CHAPTER 12 – Conclusion

Overview
• 1. Introduction
• 2. Project Management
• 3. Use Cases
• 4. Domain Modeling
• 5. Testing
• 6. Design by Contract
• 7. Formal Specification
• 8. Software Architecture
• 9. Quality Control
• 10. Software Metrics
• 11. Refactoring

Articles
• The Quest for the Silver Bullet
• The Case of the Killer Robot

Professional Ethics
• Cases

The future: Software Engineering Tools
Software Process & Process

- **Software Process:**
  - Requirements Collection + Analysis + Design + Implementation + Testing + Maintenance + Quality Assurance

- **Software Product:**
  - Requirements Specification (= functional & non-functional) + System (= executable + code + documentation)
Evaluation Criteria

For large scale projects, a *feedback loop* is necessary ...
⇒ Mix of Iterative and Incremental Development

2 evaluation criteria to assess techniques applied during process

**Correctness**
- Are we building the right product?
- Are we building the product right?

**Traceability**
- Can we deduce which product components will be affected by changes?
Overview

Testing
Testing

Maintenance
Refactoring

Implementation
Design by Contract

Quality Assurance
Project Management
Quality Control
Software Metrics

Design
Software Architecture
Formal Specifications

Requirements Collection
Use Cases

Analysis
Domain Modeling
Project Management

- Plan the work and work the plan
- PERT and Gantt charts with various options
- Critical path analysis and monitoring

Are we building the system right?
- Deliver what’s required on time within budget
- Calculate risk to the schedule via optimistic and pessimistic estimates
- Monitor the critical path to detect delays early
- Plan to re-plan to meet the deadline

Traceability? Project Plan ↔ Requirements & System
- The purpose of a plan is to detect deviations as soon as possible
- Small tasks + Milestones verifiable by customer
Use Cases

Use Cases
• Specify expected system behavior as a set of generic scenarios

Are we building the system right?
• Well specified scenarios help to verify system against requirements

Are we building the right system?
• NO: validation by means of CRC Cards and role playing.

Traceability? Requirements ⇔ System
• Via proper naming conventions

Traceability? Requirements ⇔ Project Plan
• Use cases form good milestones
Domain Modeling

CRC Cards
- Analyse system as a set of classes
  - ... each of them having a few responsibilities
  - ... and collaborating with other classes to fulfill these responsibilities

Feature Model
- a set of reusable and configurable requirements for specifying system families (a.k.a. product line)

Are we building the system right?
- A robust domain model is easier to maintain
  (= long-term reliability).

Are we building the right system?
- CRC Cards and role playing validate use cases.
- Feature diagrams make product differences (and choices) explicit

Traceability?
- Via proper naming conventions
Testing

Regression Testing
- Deterministic tests (little user intervention), answering true or false

Are we building the system right?
- Tests only reveal the presence of defects, not their absence yet ... Tests verify whether a system is as right as it was before changes

Traceability?
- Link from requirements specification to system source code

Test techniques
- Individual test are white box or black box tests
  + White box: exploit knowledge of internal structure
    - e.g., path testing, condition testing
  + Black box: exploit knowledge about inputs/outputs
    - e.g., input- and output partitioning + boundary conditions
Design by Contract

Contractual Obligations Explicitly recorded in Interface
- pre-condition = obligation to be satisfied by invoking method
- post-condition = obligation to be satisfied by method being invoked
- class invariant = obligation to be satisfied by both parties

Are we building the system right?
- Recorded obligations prevent defects
- and ... remain in effect during changes

Traceability?
- Obligations express key requirements in source code

Weak or Strong?

<table>
<thead>
<tr>
<th></th>
<th>Invoking Method</th>
<th>Method being Invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>weak: input is unconstrained</td>
<td>strong: few input cases</td>
</tr>
<tr>
<td>Postcondition</td>
<td>strong: lots of results</td>
<td>weak: few results</td>
</tr>
</tbody>
</table>
Formal Specifications

Input/Output Specifications
• = include logic assertions (pre- and postconditions + invariants) in algorithm
  ➡ prove assertions via formal reasoning

Algebraic Specifications
• = type is described via the applicable operations
  ➡ deduce results via rewrite rules

Logic-Based (Model-based) Specifications
• = logic declarations state propositions that must be true

State-Based Specifications
• = Specify acceptable message sequences by means of state machine

Are we building the system right?
• Makes verification easier
  ➡ generation of test cases
  ➡ deduction of contractual obligations

Traceability?
• Extra intermediate representation may hinder traceability
Software Architecture

Software Architecture
• = Components & Connectors describing high-level view of a system.
• Decomposition implies trade-offs expressed via coupling and cohesion.
• Proven solutions to recurring problems are recorded as patterns.

Are we building the system right?
• For the non-functional parts of the requirements

Traceability?
• Extra level of abstraction may hinder traceability
Quality Control

Project Concern = Deliver on time and within budget

- External (and Internal) Product Attributes
- Process Attributes

Quality Control
- = include checkpoints in the process to verify quality attributes
- Formal technical reviews are very effective and cost effective!

Quality Standards (ISO9000 and CMM)
- = Checklists to verify whether a quality system may be certified

Are we building the system right?
Are we building the right system?
- Quality Control eliminates coincidence.

Traceability?
- Only when part of the quality plan/system
Software Metrics

Effort and Cost Estimation
• = measure early products to estimate costs of later products
• algorithmic cost modeling, i.e. estimate based on previous experience

Correctness?
• Algorithmic cost modeling provides reliable estimates (incl. risk factor)

Traceability?
• Quantification of estimates allows for negotiations

Quality Assurance
• = quantify the quality model
• Via internal and external product metrics

Correctness & Traceability?
• Software metrics are too premature too assure reliable assessment
Refactoring

Refactoring Operation
• = Behaviour-preserving program transformation
• e.g., rename, move methods and attributes up and down in the hierarchy

Refactoring Process
• = Improve internal structure without altering external behaviour

Code Smell
• = Symptom of a not so good internal structure
• e.g, complex conditionals, duplicated code

Are we building the system right ?
• Behaviour preserving ⇒ as right as it was before (cfr. tests)

Are we building the right system ?
• Improve internal structure ⇒ cope with requirements mismatches.

Traceability ?
• Renaming may help to maintain naming conventions
• Refactoring makes it (too) easy to alter the code without changing the documentation
CHAPTER 12 – Conclusion

Overview
- 1. Introduction
- 2. Project Management
- 3. Use Cases
- 4. Domain Modeling
- 5. Testing
- 6. Design by Contract
- 7. Formal Specification
- 8. Software Architecture
- 9. Quality Control
- 10. Software Metrics
- 11. Refactoring

Articles
- The Quest for the Silver Bullet
- The Case of the Killer Robot

Professional Ethics
- Cases

The future: Software Engineering Tools
Assignment: Study an Article of your Choice

➡ Find and read both of the following articles. Pick the one you liked the most, study it carefully and compare the article with the course contents.

The Quest for the Silver Bullet
  ➡ The article is more than 15 years old. Yet, it succeeds in explaining why there will never be an easy solution for solving the problems involved in building large and complex software systems.

The Killer Robot Case
  ➡ The article is a faked series of newspaper articles concerning a robot which killed its operators due to a software fault. The series of articles conveys the different viewpoints one might have concerning the production of quality software.
Code of Ethics

- Software Engineering Code of Ethics and Professional Practice
  + ACM-site: http://www.acm.org/serving/se/code.htm
  + IEEE-site: http://computer.org/tab/swecc/code.htm
- Recommended by
  + IEEE-CS (Institute of Electrical and Electronics Engineers - Computer Society)
  + ACM (Association for Computing Machinery)

  + Announces the revised 5.2 version of the Code

  + Discusses 9 cases of situations you might encounter and how (an older version of) the code address them
Software Engineering & Society

Lives are at stake 
(e.g., automatic pilot)

Huge amounts of money 
are at stake 
(e.g., Ariane V crash)

Software became Ubiquitous
Our society is vulnerable! 
⇒ Deontology, Licensing, ...

Corporate success or failure is at stake 
(e.g., telephone billing, VTM launching 2nd channel)

Your personal future is 
at stake (e.g., Y2K lawsuits)
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.

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.
Case: Privacy - Description

Case Description
• You consult a company concerning a database for personnel management.
• Database will include sensitive data: performance evaluations, medical data.
• System costs too much and company wants to cut back in security.

What does the code say?
• 1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.
• 3.12. Work to develop software and related documents that respect the privacy of those who will be affected by that software.

→ Situation is unacceptable.
Case study: Privacy - Solution

Applicable Clauses

- 1.02. Moderate the interests of the software engineer, the employer, the client and the users with the public good.
- 1.04. Disclose to appropriate persons or authorities any actual or potential danger to the user, the public, or the environment, that they reasonably believe to be associated with software or related documents.
- 2.07. Identify, document, and report significant issues of social concern, of which they are aware, in software or related documents, to the employer or the client.
- 6.09. Ensure that clients, employers, and supervisors know of the software engineer's commitment to this Code of ethics, and the subsequent ramifications of such commitment.

Actions

- Try to convince management to keep high security standards.
- Include in contract a clause to cancel contract when against the code of ethics.
- Alarm other institutions if you later hear that others accepted the contract.
Case: Unreliability

Case Description
• You’re the team leader of a team building software for calculating taxes.
• Your team and your boss are aware that the system contains a lot of defects. Consequently you state that the product can’t be shipped in its current form.
• Your boss ships the product anyway, with a disclaimer “Company X is not responsible for errors resulting from the use of this program”.

What does the code say?
• 1.03. Approve software only if they have a well-founded belief that it is safe, meets specifications, passes appropriate tests, and does not diminish quality of life, diminish privacy or harm the environment. The ultimate effect of the work should be to the public good.
• 5.11. Not ask a software engineer to do anything inconsistent with this Code.
• 5.12. Not punish anyone for expressing ethical concerns about a project.
  ➔ Disclaimer does not apply: can only be made in “good conscience”.
  ➔ In court you can not be held liable.
CHAPTER 12 – Conclusion

Overview

1. Introduction
2. Project Management
3. Use Cases
4. Domain Modeling
5. Testing
6. Design by Contract
7. Formal Specification
8. Software Architecture
9. Quality Control
10. Software Metrics
11. Refactoring

Articles

- The Quest for the Silver Bullet
- The Case of the Killer Robot

Professional Ethics

Cases

The future: Software Engineering Tools
Introduction

- Reliability vs. Agility

Mining Software Repositories

- Tests (= visualisation)
  + How good was our testing process?

- Bugs (= text mining)
  + Who should fix this bug?
  + How long will it take to fix this bug?
  + What is the severity of this bug?

- Expertise (= social network analysis)
  + Who are the key personalities?
  + Who can help me with this file?
  + Where should we focus our (regression) tests?

Conclusion

- The hype-cycle
Innovation

12. Conclusion

Underlying Technology

Technology changes every 20 years...

Underlying business models rarely change!

1529 — European coffee house
1475 — Kiva Han coffee house (Constantinople)
1908 — patent on paper filter
2001 — senseo
1946 — commercial piston espresso machine
1971 — Starbucks (Seattle)
1971 — Starbucks (Vienna)

Business Models

1971 — Starbucks (Seattle)
1529 — European coffee house
1908 — patent on paper filter
2001 — senseo
1946 — commercial piston espresso machine
1971 — Starbucks (Seattle)
1971 — Starbucks (Vienna)
Innovation in ICT

Underlying business models change often.

Technology changes every 5 years.
Market pressure in ICT

Measure of innovation
• # products in portfolio younger than 5 years
  + in ICT usually more than 1/2 the portfolio

Significant investment in R&D
• more products ... faster

RELIABILITY  ↔  AGILITY
Reliability vs. Agility

Software is vital to our society ⇒ Software must be reliable

Traditional Software Engineering
Reliable = Software without bugs

Today’s Software Engineering
Reliable = Easy to Adapt

Striving for RELIABILITY
(Optimise for perfection)

Striving for AGILITY
(Optimise for development speed)
Software Evolution

It is not age that turns a piece of software into a legacy system, but the rate at which it has been developed and adapted without being reengineered.

[Demeyer, Ducasse and Nierstrasz: Object-Oriented Reengineering Patterns]

Components are very brittle ... After a while one inevitably resorts to glue :)

Take action here ... … instead of waiting until here.
Software Repositories & Archives

Version Control
• CVS, Subversion, ...
• Rational ClearCase
• Perforce,
• Visual Source Safe
• ...

Automate the Build
• make
• Ant, Maven
• MSBuild
• OpenMake
• Build Forge

Automated Testing
• HP QuickTest Professional
• IBM Rational Functional Tester
• Maveryx
• Selenium
• TestComplete
• Visual Studio Test Professional Microsoft 2010
• ...

Issue Tracking
• Bugzilla
• BugTracker.NET
• ClearQuest
• JIRA
• Mant
• Visual Studio Team Foundation Server
• ...

All of a sudden empirical research has what any empirical science needs: a large corpus of objects to analyze.
[Bertrand Meyer's technology blog]

... mailing archives, newsgroups, chat-boxes, facebook, twitter, ...
The Mining Software Repositories (MSR) field analyzes the rich data available in software repositories to uncover interesting and actionable information about software systems and projects.

**Conferences**
- 2012—9th edition, Zürich, CH
- 2011—8th edition, Honolulu, HI, USA
- 2010—7th edition, Cape Town, ZAF
- 2008—5th edition, Leipzig, DEU
- 2007—4th edition, Minneapolis, MN, USA
- 2006—3rd edition, Shanghai, CHN
- 2005—2nd edition, Saint Luis, MO, USA
- 2004—1st edition, Edinburgh, UK

**Hall of Fame—Mining Challenge Winners**
- 2011—Apples Vs. Oranges? An exploration of the challenges of comparing the source code of two software systems (Daniel M. German and Julius Davies)
- 2010—Cloning and Copying between GNOME Projects (Jens Krinke, Nicolas Gold, Yue Jia, and David Binkley)
- 2009—On the use of Internet Relay Chat (IRC) meeting by developers of the GNOME GTK+ project (Emad Shihab, Zhen Ming Jiang and Ahmed E. Hassan)
- 2008—A newbie's guide to Eclipse APIs (Reid Holmes and Robert J. Walker)
- 2007—Mining Eclipse Developer Contributions via Author-Topic Models (Erik Linstead, Paul Rigor, Sushil Bajracharya, Cristina Lopes, and Pierre Baldi)
- 2006—A study of the contributors of PostgreSQL (Daniel M. German)
The Future — Software Engineering Tools

Introduction
• Reliability vs. Agility

Mining Software Repositories

• Tests (= visualisation)
  + How good was our testing process?

• Bugs (= text mining)
  + Who should fix this bug?
  + How long will it take to fix this bug?
  + What is the severity of this bug?

• Expertise (= social network analysis)
  + Who are the key personalities?
  + Who can help me with this file?
  + Where should we focus our (regression) tests?

Conclusion
• The hype-cycle
12. Conclusion

Test Monitor — Change History

http://swerl.tudelft.nl/bin/view/Main/TestHistory  
Case = Checkstyle

Diagram showing:
- Integration tests
- Phased Testing
- Steady Testing & Coding
- Single Test
- Unit Testing
Test Monitor — Growth History

12. Conclusion

- Few Tests
- Steady Testing & Coding
- Increased Test Activity

Diagram showing growth history with various release milestones and test activity levels.
Test Monitor — Coverage Evolution

12. Conclusion
The Future — Software Engineering Tools

Introduction
• Reliability vs. Agility

Mining Software Repositories

• Tests (= visualisation)
  + How good was our testing process?

• Bugs (= text mining)
  + Who should fix this bug?
  + How long will it take to fix this bug?
  + What is the severity of this bug?

• Expertise (= social network analysis)
  + Who are the key personalities?
  + Who can help me with this file?
  + Where should we focus our (regression) tests?

Conclusion
• The hype-cycle
Before reporting a bug, please read the bug writing guidelines, please look at the list of most frequently reported bugs, and please search for the bug.

Reporter: karsten.thoms@itemis.de
Version: 4.2.1
4.3.0
4.3.1
4.3.1 RC1
4.3.1 RC2
Severity: enhancement
Priority: P5
Initial State: NEW
Assign To:
Cc:
Default CC:
Estimated Hours: 0.0
Deadline: (YYYY-MM-DD)
URL: http://
Summary:
Description:
Attachment: Add an attachment
Depends on:
Blocks:

We've made a guess at your operating system and platform. Please check them and, if we got it wrong, email karsten.thoms@itemis.de.
Bug Database

(1) & (2)
Extract and preprocess bug reports

Bug Reports

(3) Training

New report
(4) Predict
• severity
• assigned-to
• estimated time

predictor
prediction

• true positive
• false positive
• true negative
• false negative
⇒ precision & recall
## Results

<table>
<thead>
<tr>
<th>Question</th>
<th>Cases</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who should fix this bug ?</td>
<td>Eclipse, Firefox, gcc</td>
<td>eclipse: 57%</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>firefox: 64%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gcc: 6%</td>
<td></td>
</tr>
<tr>
<td>How long will it take to fix this bug ?</td>
<td>JBoss</td>
<td>depends on the component</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>many similar reports: off by one hour</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>few similar reports: off by 7 hours</td>
<td></td>
</tr>
<tr>
<td>What is the severity of this bug ?</td>
<td>Mozilla, Eclipse, Gnome</td>
<td>mozilla, eclipse: 67% - 73%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gnome: 75%-82%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mozilla, eclipse: 50% - 75%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>gnome: 68%-84%</td>
<td></td>
</tr>
</tbody>
</table>

Promising results but ...
- how much training is needed ?
- how reliable is the data ? (estimates, severity, assigned-to)
- does this generalize ? (on industrial scale ?)

⇒ replication is needed
Introduction

- Reliability vs. Agility

Mining Software Repositories

- Tests (= visualisation)
  + How good was our testing process?

- Bugs (= text mining)
  + Who should fix this bug?
  + How long will it take to fix this bug?
  + What is the severity of this bug?

- Expertise (= social network analysis)
  + Who are the key personalities?
  + Who can help me with this file?
  + Where should we focus our (regression) tests?

Conclusion

- The hype-cycle
12. Conclusion

Fig. 13.1 Communication paths extracted from the Eclipse Platform Core developer mailing list.

A mail addressed to a mailing list is processed by the reflector and sent to all subscribers. This means that the To: address is always the mailing list address itself; hence, there is no explicit receiver address. In our example this address is platform-core-dev.

The receiver needs to be reconstructed from subsequent answering mails. The identification of the sender is given by the From: field which is denoted by the name on the right side of a message. For determining the receivers of emails we analyze the tree structure of a mail thread and compute the To: and Cc: paths.

Figure 13.1 illustrates the two paths in our example thread whereas gray arrows denote the To: path and light gray arrows the Cc: path. A gray arrow is established between an initial mail and its replies. For example, Philippe Ombredanne is first replying to the mail of Thomas Watson, so in this case Philippe Ombredanne is the sender and Thomas Watson is the receiver of the mail.

To derive Cc: receivers we consider the person answering a mail as an intended receiver of this mail. In case this person is already the To: receiver (as it applies with the mails number 3–5 between Bob Foster and Pascal Rapicault) no additional path is derived, because we assume that a mail is not sent to a person twice.

For importing the data from the mailing lists archives we extended the iQuest tool. iQuest is part of TeCFlow, as e to ft ool st ov i s u a l i z et h et e m p o r a l e v o l u t i o n of communication patterns among groups of people. It contains a component to parse mailing lists and import them into a MySQL database. Our extension aims at including the follow-up information of mails to fully reconstruct the structure of a mail-thread. The sample thread shown above consists of 15 mails that result in 25 communication paths.

13.3.1.2 Deriving Communication Paths from Bug Reports

The second source outlined for modeling communication paths is a bug tracking repository, such as, Bugzilla. Bugzilla users create reports and comments and give

More information on Bugzilla and its usage can be found at http://www.bugzilla.org/.

While this algorithm works fine for person information obtained from Bugzilla and mailing lists, there are problems with matching persons obtained from CVS log data. Typically, the author stored in CVS logs indicates the CVS user name, but not the real name of a person. Because of the high number of false matches, the mapping of these persons is done manually.

In addition to the information of a person, email addresses contain domain information that, for example, denotes the business unit of a developer. We use this information to assign developers to teams. We obtain email addresses that have been generated with MHonArc.

The problem is that MHonArc provides a spam mode which deters spam address harvesters by hiding the domain information of email addresses. For example, the email address of Chris McGee is displayed as

More information on MHonArc can be found at http://www.mhonarc.org/.

Construct Social Network

Mailing lists + Bug reports + CVS Logs
12. Conclusion

Code Ownership (& Alien Commits)

Fig. 13.7 Collaboration in the Eclipse Platform Core project observed in the time from 14th to 28th February 2005.

Information ideally gets processed. Figure 13.8 illustrates the communication via the developer mailing list and Bugzilla data over 21 months. The amount of communication (i.e., the number of communication paths reconstructed from bug reports and emails) is illustrated by the width of edges. The wider the edges of a person’s node are, the more this person communicated with other developers.

The graph in Fig. 13.8 shows the core development team whose members frequently communicate with each other. Rafael Chaves, Dj Houghton, Jeff Mcaffer, Thomas Watson, John Arthorne, and Pascal Rapicault form the core team. They are the communicators who keep the network together and play an important role within the project. Interesting is that they all belong to either the group @ca.ibm.com or @us.ibm.com as indicated by the shadows of rectangles representing these developers.

Another highly connected group is formed by the Swiss team (@ch.ibm.ch) whose members are represented by the nodes on the right side of the graph. Almost each developer of the Swiss team is in touch with the US team; however, Markus Keller and Daniel Megert turn out as the main communicators between the two teams during that time.

Another interesting finding concerns the environment via which the developers communicated. Most of the communication was via Bugzilla bug reports indicated by the gray edges. Only the core team also used the mailing list to discuss Eclipse.
13.5.4 Project Dynamics

Newcomers should be integrated fast into development teams to rapidly increase productivity and foster synergy among team members. With STNA-Cockpit the project manager can observe how newcomers actually are integrated into their teams. For this, the project manager selects the starting observation period and uses the time-navigation facility of STNA-Cockpit to investigate the evolution of the communication and collaboration network over time. The graph animation allows the project manager to observe how the newcomer behaves concerning communication and collaboration with other team members. In particular, she looks for communication paths that tell her the newcomer gets actively involved.

Key personalities

Communicators of the Eclipse Platform Core project as from May 2004 to February 2006

Fig. 13.8

Such findings are of particular interest when new ways of communication are considered.

"swiss group"
Consider the following scenario in which Kevin Barness is entering the US team of the Eclipse Platform Core project in April 2004. Figure 13.9 depicts various snapshots taken from the network created for subsequent points in time. Kevin Barness is starting as a developer in the Eclipse Platform Core team at the beginning of April 2004. His first action is to get in touch with some key personalities of the project, namely Rafael Chaves and John Arthorne. His first contacts are visualized by the graphs depicted by (Fig. 13.9a, b). In the following weeks he communicates also with other project members to get more involved into the project (see Fig. 13.9c), namely Darin Wright and Darin Swanson. As (Fig. 13.9d) illustrates, Darin Wright is a developer and Darin Swanson the owner of the files that are going to be modified by Kevin. Rafael Chaves seems to play the role of the connector who introduces the new developer Kevin Barness to the responsible persons. According to the graph, he is communicating with two senior developers.
12. Conclusion

Expertise Browser

Used within a geographically dispersed team
• 120 developers at two sites (Germany and England) grew to 250 developers (incl. satellite site in France)
• satellite teams: locate expertise
• established teams: who is doing what?
Code Ownership vs. Code Quality

Software components with a high level of ownership will have fewer failures than components with lower top ownership levels.

Software components with many minor contributors will have more failures than software components that have fewer.

Data from Windows Vista and Windows 7

12. Conclusion
Table Of Contents

Introduction
• Reliability vs. Agility

Mining Software Repositories

• Tests (= visualisation)
  + How good was our testing process ?

• Bugs (= text mining)
  + Who should fix this bug ?
  + How long will it take to fix this bug ?
  + What is the severity of this bug ?

• Expertise (= social network analysis)
  + Who are the key personalities ?
  + Who can help me with this file ?
  + Where should we focus our (regression) tests ?

Conclusion
• The hype-cycle
Hype Cycle

12. Conclusion

Visibility

Maturity

Technology Trigger

Peak of Inflated Expectations

Trough of Disillusionment

Plateau of Productivity

Slope of Enlightenment

Hype Cycle © Gartner
The Future?

Personal Opinion

Visibility

Maturity

Hype Cycle © Gartner

12. Conclusion
Summary (i)

You should know the answers to these questions

- Name 3 items from the code of ethics and provide a one-line explanation.
- If you are an independent consultant, how can you ensure that you will not have to act against the code of ethics?
- What would be a possible metric for measuring the amount of innovation of a manufacturing company?
- What can we do to avoid that glue-code erodes our component architecture?
- Explain in 3 sentences how a software tool could recommend on certain fields in a bug report (severity, assigned-to, estimated time).
- Which components would be more susceptible to defects: the ones with a high level of ownership or the ones with a low level of ownership?

When you chose the “No Silver Bullet” paper

- What’s the distinction between essence and accidents?
- Name 3 reasons why the building of software is essentially a hard task? Provide a one-line explanation.
- Why is “object-oriented programming” no silver bullet?
- Why is “program verification” no silver bullet?
- Why are “components” a potential silver bullet?

When you chose the “Killer Robot” paper

- Which regression tests would you have written to prevent the “killer robot”?
- Was code reviewing applied as part of the QA process? Why (not)??
- Why was the waterfall process disastrous in this particular case?
- Why was the user-interface design flawed?
Summary (ii)

Can you answer the following questions?

- You are an experienced designer and you heard that the sales people earn more money than you do. You want to ask your boss for a salary-increase; how would you argue your case?
- Software products are usually released with a disclaimer like “Company X is not responsible for errors resulting from the use of this program”. Does this mean that you shouldn’t test your software? Motivate your answer.
- You are a QA manager and are requested to produce a monthly report about the quality of the test process. How would you do that?
- Give 3 scenarios to illustrate why it would be interesting to know which developers are most experienced with a given piece of code.
- It has been demonstrated that it is feasible to make reliable recommendations concerning fields in a bug report. What is still needed before such a recommendation tool can be incorporated into state-of-the-art tools (bugzilla, jira, ...)

When you chose the “No Silver Bullet” paper

- Explain why incremental development is a promising attack on conceptual essence. Give examples from the different topics addressed in the course.
- “Software components” are said to be a promising attack on conceptual essence. Which techniques in the course are applicable? Which techniques aren’t?

When you chose the “Killer Robot” paper

- Recount the story of the Killer Robot case. List the three most important causes for the failure and argue why you think these are the most important.