the minute?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$

What is the maximum number of digits for the second?

- $30^{18}$
- $30^{40}$
- $30^{50}$
- $30^{40}$
Conformity

- Conformity comes from the fact that all new software must conform to the way things were done in the past, because it is hard to change everyone.
- Main reason software must conform is that it is new on the scene.
Changeability

- Being able to change something after it has been constructed is a new concept.
- Software must be able to change, and the complexity of the systems becomes harder to follow.
- All successful software gets changed.
- Successful software also survives beyond the normal life of the machine vehicle for which it is first written for.
Invisibility

- Software has no 3-D way on which it can be laid out.
- Software does not have one map or graph, it will have multiple maps and graphs.
- Communication becomes hard because each person my see it a different way.
The software product is intangible

$70 FOR THIS MEAL? Hmm... I GUESS THAT SOUNDS FAIR.

WHAT?!!
$2 FOR A PIECE OF SOFTWARE? ARE THEY CRAZY?!?!
What will (not) make a big difference:
[Brooks1987]

• Minor
  • Ada or Java or C# or Python or …
  • Object-Oriented Programming
  • Artificial Intelligence
  • Automatic Programming
  • Graphical programming
  • Program verification
  • Environments and tools
  • Workstations

• Major
  • Buy vs build
  • Requirements refinement and prototyping
  • Incremental development
  • Great designers (Unix, Pascal, Smalltalk vs Cobol, PL/I, Ada, MS-DOS)
No Silver Bullet Reloaded Retrospective
OOPSLA Panel 2007
Bottom line (by Berry)

*I no longer get excited over any new language, development model, method, tool, or environment that is supposed to improve programming…*

…*The most important work is addressing requirements, changes, and the psychology and sociology of programming*
משולש האילוצים

בגרע שมงคลים שניים
אי אפשר לרצות את כל שלושת האילוצים

• בעל עניי חיים לוליטה
• אל שננים חשבים יתור

Software Project Management - Fall 2016 SCE Dr. Azzam Maraee
SE: Processes, Models, and Tools

- **Processes:**
  - Systematic ways of organizing teams and tasks so that there is a clear, traceable path from customer requirements to the final product (e.g. Waterfall, Prototyping, Spiral, etc.)
  - Processes help organize and co-ordinate teams, prepare documentation, reduce bugs, manage risk, increase productivity, etc.

- **Models:**
  - Well-defined formal or informal languages and techniques for organizing and communicating arguments and decisions about software. e.g:
    - specification languages (Z, etc)
    - design models (UML, etc)
  - Models help stakeholders communicate: customers with developers, designers and developers, developers and testers etc.
  - If they are formal, they also can help support automation

- **Tools:**
  - Programs which automate or otherwise support software development tasks: e.g., Eclipse, Make, CVS, etc.
  - Tools increase productivity, quality and can reduce costs
What is a software process?

• **A structured set of activities required to develop a software system**

• Generic activities in all software processes are:
  • **Specification** - what the system should do and its development constraints.
  • **Development** - production of the software system
  • **Validation** - checking that the software is what the customer wants.
  • **Evolution** - changing the software in response to changing demands.
Software Engineering Activities

1. **Problem statement**
   - needs analysis
   - requirements specification: functional, non-functional

2. **Design**
   - architectural
   - detailed
   - (communication, database)

3. **Implementation**
   - coding
   - testing:
   - module
   - integration
   - documentation

4. **Maintenance**
   - corrective
   - adaptive
   - enhancement
What is a software process model?

- **An abstract representation of a process**
  - It presents a description of a process from some particular perspective

- **Examples of process perspectives:**
  - **Workflow perspective** represents inputs, outputs and dependencies.
  - **Data-flow perspective** represents data transformation activities.
  - **Role/action perspective** represents the roles/activities of the people involved in software process.

- **Generic process models:**
  - **Waterfall**
    - Separate and distinct phases of specification and development
  - **Evolutionary development**
    - Specification, development and testing are interleaved
  - **Iterative Development**
  - **Formal transformation**
    - A mathematical system model is formally transformed to an implementation
  - **Integration from reusable components**
    - The system is assembled from existing components

In practice applied for the design and implementation phase of software development.
The attributes of good software?

- The software should deliver the required functionality and performance to the user and should be **maintainable**, **dependable**, **efficient** and **usable**

- **Maintainability**
  - Software must evolve to meet changing needs.

- **Dependability**
  - Software must be trustworthy.

- **Efficiency**
  - Software should not make wasteful use of system resources.

- **Usability**
  - Software must be usable by the users for which it was designed.
הנדסת תוכנה – נוף הידע - SWEBOK

mekor: פרופ' עמיר חומר, שימור ביהנמסה תוכנה 2014, מoccan

www.swebok.org
What is the difference between software engineering and system engineering?

- **Software engineering** is part of **System engineering**
- **System engineering** is concerned with all aspects of computer-based systems development including
  - hardware,
  - software and
  - process engineering
- **System engineers** are involved in
  - system specification,
  - architectural design,
  - integration and deployment
What are the key challenges facing software engineering?

Software engineering in the 21st century faces three key challenges:

- **Heterogeneity**
  - Systems are distributed and include a mix of hardware and software.
  - Legacy systems (old valuable systems) must be maintained, updated and integrated into new systems.

- **Delivery**
  - There is an increasing pressure for faster delivery of software.

- **Trust**
  - Software must be trusted by its naive users.
Professional and ethical responsibility (1)

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law.
Issues of professional responsibility (2)

- **Confidentiality**
  - Engineers should normally respect the **confidentiality** of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.

- **Competence**
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is out with their competence.
Issues of professional responsibility (3)

- **Intellectual property rights**
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

- **Computer misuse**
  - Software engineers should not use their technical skills to misuse other people’s computers. Computer misuse ranges from relatively trivial (game playing on an employer’s machine, say) to extremely serious (dissemination of viruses).
ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.
Software Engineering Code of Ethics and Professional Practice


- Short Version
- Full Version

Software Engineering Code of Ethics and Professional Practice (Short Version)

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction: the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC - Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software
Code of ethics - principles

1. PUBLIC
   Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER
   Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT
   Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT**
   Software engineers shall maintain integrity and independence in their professional judgment.

5. **MANAGEMENT**
   Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. **PROFESSION**
   Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
Code of ethics - principles

7. COLLEAGUES
    Software engineers shall be fair to and supportive of their colleagues.

8. SELF
    Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.
Ethical dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.
Summary: Key points (1)

- **Software engineering** is an engineering discipline that is concerned with all aspects of software production.

- **Software products** consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.

- The **software process** consists of activities that are involved in developing software products. Basic activities are software specification, development, validation and evolution.

- **Methods** are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.
Summary: Key points (2)

- **CASE tools** are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.

- **Software engineers** have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.

- **Professional societies** publish codes of conduct which set out the standards of behaviour expected of their members.
The Software Process

- A **structured set of activities** required to develop a **software system**
  - Specification
  - Analysis, design and implementation.
  - Validation
  - Evolution

- **A software process model** is an abstract representation of a **process**
  - A **general approach** for organizing a project into activities;
  - It presents a description of a process from some particular perspective
  - An aid to thinking, not a rigid prescription of the way to do things
Waterfall Model

- Inflexible partitioning of the project into distinct stages
- Difficult to respond to changing customer requirements
“... the implementation described above is risky and invites failure.”

W. W. Royce, 1970
Waterfall

- Successful: 29%
- Challenged: 57%
- Failed: 14%

Agile

- Successful: 42%
- Challenged: 49%
- Failed: 9%

Incremental Development

- **Incremental** development is a staging and scheduling strategy in which various parts of the system are developed at different times or rates, and integrated as they are completed.
- The alternative strategy to incremental development is to develop the entire system with a “big bang” integration at the end.
Incremental Development – Version I

- Incremental Development
- Construction
- Maintenance
- Operational
- Deployment
- Application
- Testing
- System Design
- Integration
- Documentation
- Deployment (build) 1, 2, ..., n
- Deployment functionality
- Finalization

This diagram illustrates the incremental development process, focusing on the systematic construction, maintenance, and operational phases with specific emphasis on testing, integration, and documentation. Each step is crucial in ensuring the application is robust and ready for deployment.
Incremental Development – Version II

Diagram showing the process of incremental development with multiple versions.
Iterative Development

- **Iterative** development is a rework scheduling strategy in which time is set aside to revise and improve parts of the system.
- The alternative strategy to iterative development is to plan to get everything right the first time.
 Iterative Development

• There are two particular, specialized rework strategies:
  • Develop the system as well as possible, in the thinking that if it is done sufficiently well, the changes will be relatively minor and can be incorporated quickly

  • Develop the least amount possible before sending out for evaluation, in the thinking that less work will be wasted when the new information arrives.
    • Mona Liza example
Incremental means adding, iterative means reworking  (by Alistair Cockburn)

- **Incremental development** is a staging and scheduling strategy in which the various parts of the system are developed at different times or rates and integrated as they are completed. The alternative is to develop the entire system with a big bang integration at the end.

- **Iterative development** is a rework scheduling strategy in which time is set aside to revise and improve parts of the system. The alternative development is to get it right the first time (or at least declare that it is right!).

<table>
<thead>
<tr>
<th>Iterate</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>fundamentally means “change”</td>
<td>fundamentally means “add onto”</td>
</tr>
<tr>
<td>repeating the process on the same section of work</td>
<td>repeating the process on a new section of work.</td>
</tr>
<tr>
<td>repeat the process (design, implement, evaluate),</td>
<td>repeat the process (design, implement, evaluate),</td>
</tr>
</tbody>
</table>

Spiral model (Boehm 88)

B. Boehm, 1988

1. Planning
   - Characteristics of the solution
   - Requirements
   - Project risks/assumptions
   - Velocity of developments

2. Risk Analysis
   - Evaluation of the solution
   - Evaluation of the solution
   - Evaluation of the solution
   - Evaluation of the solution

3. Engineering
   - Identification of the solution
   - Identification of the solution
   - Identification of the solution

4. Evaluation
   - Evaluation of the solution
   - Evaluation of the solution
   - Evaluation of the solution
   - Evaluation of the solution
Agile
What is an Agile method?

- Focus on the code rather than the design.
- Based on an iterative approach to software development.
- Intended to deliver working software quickly.
- Evolve quickly to meet changing requirements.
- There are claims that agile methods are probably best suited to small/medium-sized business systems or PC products.
- More active customer involvement needed
- Considered People-based rather than Plan-based
- Several agile methods
  - Extreme Programming (XP) most popular
- No single definition - Agile Manifesto (2001) closest to a definition
  - Set of principles
  - Developed by Agile Alliance
The players in the game are “People”

Weak on:
- Consistency
- Discipline
- Following instructions

Strong on:
- Communicating
- Looking around
- Copy / modifying

Motivated by:
- Pride in work
- Pride in contributing
- Pride in accomplishment
## Summary of Principles of agile methods

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer involvement</td>
<td>The customer should be closely involved throughout the development process. Their role is to provide and prioritise new system requirements and to evaluate the iterations of the system.</td>
</tr>
<tr>
<td>Incremental delivery</td>
<td>The software is developed in increments with the customer specifying the requirements to be included in each increment.</td>
</tr>
<tr>
<td>People not process</td>
<td>The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.</td>
</tr>
<tr>
<td>Embrace change</td>
<td>Expect the system requirements to change and design the system so that it can accommodate these changes.</td>
</tr>
<tr>
<td>Maintain simplicity</td>
<td>Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.</td>
</tr>
</tbody>
</table>
What are the Agile Methodologies?

eXtreme Programming has received the most attention, but here is a list:

- XP
- SCRUM
- DSDM
- The Crystal Family
- ASD
- FDD
- dX (agile RUP)
- Open Source
- Agile Modeling
- Pragmatic Programming
eXtreme Programming

• Development and delivery of very small increments of functionality.

• Relies on constant code improvement, user involvement in the development team and pair wise programming.

• Emphasizes Test Driven Development (TDD) as part of the small development iterations.
Claimed Problems with agile methods

- It can be difficult to **keep the interest of customers** who are involved in the process.
- **Team members may be unsuited** to the intense involvement that characterizes agile methods.
- **Prioritising changes can be difficult** where there are multiple stakeholders.
- **Maintaining simplicity** requires extra work.
- **Contracts may be a problem** as with other approaches to iterative development.
Problems with incremental development (from a waterfall eye...)

- **Management problems**
  - Progress can be hard to judge and problems hard to find because there is no documentation to demonstrate what has been done.

- **Contractual problems**
  - The normal contract may include a specification; without a specification, different forms of contract have to be used.

- **Validation problems**
  - Without a specification, what is the system being tested against?

- **Maintenance problems**
  - Continual change tends to corrupt software structure making it more expensive to change and evolve to meet new requirements.
Occasionally, a researcher gets an idea for a new approach that may seem ridiculous to others, who discount it. However, in any area of active research, it is dangerous to discount any approach that has a reason that it might work and that has not been specifically disproved.

No one can know ahead of time whether or not the approach will work. If we discount an approach that would have worked, we are cutting ourselves off from a benefit. Ultimately, we cannot discount anything that might yield a solution, anything reasonable that has not been demonstrated not to have any impact on the process or the software.

Academic Legitimacy of the Software Engineering Discipline (Daniel M. Berry)