CSEE&T 2016 Panel IV

Delivering Software Engineering Content to Computer Science Majors

Donald J. Bagert, Mike Barker, Dick Fairley, David Kung

Conference on Software Engineering and Training
Dallas, Texas, USA
April 6, 2016, 1:30-3:00 pm

Delivering Software Engineering Content to Computer Science Majors

Donald J. Bagert
Benedictine College
dbagert@benedictine.edu

Conference on Software Engineering and Training
Dallas, Texas, USA
April 6, 2016

Introduction

- The Panelists
- Audience: What's Your Background?
- Purpose of This Panel
- Panel Format

Purpose of This Panel - 1

- Software Engineering undergraduate degree programs have emerged
- However, most software is developed by people with CS and other backgrounds
- There is limited space available for SE content in CS curricula
  - So it must be used effectively

Purpose of This Panel - 2

- This has been an ongoing challenge for decades
- Little consensus on format has occurred
- Topic of CSEE&T 2015 Keynote
  - This panel is a follow-up to it
- Want to provide you with “real takeaways” you can adopt relatively easily

Panel Format

- Panelist Position Statements
- Any Follow-Up Comments from Panel?
- Questions from the Audience

Thank You!!!!!!!!!!!!!

Delivering Software Engineering Content to Computer Science Majors

David Kung
Department of Computer Science and Engineering
The University of Texas at Arlington
kung@uta.edu
Conference on Software Engineering Education and Training
Dallas, Texas, April 5-6, 2016

Topics for Discussion

- How I View Process and Methodology
- How I View Agile Development
- How I Would Teach That SE Course(s)

Process and Methodology

- We’ve focused on processes, and spent tons of $$$.
- We’ve confused process with methodology, and vice versa.
- We’ve been teaching processes but rarely any methodology.
- Process alone will not make students produce quality software.

Process and Methodology

Process

- Defines a framework of phased activities
- Specifies phases of WHAT
- Does not dictate representations of artifacts
- It is paradigm-independent
- A phase can be realized by different methodologies.

Examples

Waterfall, spiral, prototyping, unified, and agile processes

Methodology

- Defines steps to carry out phases of a process
- Describes steps of HOW
- Defines representations of artifacts (e.g., UML)
- It is paradigm-dependent
- Steps describe procedures, techniques & guidelines

Examples

Structured analysis/structured design (SA/SD), Object Modeling Technique (OMT)

Process and Methodology

- "A Rational Design Process: How and Why to Fake It," David Parnas and Paul Clements, IEEE TSE, 1986.

Methodology: 4 Categories

- Normative: sequence of steps known to work for the discipline
- Rational (nothing to do w/the acquired company): based on methods and techniques
- Participative: stakeholder based, customer involvement
- Heuristic: based on lessons learned

Maier and Rechtin, 2000

Process and Methodology

“methodologies move from heuristic to normative and become ... standard solutions ... search ***algorithms*** have reached that point.”

“Most of software development is still in the stage where ***heuristic*** methodologies are appropriate.”

--- Alistair Cockburn,

“Agile Software Development,” 2nd Ed. 2007.

How I View Agile Development

- Companies are rapidly moving to agile, and hire graduates who know agile.
- Students want to learn and practice agile development.
- Agile development needs an agile development methodology.
- An agile development methodology helps students learn modeling, analysis, and design skills, and build up their confidence in the job market..

Agile Methodology Is Needed

“Without the structure of ‘heavyweight’ processes, XP actually requires **more** self-discipline to use; self-discipline that inexperienced students don't have. Students use XP as an excuse to adopt a sloppy, poor quality development style.”

John Dalby, “An XP failure in under-graduate SE”

Agile Methodology Is Needed

Most software is written w/o requirements/ diagrams and agile methodology such as Scrum that many companies follow loosely only makes things worse. Most students do not receive a formal education in analysis and design and we see the result in the current state of software.

--- An anonymous reviewer

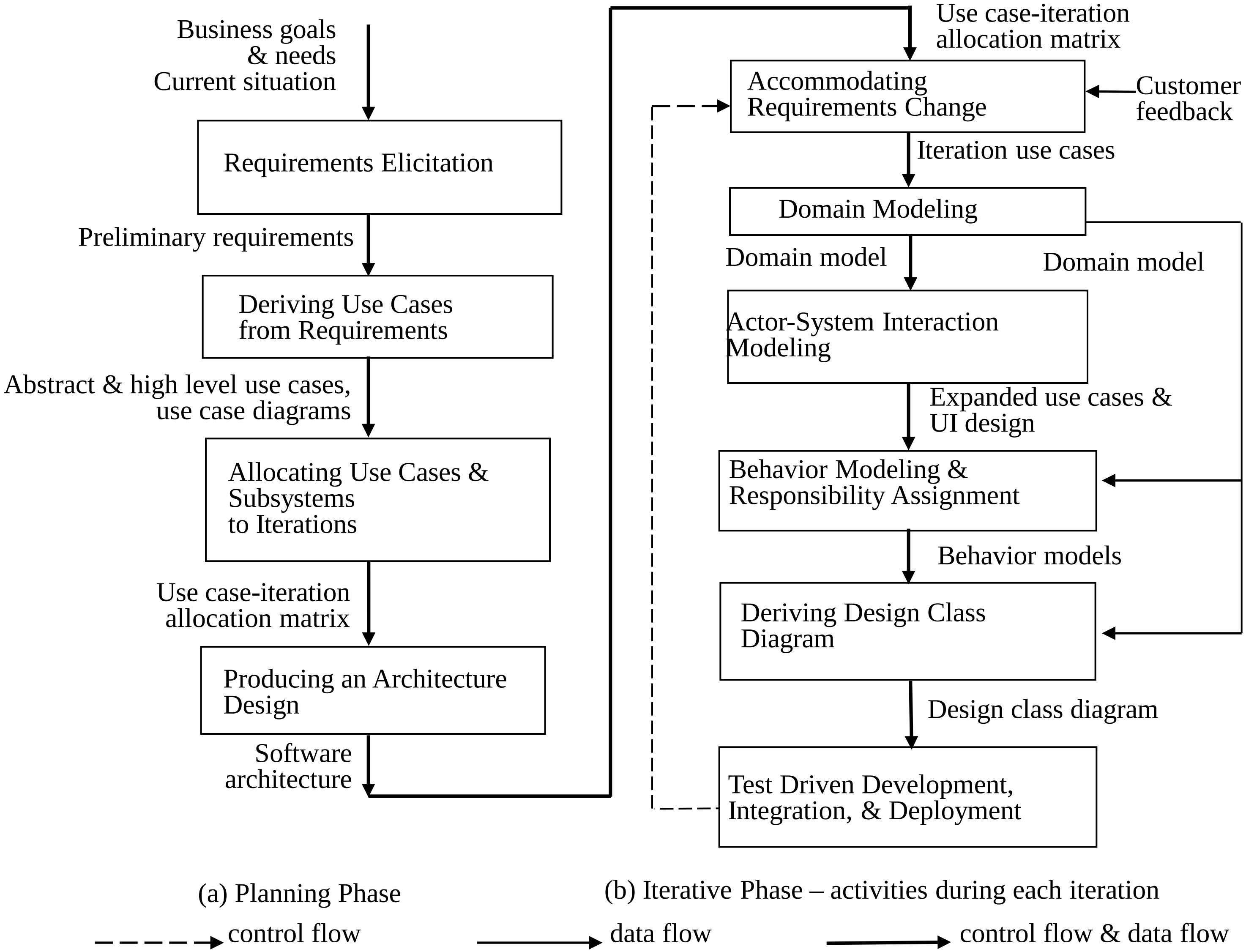
Teaching The SE Course

- Select a team project that is big enough to show challenges and small enough to fit into one or two semesters.
- Choose waterfall or agile as you feel comfortable with; agile uses N iterations, set N=1 for waterfall.
- Teach selected SE topics along with the development of the team project.
- Tool support to the methodology: manual, semi-automatic, and automatic

Teaching SE Course: Example

- Select a team project: personal calendar, room reservation, academic advising, or car rental ...
- Choose agile: set N=3, may cut to 2.
- Teach a waterfall, and/or an agile process (differ in sequential/iterative, agile manifesto, and agile principles)

An Agile Process



Teaching The SE Course

- Week 1:
  1. present project, require all team members to take notes
  2. in addition to above, require students to survey existing products, and make a feature list
- Teach requirements acquisition and specification
- Assignment: produce a prioritized list of requirements for the project, due in one week.
- ***Good enough is enough!***

Teaching The SE Course

- Week 2: Deriving use cases from requirements.
- Teach “what is a use case,” and related concepts, how to derive use cases from requirements, use case diagram
- Select one of the best requirements specs and use it as the SRS; modify if needed.
- Assignment: Deriving use cases from SRS, due in one week.

Teaching The SE Course

- If needed and time permits, teach effort estimation for use cases
- Teach or ask students to study the poker game agile estimation method
- Don’t worry about the estimates are correct or not, the purpose is learning a useful method/technique

Teaching The SE Course

- Week 3: Domain Modeling
- Teach WWWH, and UML class diagram
- Assignment: Construct a domain model for the application of the project, due in one week.

Teaching The SE Course

- Week 4: Actor-System Interaction Modeling (ASIM)
- Teach WWWH
- Select 1, 2, or 3 use cases to develop in iteration 1; it is not the more the better; quality is king. Note use case priorities and dependencies.
- Assignment: Conduct ASIM for selected use cases, due in one week. UI prototypes may be produced.

Teaching The SE Course

- Week 5: Iteration 1 team presentations.
- All students must attend; 10% deduction if absent w/o prior permission.
- Students sign in with TA at room entrance; x points deduction for late arrival.
- Students are encouraged to make remarks and ask questions while other teams present. Instructor may stimulate students to do these.

Teaching The SE Course

- Weeks 6-7: Object Interaction Modeling.
- Teach WWWH, scenario, sequence diagram
- Teach responsibility assignment patterns (controller, expert, creator)
- Assignment: produce sequence diagrams from the ASIMs for the selected use cases, apply patterns, due in one week.

Teaching The SE Course

- Week 8: Deriving Design Class Diagram
- Teach WWWH
- Assignment: derive DCD from the sequence diagrams.

Teaching The SE Course

- Week 9: Implementation and Testing
- Teach implementation consideration, code review, test techniques, Junit, test driven development, etc.
- Assignment: Implement the selected use cases, test some of the classes, due depends on the difficulty and student preparation.
- Implementation and testing are optional but recommended.

Teaching The SE Course

- Week 10: Iteration 2 team presentations
- Same as iteration 1 but this time student teams may be required to demo the use cases implemented (demo as much as they have been able to implement)
- Software demo may take place at a later time (say one or two weeks later).

Teaching The SE Course

- Week 11-15: Repeat for iteration 3 with new use cases
- Teach SQA, SCM, project management, and/or other topics.
- If waterfall process is used, then weeks 1-8 perform analysis and design, weeks 9-15 perform implementation, testing and demo.

Delivering Software Engineering Content to Computer Science Majors

Dick Fairley
Software and Systems Engineering Associates (S2EA)
d.fairley@computer.org

Conference on Software Engineering Education and Training
Dallas, Texas, April 5-6, 2016

Good news and bad news

- Good news:
  - We can expect (hope) that students in an upper division SwE course will be familiar with algorithms, data structures, and programming-in-the-small
- Bad news:
  - We will probably have only two SwE courses
    - A pedagogical and a project course
    - Or two project courses

The challenge

- How do we teach large-scale issues in a small-scale setting?
- To students who may not be motivated
  - Because they don’t have the experience or context to understand the issues and why they are important

Topics

- Skills-based education & training
- Industrial experience for faculty members
- Some backup slides
  - Teaching interdisciplinary projects

Competency and skills

- A competent individual has the knowledge, skills, and ability to perform assigned tasks efficiently and effectively, *at a given level of competency*
  - Knowledge is what one knows
  - Skill is what one can demonstrably do
  - Ability includes the cognitive and interpersonal attributes that enable an individual to perform assigned tasks efficiently and effectively

Knowledge and ability are prerequisites of skill

SWECOM Competency Levels

- SWECOM includes five competency levels for software engineering technical activities:
  1. technician
  2. entry level practitioner
  3. practitioner
  4. technical leader
  5. senior software engineer
- Some activities do not include competencies at all five competency levels
  – e.g., no technician level skills for selecting a team software process

For more information

- For more information see the

  Software Engineering Competency Model (SWECOM)

  https://www.computer.org/web/peb/swecom.
- And the Software Assurance Competency Model

  http://www.sei.cmu.edu/reports/13tn004.pdf.

How to impart skills?

- Each course has a term project
- With weekly project deliverables that cumulatively result in a final report
  - Weekly deliverables build on previous ones and can be revised based on instructor feedback
  - Earlier assignments may also have to be revised
    - Welcome to the real world
- Project assignments, for each class, require application of specific skills to solve a problem
  - Classroom exercises introduce the skills

How to assess individual skills?

- Presentations
- Observations
- Demonstrations
- Weekly project deliverables
- Weekly status reports
- Preparation of individual portfolios of work

Weekly individual status reports

- Weekly individual status reports
  - What did you plan to accomplish last week?
  - What did you accomplish?
  - What new skills did you learn and use last week?
  - What other skills did you apply?
  - What do you need help with?

What about team projects?

- Often, team projects do not build skills for the project members
  - Joe will do the coding
    - which he knows how to do – kind of
  - Sue will do the testing
    - which she knows how to do – kind of
  - John will do the documentation
    - which he knows how to do – kind of
  - The delivered product works – kind of
- Result: no one acquires any new individual or team skills

An approach to building software engineering team skills

- Team jointly interviews the project customer and develops the requirements in consultation with the customer
  - And jointly develop and review use cases
  - Using text, state diagrams, and sequence diagrams to document scenarios
- Team adopts an architectural pattern and a design metaphor to guide design decisions
- Team uses an agile development process with all team members participating
- Guidelines
  - Teams do project work during weekly class time
  - With periodic customer involvement
  - And instructor coaching

How to assess team skills

- Observation of team at work
- Weekly demonstrations of team progress
- Weekly individual progress reports
- Weekly progress and planning meetings
  - Using burndown and velocity charts
  - With instructor coaching
- Monthly confidential peer reviews
- Monthly meetings with individual team members

How to assess individual skills within teams?

Weekly individual reports

- Individual skills
  - What did you plan to accomplish last week?
  - What did you accomplish?
  - What new skills did you learn and use last week?
  - What other skills did you apply?
  - What do you need help with?
- Team skills
  - How did your work contribute to the team effort and to project success?
  - What needs to happen for you and your team to be more successful?

How can faculty members gain needed practical experience?

- Summer internships
- Consulting with local industry
- Visiting resident industry faculty member
  - To team teach SwE courses and projects
  - To give seminars on industry practices
  - To conduct research for his or her organization
- Industrial advisor for student team projects
  - To advise students
  - (and faculty members)

Backup slides

Teaching interdisciplinary projects

A couple of quotes from SEBoK*

- “Systems engineering and software engineering are not merely related disciplines; they are intimately intertwined.”
- “The SEBoK explicitly recognizes and embraces the intertwining between systems engineering and software engineering as well as defining the relationship between the SEBoK and the Guide to the Software Engineering Body of Knowledge (SWEBOK) (Bourque and Fairley 2014).”

* A Guide to the Systems Engineering Body of Knowledge (SEBoK)
www.sebokwiki.org

Another quote

- “The link between systems engineering and software engineering is broken and needs to be fixed.”

  David Long

  INCOSE president, 2015

Interdisciplinary projects

- Some schools teach interdisciplinary senior design courses that include computer science and software engineering students
- Many people recognize that software is an essential element of most modern systems
  - Software provides most of the functionality, behavior, and quality attributes in modern systems
  - Plus the interfaces among system components and to the external environment

Some kind-of good news

- Good news: the importance of software in interdisciplinary systems is often recognized
  - It is generally understand that software is necessary
  - but not what it is or what to do about it

A paraphrase

Charles Dudley Warner almost said:

“Everyone talks about software but no one knows what to do about it”

Now for the bad news

- Bad news #1: faculty (including CS, SwE, and others) don’t know what to do about it
- Bad news #2: the thoughts and opinions of CS and SwE students are often ignored in multidisciplinary team meetings
  - Welcome to the real world
- Bad news #3: software development is usually relegated to lower levels in a system hierarchy
  - And partitioned among the system components
  - And you software guys figure out how to make it all work together

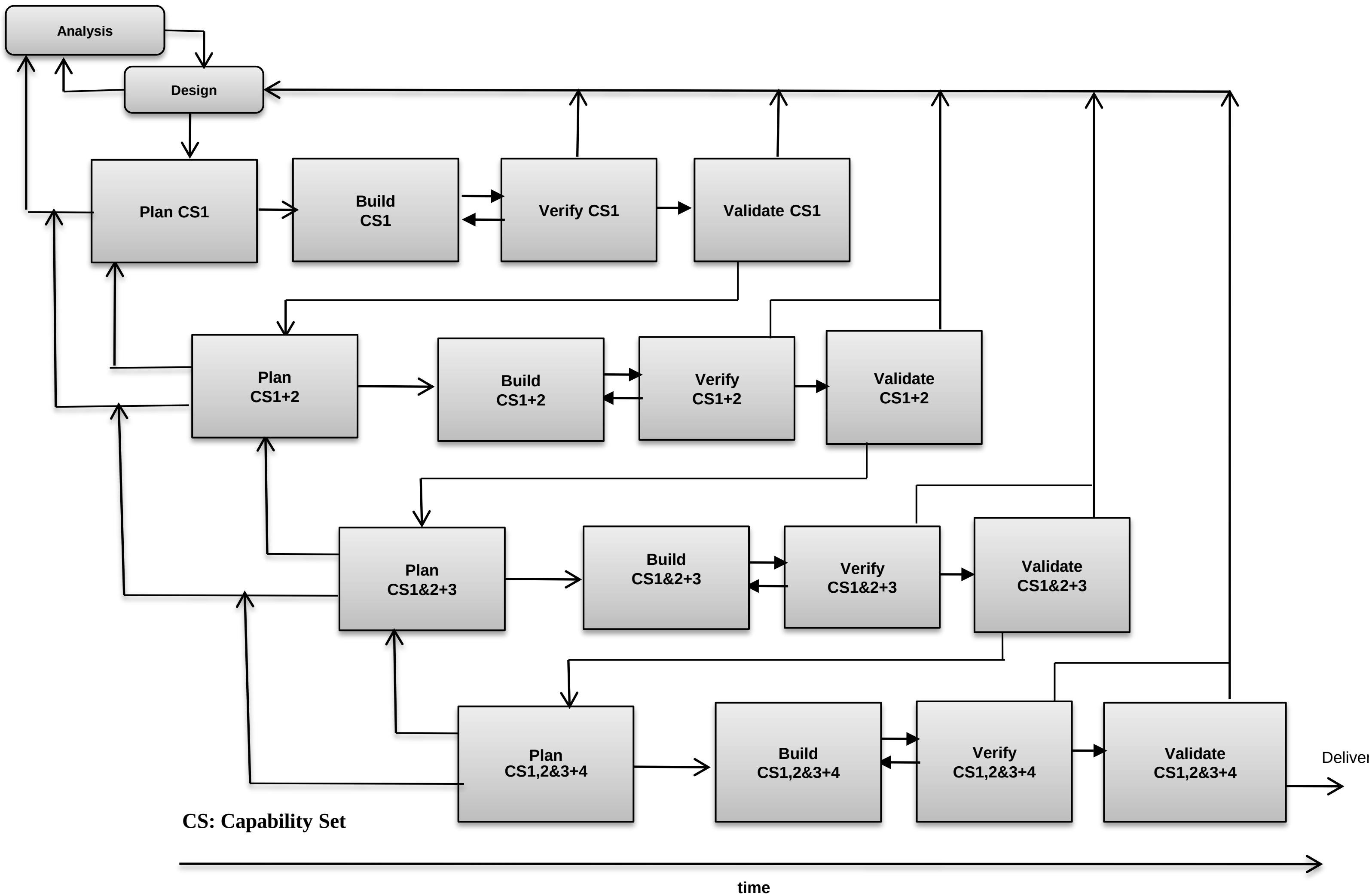
Traditional engineering and software engineering

- Development of physical systems is based on variants of the waterfall model
  - Because of the nature of physical entities
- Software engineering has much more flexibility in development processes
  - Because of the nature of software
- An issue: how to seamlessly integrate traditional engineering and software engineering processes?

An approach for interdisciplinary project courses (and for real-world projects)

- CS/SwE students lead the preliminary analysis phase
  - And involve other engineers in developing use cases
  - And participate in developing the system architecture
  - And participate in making design tradeoff decisions
- Model-based system development is done using real and simulated hardware and software
  - Using an incremental development process based on partitioned system-level capability sets
  - Capability sets are partitioned using one or more prioritization criteria

An incremental system development process



Note

- This approach is consistent with Barry Boehm's Incremental Commitment Model
- A good ICM introductory paper is:

  Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering

  http://csse.usc.edu/TECHRPTS/2007/usc-csse-2007-715/usc-csse-2007-715.pdf.

Incremental System Partitioning Criteria

- Develop an architectural skeleton of simulated components with interfaces and communication protocols among them; then incrementally build and demonstrate real components based on capability sets prioritized by functionality, behavior, and quality attributes.
- Allocate requirements to one or more initial capability sets that include the most difficult, highest risk elements of a system; then incrementally add other system elements based on capability sets prioritized by functionality, behavior, and quality attributes.

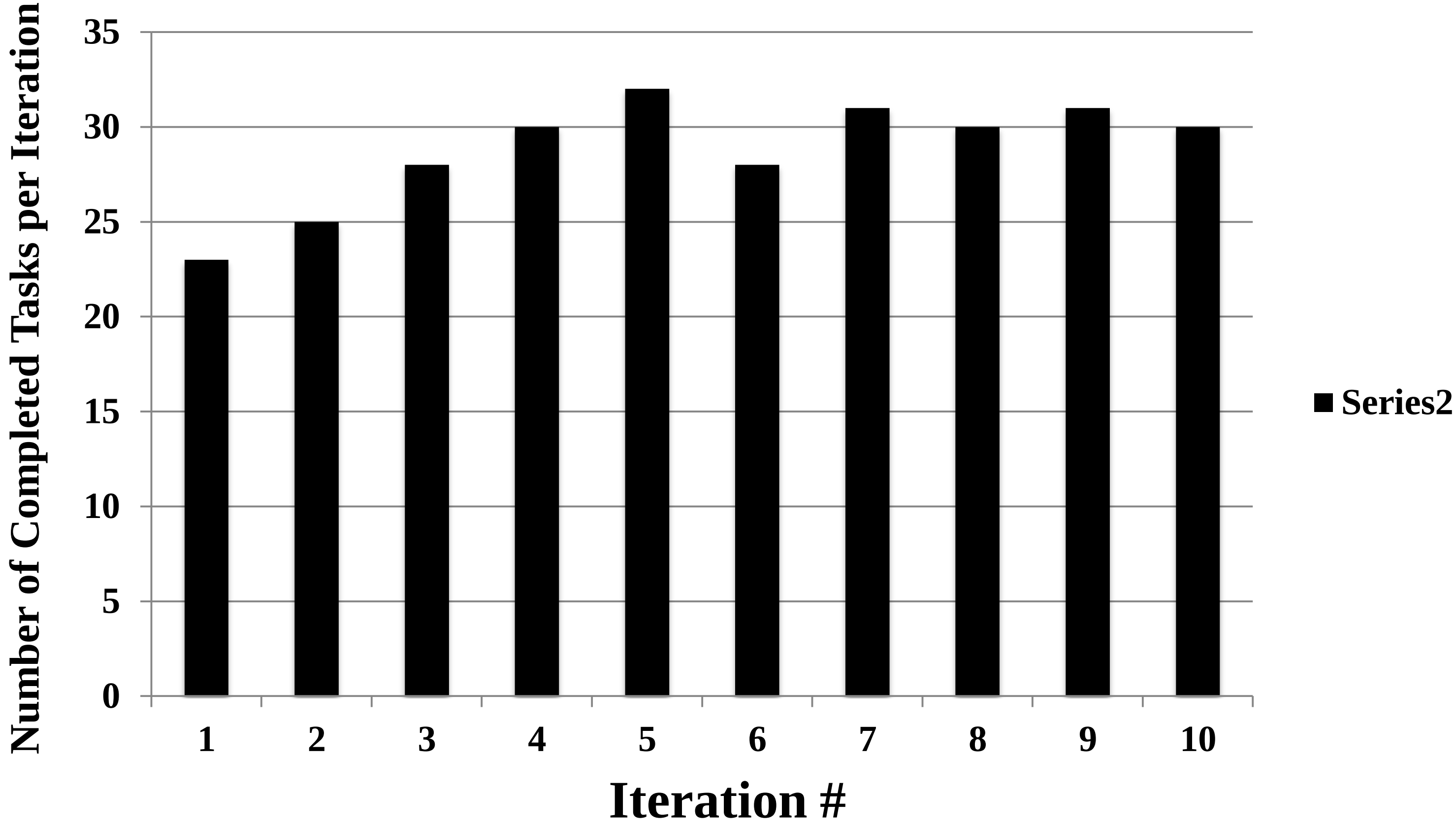
Incremental Partitioning Criteria - 2

- Allocate requirements to one or more initial capability sets that include the easiest, lowest risk elements of a system to learn about and demonstrate the feasibility of incremental system development; then incrementally add other system elements based on capability sets prioritized by functionality, behavior, and quality attributes.
- Allocate requirements to capability sets that initially evaluate the acceptability of acquired components and those to be used and reused; then incrementally add other system elements based on capability sets prioritized by functionality, behavior, and quality attributes.

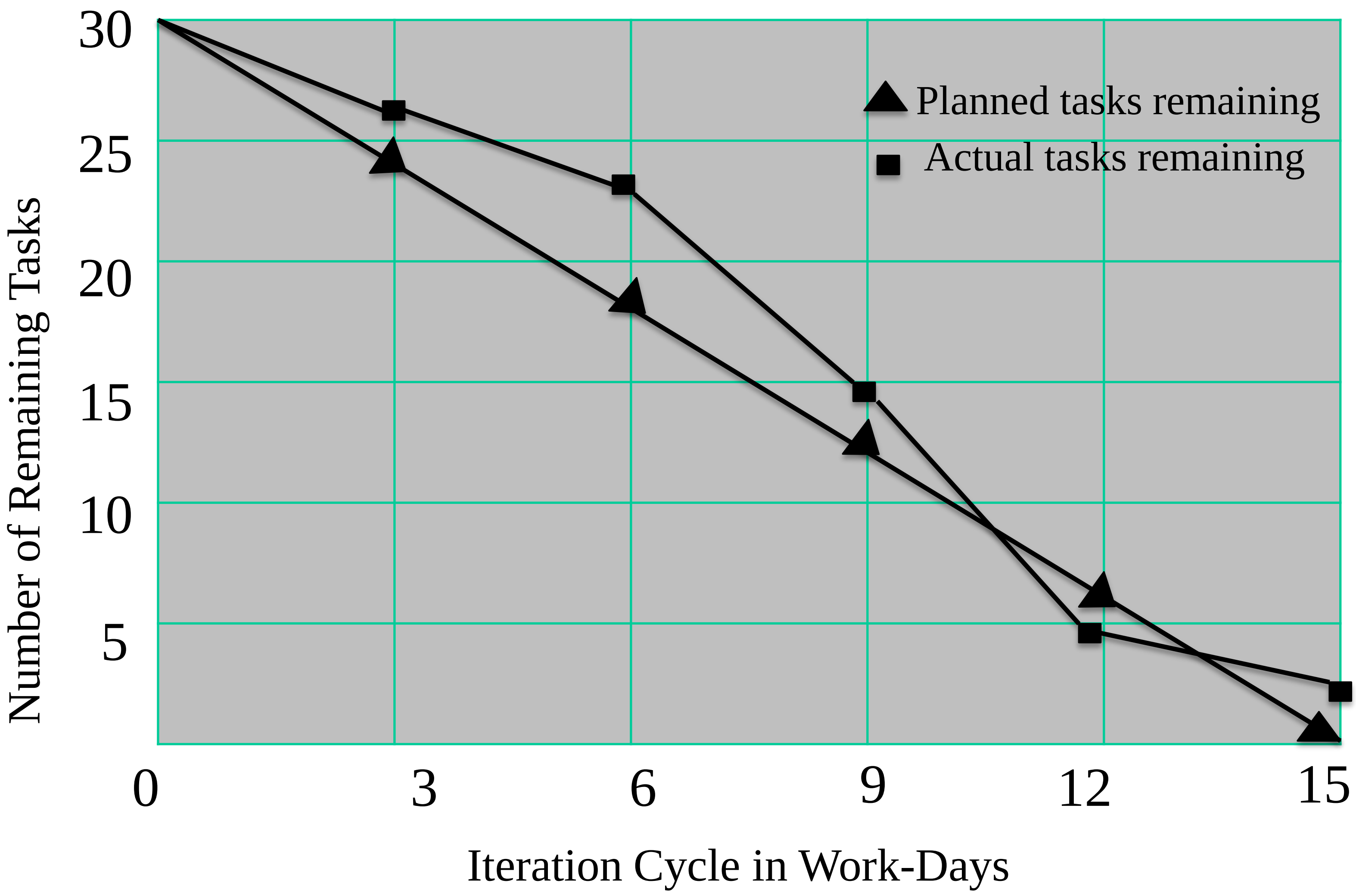
Incremental Partitioning Criteria - 3

- Allocate the system requirements to capability sets that incrementally result in a succession of virtual machines.
- Allocate the system requirements to capability sets of growing system capabilities to be periodic delivered into the operational environment in a preplanned manner.

Tracking Progress Using a Velocity Chart



A Burndown Chart for an Iteration Cycle

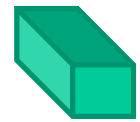


What does everyone know about Software Engineering?

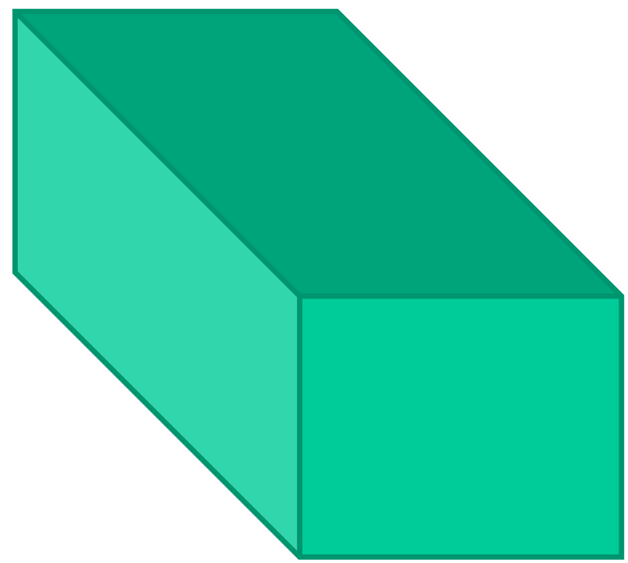
Mike Barker
Nara Institute of Science and Technology, Japan
mbarker@is.naist.jp

It's simple!
Just some coding…

- A man-month of coding?

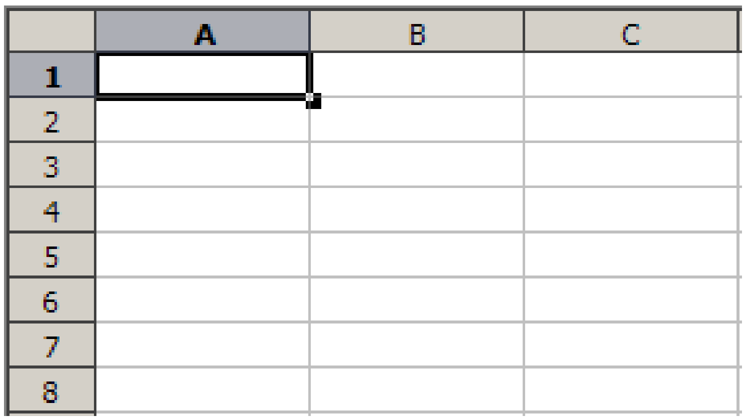


- Or a two-year, four-person project?



How hard can it be?

A spreadsheet doesn't need software engineering!

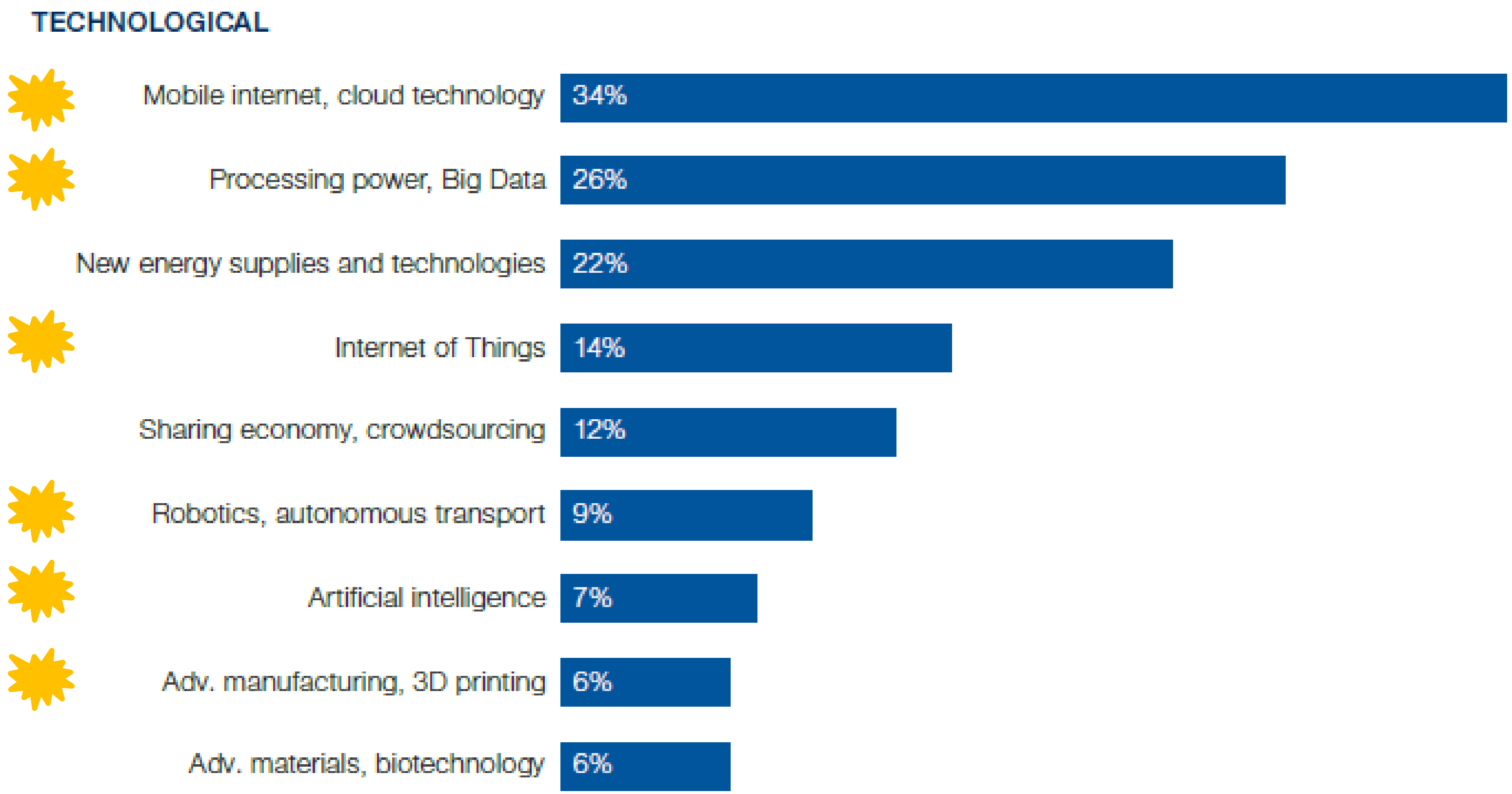


- Error checking?
- Regression tests?
- Reviews?

Remember, computer programs let you find the wrong answer at high speed!

How much money can we lose?

Who isn't going to need software engineering skills?



Source: Future of Jobs Survey, World Economic Forum.
Note: Names of drivers have been abbreviated to ensure legibility.

Computer literacy?

- What about software engineering literacy?

KA/KU	Title	Hours	KA/KU	Title	Hours
CMP	**Computing essentials**	**152**	**DES**	**Software design**	**48**
CMP.cf	Computer science foundations	120	DES.con	Design concepts	3
CMP.ct	Construction technologies	20	DES.str	Design strategies	6
CMP.tl	Construction tools	12	DES.ar	Architectural design	12
			DES.hci	Human-computer interaction design	10
			DES.dd	Detailed design	14
			DES.ev	Design evaluation	3
FND	**Mathematical and engineering fundamentals**	**80**	**VAV**	**Software verification and validation**	**37**
FND.mf	Mathematical foundations	50	VAV.fnd	V&V terminology and foundations	5
FND.ef	Engineering foundations for software	22	VAV.rev	Reviews and static analysis	9
FND.ec	Engineering economics for software	8	VAV.tst	Testing	18
			VAV.par	Problem analysis and reporting	5
PRF	**Professional practice**	**29**	**PRO**	**Software process**	**33**
PRF.psy	Group dynamics and psychology	8	PRO.con	Process concepts	3
PRF.com	Communications skills (specific to SE)	15	PRO.imp	Process implementation	8
PRF.pr	Professionalism	6	PRO.pp	Project planning and tracking	8
			PRO.cm	Software configuration management	6
			PRO.evo	Evolution processes and activities	8
MAA	**Software modeling and analysis**	**28**	**QUA**	**Software quality**	**10**
MAA.md	Modeling foundations	8	QUA.cc	Software quality concepts and culture	2
MAA.tm	Types of models	12	QUA.pca	Process assurance	4
MAA.af	Analysis fundamentals	8	QUA.pda	Product assurance	4
REQ	**Requirements analysis and specification**	**30**	**SEC**	**Security**	**20**
REQ.rfd	Requirements fundamentals	6	SEC.sfd	Security fundamentals	4
REQ.er	Eliciting requirements	10	SEC.net	Computer and network security	8
REQ.rsd	Requirements specification and documentation	10	SEC.dev	Developing secure software	8
REQ.rv	Requirements validation	4			

- Professional Practice
- Requirements analysis and specification
- Software design
- Software V&V
- Software process
- Software quality
- Security

A reflection journal?

- What happened?
- How did you feel about it?
- What do you wish had happened?
- What do you intend to do next time?

References

- The Future of Jobs: Employment, Skills and Workforce Strategy for the Fourth Industrial Revolution, World Economic Forum, Jan. 2016 http://www3.weforum.org/docs/WEF_FOJ_Executive_Summary_Jobs.pdf
- Software Engineering 2014, ACM http://www.acm.org/education/se2014.pdf

Wait! What do Dr. Google and Mr. Wikipedia say?

- **Software engineering** is a field of engineering, for designing and writing programs for computers or other electronic devices. A software engineer, or programmer, writes software (or changes existing software) and compiles software using methods that make it better quality. Better quality software is easier to use, and the code is easier to understand, to maintain, and to add new features. Becoming a software engineer requires university level classes and practice writing code. Software engineering can be very difficult work.[1] Software engineering is often done as part of a team.

- https://simple.wikipedia.org/wiki/Software_engineering